VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

HCM University of Technology
Faculty of Electronics & Electrical Engineering
----------✧----------
No: ……/HCMUT

SOCIALIST REPUBLIC OF VIETNAM
Independence – Freedom – Happiness
----------✧----------

SENIOR PROJECT'S WORKS

Student: VO NGUYEN DUY ID: ILI10192

Major: COMMUNICATING SYSTEM ENGINEERING

Class: CT10TIEN

1. Title of thesis: **IMPLEMENTATION OF AN ORIENTATION ESTIMATION SYSTEM USING KALMAN FILTER**
2. Senior Project's works:
   a) Study on low priced sensors: MPU6050, HMC5883
   b) Study on different approaches to orientation estimation.
   c) Study on Kalman filter.
   d) Study on sensor calibration.
   e) Matlab simulation on PC.
   f) Implementation on ARM cortex processor.
   g) Implementation base station on .NET platform.
3. Date of thesis receiving : Jun 2016
4. Date of thesis finishing : Dec 2016
5. Instructor : Dr. Nguyen Vinh Hao

Content and Requirements of thesis are approved by Head of Division and Reviewer.

Date …… Date ……

HEAD OF DIVISION REVIEWER INSTRUCTOR

(Signature) *(Signature)* *(Signature)*

ABSTRACT

Orientation estimation is a technique commonly used by autonomous vehicles to determine orientation in three-dimensional space. The basic premise of Orientation estimation is that measurements of acceleration and angular velocity from an inertial measurement unit (IMU) are integrated over time to produce estimate of orientation. However, this process is a particularly involved one. The raw inertial data must first be properly analyzed and modeled in order to ensure that any system that uses the data will produce accurate results.

This thesis describes the process of analyzing and modeling raw IMU data, as well as how to use the results of that analysis to design a Kalman filter for an orientation estimation system.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank our instructor Dr. Nguyen Vinh Hao for guiding me through capstone project 1, 2 and senior design project.

Secondly, I would like to thank my family and friends for their unwavering support and the many years preceding it. You have always been confident in my abilities even when I wasn't, and without you this thesis simply would not have been possible.

Thirdly, I would like to express our gratitude to laboratory of electrical machinery for providing me sufficient equipment.

Finally, I am grateful to our friends for sharing ideas and solutions.

HCM city, December 2015

Vo Nguyen Duy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Into orientation estimation

An inertial measurement unit (IMU) is an electronic device that uses a combination of accelerometers and gyroscopes, sometimes also magnetometers. An IMU works by detecting the current rate of acceleration using an accelerometer, detecting changes in rotational attributes like pitch, roll and yaw using a gyroscope, and sometimes also including a magnetometer:

- The gyrometers measure the three angles' velocity of every sampling time of the object body. The only way to derive attitude orientation from gyrometers' data is to take integration of the three angular velocities. However the set of data include its errors is also taken into account too. That is the reason orientation estimation from only gyrometers will drift over time. That means they cannot be trusted for a long period, but they are very precise for a short time.
- The accelerometers are placed such that their measuring axes are orthogonal to each other. They measure inertial acceleration, also known as G-forces. They do not have any drift, but they are too unstable for short period. Another problem of the accelerometers is that the measurement also includes the external force acting on the sensor. This is the most challenging issue in most of estimation approach.
- The magnetometers magnetic field surrounding the body, mostly to assist calibration against orientation drift.

Orientation estimation includes attitude and heading estimation. There are three principle methods for propagation the transforming matrix from the differential form. They are Euler, Direction Cosine and Quaternion, which are discussed in the following chapter. Based on that relationship, many kinds of orientation estimation have been developed and this thesis will mainly focus on Kalman approaches

1.2 Application

The IMU is the main component of inertial navigation systems (INS) used in aircraft, spacecraft, watercraft, drones, UAV and guided missiles among others. In this capacity, the data collected from the IMU's sensors allows a computer to track a craft's position, using a method known as dead reckoning.

However, the set of measurements given by IMU is so both noisy and drifting. Therefore, Kalman filter appears to be the best filter system to combine weaknesses and strength then output accurate results.

The Kalman filter also has numerous applications in technology. A common application is for guidance, navigation and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization.

IMU – Kalman filter can be used in body tracking and gaming such as WII of Nintendo's console:

Figure 1: WII

CHAPTER 2 ORIENTATION ESTIMATION MODELING

2.1 Coordinate Frame

The first step in understanding the inertial navigation algorithm is to understand the various coordinate frames that are used to describe how specific pieces of information are interpreted. The inertial navigation algorithm presented in this thesis uses four coordinate frames: the **Body frame**, the **Navigation frame**, the **Earth frame** and the **Inertial frame**.

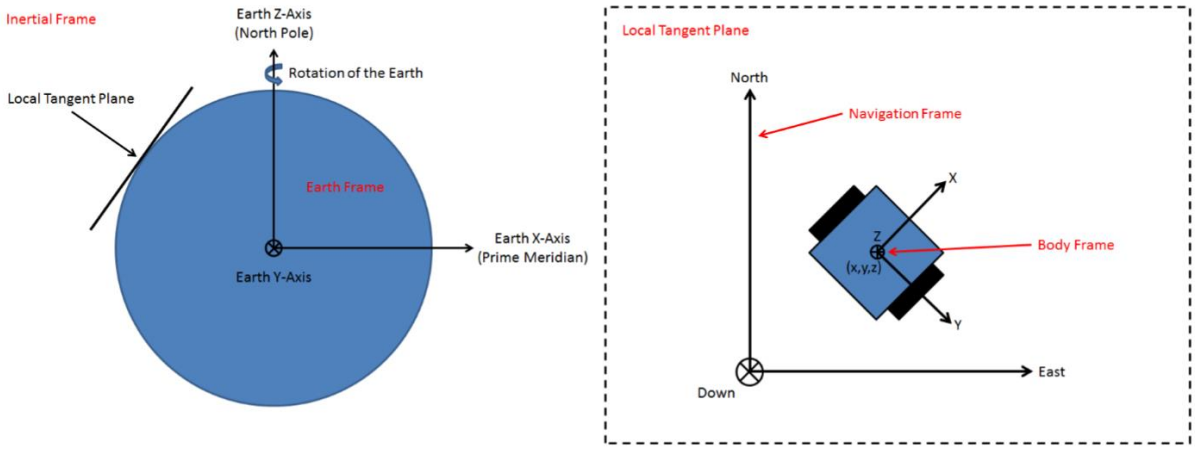


Figure 2: The four coordinate frames used by the inertial navigation algorithm.

The first coordinate frame in the chain is the Body frame. The origin of the Body frame is fixed to the center of the vehicle (or another more applicable reference point) and its axes are oriented such that the X-axis points out through the front of the vehicle, the Z-axis points down through the bottom of the vehicle and the Y-axis points out through the right side of the vehicle. The Body frame translates and rotates with the vehicle as it moves through three-dimensional space, and the origin of the Body frame is used as the reference point for the (x, y, z) position of the vehicle. The Body frame is also used to define the positive acceleration and angular velocity conventions for the system. Positive accelerations are defined as accelerations that occur along the positive axes of the Body frame, and positive angular velocities.

The Navigation frame has the origin initially set at the starting location of the vehicle and its axes are oriented such that the X-axis points due North, the Z-axis points straight down towards the center of the Earth and the Y-axis points due East (this is also commonly referred to as the North, East, Down or NED convention). Unlike the Body frame, the Navigation frame does not rotate with the vehicle. Its orientation is permanently fixed according to the NED convention. The Navigation frame may or may not translate with the vehicle depending on how far the vehicle travels. In order to fully understand how the Navigation frame is defined, the next coordinate frame up the chain must first be introduced.

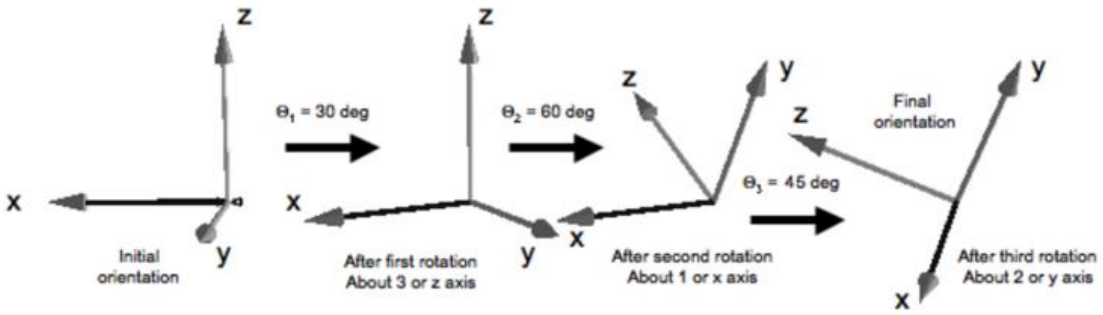
The Earth frame has the origin fixed at the center of the Earth and its axes are oriented such that the X-axis points out of the Earth along the prime meridian, the Z-axis points out of the Earth through the North Pole and the Y-axis is aligned to complete the right-handed orthogonal coordinate system. The Earth frame rotates about its Z-axis along with the Earth (at a rate of approximately fifteen degrees per hour) so that the X axis of the Earth frame is always aligned with the Prime Meridian.

The origin of the Navigation frame remains permanently fixed at the same location on the Earth's surface (the starting location of the vehicle) and does not translate with the vehicle. If the vehicle is navigating over a small enough area, then the curvature of the Earth's surface is so slight that it can be considered negligible. Because the curvature of the Earth's surface is negligible in this scenario, the navigation of the vehicle can be constrained to a single planar surface instead of a curved surface. This means that the position of the vehicle must be represented in terms of spherical coordinates (latitude, longitude, altitude) instead of planar coordinates (NED).

The last coordinate frame in the chain is the Inertial frame. The Inertial frame is a coordinate frame whose origin is fixed at some point in outer space far away from the surface of the Earth.

2.2 Euler Angle

Euler angle system defines the axes of rotation designated as axes 1, 2, and 3 or x, y, and z. The order in which the axes of rotation are taken is referred to as the Euler rotation sequence. There are twelve of these sequences: 1-2-3 (x, y, z), 1-2-1 (x, y, x) and so on including all combinations with no two succeeding rotations about the same axis.



Ambiguity: For small values of Euler angles the Euler Rotation Sequence may not be important. However, for large angles, the rotation sequence becomes critical. For example, for a given set of three Euler angles, the result of a 1-2-3 rotation sequence is very different from that of a 3-2-1 sequence. There is no industry accepted standard rotation sequence; thus, there is an inherent risk of mistaken assumption of rotation sequence in performing analysis and communicating using Euler angles.

Singularities: Any set of Euler angles where the second rotation aligns the axes of the first and third rotations causes a singularity. For an Euler Rotation sequence where the first and third axes are the same, called a repeated axis sequence, singularities occur for second rotation angles of zero and 180 degrees; for non-repeated axis sequences singularities occur at +/- 90 degrees. The mechanical manifestation of this mathematical singularity is the dreaded "gimbal lock.

There is two way to compute Euler angle from accelerometer data $a = (a_x \quad a_y \quad a_z)$. The first is which follow (x, y, z) rotate sequence:

$$\phi_{xyz} = a\tan\left(\frac{a_x}{a_z}\right)$$

$$\theta_{xyz} = a\tan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \qquad \text{Eq. 1}$$

And the second approach is refer to (y, x, z) rotate sequence:

$$\phi_{yxz} = a\tan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

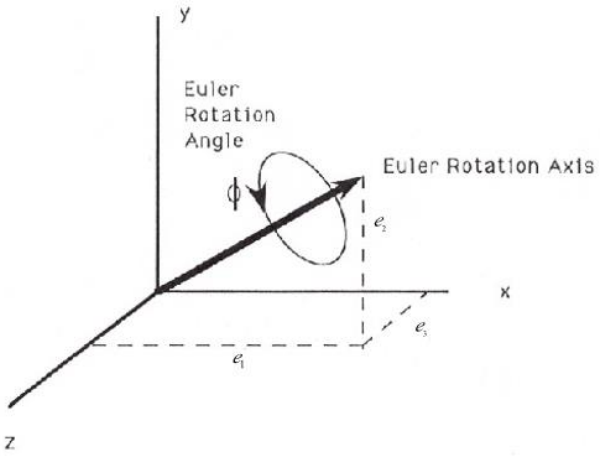
$$\theta_{yxz} = a\tan\left(\frac{-a_x}{a_z}\right) \qquad \text{Eq. 2}$$

Finally, **tilt compensating method** introduce to combine the archived angles to find the yaw angle directly:

$$\begin{aligned} X_h &= m_x.\cos(\theta) + m_y.\sin(\theta).\sin(\phi) + m_z.\cos(\theta).\cos(\theta) \\ Y_h &= m_y.\cos(\phi) + m_z.\sin(\phi) \\ \psi &= \arctan 2(X_h, Y_h) \end{aligned} \qquad \text{Eq. 3}$$

2.3 Quaternion

Quaternions provide an alternative measurement technique that does not suffer from gimbal lock. Quaternions are less intuitive than Euler Angles and the math can be a little more complicated. This section will ignore the theoretical details about quaternions and providing only the information that is needed to use them for representing the attitude of an orientation sensor.



A quaternion is a four-element vector that can be used to encode any rotation in a 3D coordinate system and can be used for much more than rotations:

- One real element.
- Three complex elements.

A definition of a quaternion is:

$$Q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \sin(0.5\theta) \\ e_3 \sin(0.5\theta) \\ \cos(0.5\theta) \sin(0.5\theta) \end{pmatrix} \quad \text{Eq. 4}$$

The four elements are not independent, being related by the requirement that:

$$\left| Q \right| = q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \quad \text{Eq. 5}$$

Example:

$$\mathrm{Q} = (0.360423 \quad 0.439679 \quad 0.391904 \quad 0.723317)^T$$

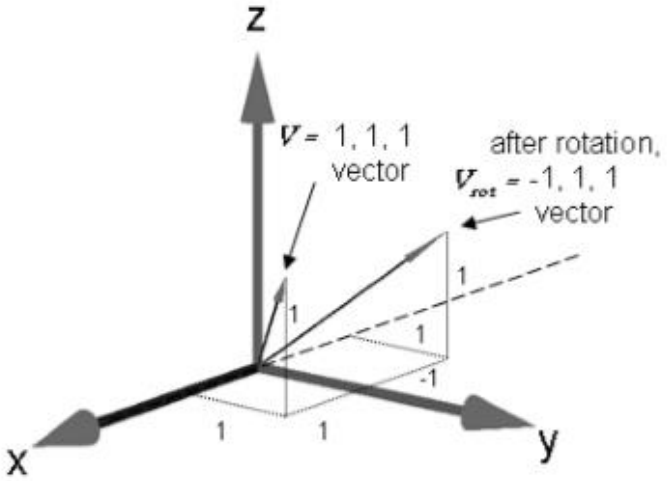
$$\begin{cases} \hat{e} = (0.5220 \quad 0.6367 \quad 0.5676)^T \\ \phi = 43.671^o \end{cases} \qquad \text{Eq. 6}$$

Quaternions product is defined as:

$$q_1 q_2 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \end{pmatrix} \begin{pmatrix} a_2 \\ b_2 \\ c_2 \\ d_2 \end{pmatrix} = \begin{pmatrix} a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \\ a_1 a_2 + b_1 b_2 + c_1 c_2 - d_1 d_2 \\ a_1 a_2 - b_1 b_2 + c_1 c_2 + d_1 d_2 \\ a_1 a_2 + b_1 b_2 - c_1 c_2 + d_1 d_2 \end{pmatrix} \qquad \text{Eq. 7}$$

A vector can be rotated about a given axis by a given angle using the quaternion generated using that axis and angle:

$$V_{rot} = QRV = Q \otimes V \otimes Q^* \qquad \text{Eq. 8}$$



As an example, the vector $(1 \quad 1 \quad 1)$ before and after being rotated by the quaternion $(0 \quad 0 \quad 0.707107 \quad 0.707107)^T$ which represents a 90 degree rotation about the Z, or 3, axis; the resultant rotated vector is $(-1 \quad 1 \quad 1)$.

2.4 Direction Cosine Matrix

DCM method is chosen and discussed deeply. Some techniques are addressed to reduce the number of updated variables from nine variables in conventional to three in the attitude estimation and six in the orientation estimation. Measurement Unit in which the accelerometers, gyrometers and magnetic compass are sampled at the same frequency, so direct filter has been used.

In order to combine magnetic compass to the estimator, there also some methods are applied such as using directly measure vector as measurement, calculating yaw angles and use as measurement, tilt compensating or automatically magnetic vector fixing method.

The ${}^{E}_{B}R$ is the rotation matrix that rotate a vector measured in the frame of reference of the body {B} to the frame of reference of the Earth {E}:

$$ {}^{E}_{B}R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} = \left({}^{E}_{B}X \; {}^{E}_{B}Y \; {}^{E}_{B}Z \right) = $$

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

Eq. 9

Assume ${}^{B}v \; and \; {}^{E}v$ are two vectors in the frame {B} and frame {E}, we have the relation:

$$ {}^{E}v = {}^{E}_{B}R.{}^{B}v \qquad \text{Or} \qquad {}^{B}v = {}^{E}_{B}R^{-1}.{}^{E}v = {}^{E}_{B}R^{T}.{}^{E}v \qquad \text{Eq. 10} $$

We also have the following characteristics:

$$ \begin{cases} \left\| X \right\| = \left\| Y \right\| = \left\| Z \right\| = 1 \\ X.\mathrm{Y}^{T} = \mathrm{Y}.\mathrm{Z}^{T} = \mathrm{Z}.\mathrm{X}^{T} = 0 \\ R.R^{T} = 1 \end{cases} \qquad \text{Eq. 11} $$

The transformation matrix $\dot{R}$ can be obtained with the following angular rates integration:

$$ \begin{aligned} {}^{E}_{B}\dot{R} &= {}^{E}_{B}R.\;{}^{B}\omega \\ &= \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}.\begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \end{aligned} \qquad \text{Eq. 12} $$

Where $\overrightarrow{{}^{B}\omega} = \begin{pmatrix} {}^{B}\omega_x & {}^{B}\omega_y & {}^{B}\omega_z \end{pmatrix}$ is the angular rate measured by gyro in body frame {B}.

However, thanks to orthogonal characteristic, only three states need to be updated in each circle:

$$\begin{pmatrix} \dot{R}_{21} \\ \dot{R}_{22} \\ \dot{R}_{23} \end{pmatrix} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} . \begin{pmatrix} R_{21} \\ R_{22} \\ R_{23} \end{pmatrix}$$
$$\begin{pmatrix} \dot{R}_{31} \\ \dot{R}_{32} \\ \dot{R}_{33} \end{pmatrix} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} . \begin{pmatrix} R_{31} \\ R_{32} \\ R_{33} \end{pmatrix} \quad \text{Eq. 13}$$

We also have measurements model using accelerometer data $B_a = ({}^{B}a_x \quad {}^{B}a_y \quad {}^{B}a_z)$:

$$\begin{pmatrix} {}^{B}a_x \\ {}^{B}a_y \\ {}^{B}a_z \end{pmatrix} = {}^{E}_{B}R^{T} . \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} R_{31} \\ R_{32} \\ R_{33} \end{pmatrix} . g \quad \text{Eq. 14}$$

While g is gravitational acceleration.

Finally tilt compensating method are used to combine the archived angles to find the yaw angle directly:

$$\begin{aligned} X_h &= m_x . \cos(\theta) + m_y . \sin(\theta) . \sin(\phi) + m_z . \cos(\theta) . \cos(\theta) \\ Y_h &= m_y . \cos(\phi) + m_z . \sin(\phi) \\ \psi &= \arctan 2(Y_h, X_h) \end{aligned} \quad \text{Eq. 15}$$

2.5 Conversion

Where ϕ is roll, θ is pitch, ψ is yaw, $q = \begin{matrix} q_0 & q_1 & q_2 & q_3 \end{matrix}$, the Quaternion – Euler conversion can be performed as:

$$\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} a\tan 2(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0 q_2 + q_3 q_1) \\ a\tan 2(2(q_2 q_3 + q_2 q_1), 1 \text{ - } 2(q_2^2 + q_3^2) \end{pmatrix} \qquad \text{Eq. 16}$$

The reverse process, the Euler – Quaternion conversion can be done as below:

$$\begin{pmatrix} q_0 \\ q_2 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \end{pmatrix} \qquad \text{Eq. 17}$$

Euler angle can be also obtained from DCM matrix:

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \qquad \text{Eq. 18}$$

The DCM to Euler angle can be achieved by:

$$\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} -a\sin(R_{31}) \\ a\tan 2(R_{32}, R_{33}) \\ -a\tan 2(R_{21}, R_{11}) \end{pmatrix} \qquad \text{Eq. 19}$$

2.6 Kalman filter

> *The Kalman filter in its various forms is clearly established as a fundamental tool for analyzing and solving a broad class of estimation problems.*
>
> *-Leonard McGee and Stanley Schmidt*

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization.

This part will form the foundation for Kalman filter understanding by propagating the mean and covariance of the state through time. Our approach to deriving the Kalman filter is first of all starting with a mathematical description of a dynamic system whose states we want to estimate. Finally, equations that describe how the mean of the state and the covariance of the state propagate with time will be implemented to form a dynamic system.

Suppose we have a linear discrete-time system given as follows:

$$\begin{aligned} x_k &= F_{k-1} x_{k-1} + G_{k-1} u_{k-1} + w_{k-1} \\ y_k &= H_k x_k + v_k \end{aligned} \qquad \text{Eq. 20}$$

The noise processes w_k and v_k are white, zero-mean, uncorrelated, and have known covariance matrices Q_k and R_k respectively:

$$\begin{aligned} & w_k \sim (0, \mathrm{Q}_k) \\ & v_k \sim (0, \mathrm{R}_k) \\ & \mathrm{E}[\mathrm{w}_k \, w_j^T] = \mathrm{Q}_k \, \delta_{k-1} \qquad \begin{matrix} \delta_{k-1} = 1, k = j \\ \delta_{k-1} = 0, k \neq j \end{matrix} \\ & \mathrm{E}[\mathrm{v}_k \, v_j^T] = R_k \delta_{k-1} \\ & \mathrm{E}[\mathrm{v}_k \, w_j^T] = 0 \end{aligned} \qquad \text{Eq. 21}$$

Our goal: is to estimate x_k based on our knowledge of the system dynamics and the availability of the noisy measurements y_k.

Posteriori: compute the expected value of conditioned on all of the measurements up to and including time k:

$$\hat{x}_k^+ = E[\mathrm{x}_k \mid y_1, y_2, \ldots, y_{\mathrm{k}}] \qquad \text{Eq. 22}$$

Priori: compute the expected value of x_k conditioned on all of the measurements before (but not including) time k:

$$\hat{x}_k^- = E[\mathrm{x}_k \mid y_1, y_2, \ldots, y_{\mathrm{k}-1}] \qquad \text{Eq. 23.}$$

Smooth: compute the expected value of x_k conditioned on available measurements:

$$\hat{x}_{k|k+N} = E[\mathrm{x}_k \mid y_{1,} y_{2,} ..., y_{\mathrm{k}}, ..., y_{\mathrm{k+N}}] \quad \text{Eq. 24.}$$

Predicted: compute the expected value of x_k conditioned on all of the measurements before (but not including) time k:

$$\hat{x}_{k|k-M} = E[\mathrm{x}_k \mid y_{1,} y_{2,} ..., y_{\mathrm{k}}, ..., y_{\mathrm{k-M}}] \quad \text{Eq. 25}$$

Covariance: We use the term P_k to denote the covariance of the estimation error. P_k^- denotes the covariance of the estimation error of $\hat{x}_k^-$, and P_k^+ denotes the covariance of the estimation error of $\hat{x}_k^+$:

$$\begin{aligned} P_k^- &= E[(\mathrm{x}_k - \hat{x}_k^-)(\mathrm{x}_k - \hat{x}_k^-)^T] \\ P_k^+ &= E[(\mathrm{x}_k - \hat{x}_k^+)(\mathrm{x}_k - \hat{x}_k^+)^T] \end{aligned} \quad \text{Eq. 26}$$

Relationship: Time line showing the relationship between the a posteriori, a priori, smoothed, and predicted state estimates. In this figure, we suppose that we have received measurements at times up to and including k = 5. An estimate of the state at k < 5 is called a smoothed estimate. An estimate of the state at k = 5 is called the posteriori estimate. An estimate of the state at k = 6 is called the a priori estimate. An estimate of the state at k > 6 is called the prediction.

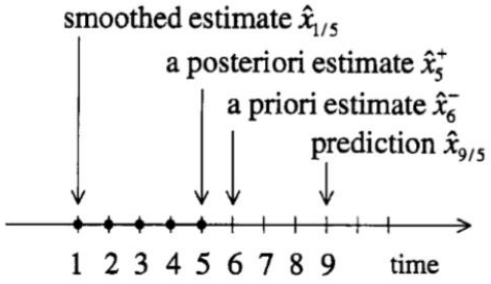


Figure 3: Kalman time line 1

The figure shows that after we process the measurement at time (k-1), we have an estimate of x_{k-1} (denoted $\hat{x}_{k-1}^{+}$) and the covariance of that estimate (denoted P_{k-1}^{+}). When time k arrives, before we process the measurement at time k we compute an estimate of x_k (denoted $\hat{x}_k^{-}$) and the covariance of that estimate (denoted P_k). Then we process the measurement at time k to refine our estimate of x_k. The resulting estimate x_k is denoted $\hat{x}_k^{+}$, and its covariance is denoted P_k^{+}.

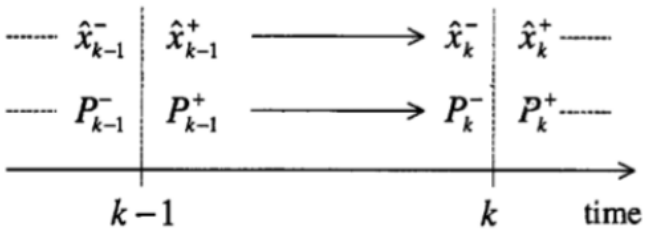


Figure 4: Kalman time line 2

Consider the dynamic system is given by:

$$\begin{aligned} x_k &= F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \\ y_k &= H_k x_k + v_k \\ E(w_k w_j^T) &= Q_k \delta_{k-j} \\ E(\mathrm{v}_k\ v_j^T) &= R_k \delta_{k-j} \\ E(w_k v_j^T) &= 0 \end{aligned} \qquad \text{Eq. 27}$$

The Kalman filter is initialized as follows:

$$\begin{aligned} \hat{x}_0^+ &= E(x_0) \\ P_0^+ &= E[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T] \end{aligned} \qquad \text{Eq. 28}$$

The Kalman filter is given by the following equations, which are computed for each time step k = 1, 2, …:

$$\begin{aligned} P_k^- &= F_{k-1} P_{k-1}^+ F_{k-1}^T + Q_{k-1} && \text{(i)} \\ K_k &= P_k^- K_k^T (\mathrm{H}_k P_k^- H_k + R_k)^{-1} && \text{(ii)} \\ &= P_k^+ \mathrm{H}_k R_k \\ \hat{x}_k^- &= F_{k-1} \hat{x}_{k-1} + G_{k-1} u_{k-1} && \text{(iii)} \\ \hat{x}_k^+ &= x_k^- + K_k (\mathrm{y}_k - H_k \hat{x}_k^-) && \text{(iv)} \\ P_k^+ &= (\mathrm{I} - K_k H_k) P_k^- (I + K_k H_k)^T + K_k R_k H_k^T && \text{(v)} \\ &= [(P_k^-)^{-1} + \mathrm{H}_k^T R_k^{-1} H_k] \\ &= (\mathrm{I} - K_k H_k) P_k^- \end{aligned}$$

Eq. 29

- The expression (i) is the covariance measurement update equation, which is more stable and robust than the expression (v) for P_k^+.
- The first form in (v) for P_k^+ guarantees that P_k^+ will always be symmetric positive definite, as long as P_k^- is symmetric positive definite.
- The third form in (v) for P_k^+ is computationally simpler than the first one, but its form does not guarantee symmetry or positive definiteness for P_k^+.
- The second form for P_k^+ is rarely implemented.

- If the second expression for K_k is used, then the second form in (v) for P_k^+ must be used. This is because the second expression for K_k depends on P_k^+, so we need to use an expression for P_k^+ that does not depend on K_k.
- The calculation of P_k^-, K_k, and P_k^+ does not depend on the measurements y_k, but depends only on the system parameters F_k, H_k, Q_k, and R_k. That means that the Kalman gain K_k can be calculated offline before the system operates and saved in memory. Then when it comes time to operate the system in real time, only the $\hat{x}_k$ equations need to be implemented in real time. The computational effort of calculating K_k can be saved during real-time operation by pre-computing it.
- Furthermore, the performance of the filter can be investigated and evaluated before the filter is actually run. This is because P_k indicates the estimation accuracy of the filter, and it can be computed offline since it does not depend on the measurements.

2.7 Quaternion-based EKF

This approach is focus on quaternion, which use four elements to represent the orientation of the sensor. The modeling is non-linear, so linearized and discretized process is necessary to apply Kalman filter. The whole processes, therefore is called extended-Kalman filter (EKF).

Following is the flowchart of quaternion-based EKF system:

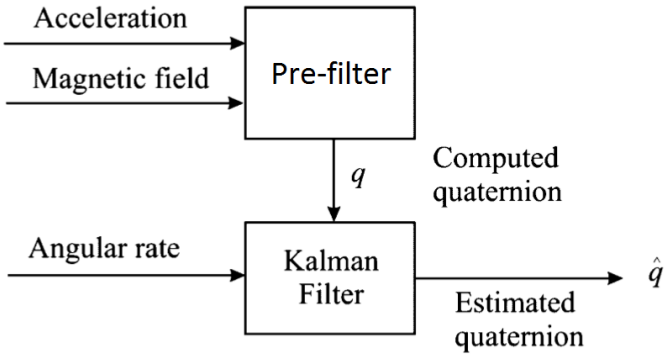


Figure 5: Quaternion-based EKF flowchart

The pre-filter firstly converts accelerometer data $a = (a_x \quad a_y \quad a_z)$ into pitch (θ) and roll (ϕ) in Euler system. Next, it combines the calculated angles and magnetometer data to obtain the yaw (ψ) by tilt-compensate method.

The (x, y, z) rotate sequence:

$$\phi_{xyz} = a\tan\left(\frac{a_x}{a_z}\right)$$
$$\theta_{xyz} = a\tan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \qquad \text{Eq. 30}$$

The (y, x, z) rotate sequence:

$$\phi_{yxz} = a\tan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$
$$\theta_{yxz} = a\tan\left(\frac{-a_x}{a_z}\right) \qquad \text{Eq. 31}$$

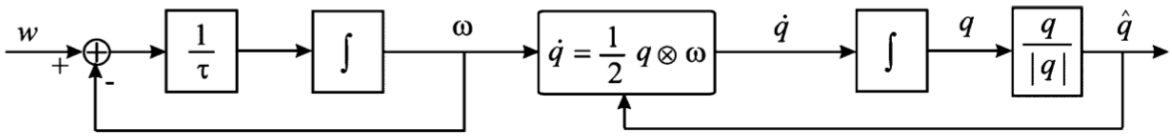
Then:

$$\begin{aligned} X_h &= m_x.\cos(\theta) + m_y.\sin(\theta).\sin(\phi) + m_z.\cos(\theta).\cos(\theta) \\ Y_h &= m_y.\cos(\phi) + m_z.\sin(\phi) \\ \psi &= \arctan 2(X_h, Y_h) \end{aligned} \quad \text{Eq. 32}$$

Finally, three Euler angles are converted to quaternion system to feed into the Kalman model by the conversion:

$$\begin{pmatrix} q_0 \\ q_2 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \end{pmatrix} \quad \text{Eq. 33}$$

In quaternion-based EKF model, the angular rates ω in body coordinates are assumed to be generated by a first-order linear system with a white noise forcing the function w. The time constant of the first-order linear system is τ. The orientation estimate produced by the filter is $\hat{q}$. The angular rates ω and the quaternion derivative $\dot{q}$ are related by:



Where the orientation quaternion $\dot{q}$ is in Earth coordinates, and $\otimes$ represents quaternion multiplication. The quaternion is normalized to unit length in the last step of the process model. It is noted that quaternions are used to represent orientation in the filter design because quaternions do not have the singularity problem associated with Euler angles and eliminate the computational expenses related to approximation of transcendental functions.

The state vector is defined as a 7-dimensional vector:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} = \begin{pmatrix} \text{angular rate } (roll) \\ \text{angular rate } (pitch) \\ \text{angular rate (yaw)} \\ \text{quaternion q}_1 \\ \text{quaternion q}_2 \\ \text{quaternion q}_3 \\ \text{quaternion q}_4 \end{pmatrix} \qquad \text{Eq. 34}$$

The state equations are given by:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \end{pmatrix} = \begin{pmatrix} \frac{1}{\tau}\left(-\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} w_1(\mathrm{t}) \\ w_2(\mathrm{t}) \\ w_3(\mathrm{t}) \end{bmatrix}\right) \\ \frac{1}{2}\begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \end{pmatrix} = \begin{pmatrix} -\frac{1}{\tau}x_1 + \frac{1}{\tau}w_1(\mathrm{t}) \\ -\frac{1}{\tau}x_2 + \frac{1}{\tau}w_2(\mathrm{t}) \\ -\frac{1}{\tau}x_3 + \frac{1}{\tau}w_3(\mathrm{t}) \\ \frac{1}{2}(x_1x_5 + x_2x_6 + x_3x_7) \\ \frac{1}{2}(x_1x_4 + x_2x_7 + x_3x_6) \\ \frac{1}{2}(x_1x_7 + x_2x_4 + x_3x_5) \\ \frac{1}{2}(x_1x_6 + x_2x_5 + x_3x_4) \end{pmatrix}$$

Eq. 35

And we want it to be linear and discrete to apply Kalman filter:

$$\begin{cases} x_{k+1} = \Phi x_k + w_k \\ z_{k+1} = H_{k+1}x_{k+1} + v_{k+1} \end{cases} \qquad \text{Eq. 36}$$

It can be rewrite in non-linear form $\dot{x}=f(x)+w(t)$ as:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\\ \dot{x}_7\end{pmatrix} = \begin{pmatrix} -\frac{1}{\tau}x_1 \\ -\frac{1}{\tau}x_2 \\ -\frac{1}{\tau}x_3 \\ 0.5(x_1x_5+x_2x_6+x_3x_7) \\ 0.5(x_1x_4+x_2x_7+x_3x_6) \\ 0.5(x_1x_7+x_2x_4+x_3x_5) \\ 0.5(x_1x_6+x_2x_5+x_3x_4)\end{pmatrix} + \begin{pmatrix}\frac{1}{\tau}w_1(\mathrm{t})\\ \frac{1}{\tau}w_2(\mathrm{t})\\ \frac{1}{\tau}w_3(\mathrm{t})\\ 0\\ 0\\ 0\\ 0\end{pmatrix} \qquad \text{Eq. 37}$$

System can be linearized as $\dot{x}=A\tilde{x}+w(\mathrm{t})$ where:

$$A = \frac{\partial f(x)}{\partial x}$$

$$= \begin{pmatrix} -\frac{1}{2}\tau & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{2}\tau & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}\tau & 0 & 0 & 0 & 0 \\ -\frac{1}{2}x_5 & -\frac{1}{2}x_6 & -\frac{1}{2}x_7 & 0 & -\frac{1}{2}x_1 & -\frac{1}{2}x_2 & -\frac{1}{2}x_3 \\ \frac{1}{2}x_4 & -\frac{1}{2}x_7 & \frac{1}{2}x_6 & \frac{1}{2}x_1 & 0 & \frac{1}{2}x_3 & -\frac{1}{2}x_2 \\ \frac{1}{2}x_7 & \frac{1}{2}x_4 & -\frac{1}{2}x_5 & \frac{1}{2}x_2 & -\frac{1}{2}x_3 & 0 & \frac{1}{2}x_1 \\ -\frac{1}{2}x_6 & \frac{1}{2}x_5 & \frac{1}{2}x_4 & \frac{1}{2}x_3 & \frac{1}{2}x_2 & -\frac{1}{2}x_1 & 0 \end{pmatrix}$$

Eq. 38

Next, it can be discretized as $x_{k+1} = \Phi x_k + w_k$ where:

$$\Phi = \begin{bmatrix} e^{A\Delta T} & (\Phi_{1x,2x,3x}) \\ I + A\Delta T & (\Phi_{4x,5x,6x,7x}) \end{bmatrix} =$$

$$\begin{pmatrix} e^{-\frac{\Delta T}{\tau}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{-\frac{\Delta T}{\tau}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{-\frac{\Delta T}{\tau}} & 0 & 0 & 0 & 0 \\ -\frac{\Delta T}{2}x_5 & -\frac{\Delta T}{2}x_6 & -\frac{\Delta T}{2}x_7 & 1 & -\frac{\Delta T}{2}x_1 & -\frac{\Delta T}{2}x_2 & -\frac{\Delta T}{2}x_3 \\ \frac{\Delta T}{2}x_4 & -\frac{\Delta T}{2}x_7 & \frac{\Delta T}{2}x_6 & \frac{\Delta T}{2}x_1 & 1 & \frac{\Delta T}{2}x_3 & -\frac{\Delta T}{2}x_2 \\ \frac{\Delta T}{2}x_7 & \frac{\Delta T}{2}x_4 & -\frac{\Delta T}{2}x_5 & \frac{\Delta T}{2}x_2 & -\frac{\Delta T}{2}x_3 & 1 & \frac{\Delta T}{2}x_1 \\ -\frac{\Delta T}{2}x_6 & \frac{\Delta T}{2}x_5 & \frac{\Delta T}{2}x_4 & \frac{\Delta T}{2}x_3 & \frac{\Delta T}{2}x_2 & -\frac{\Delta T}{2}x_1 & 1 \end{pmatrix}$$

Eq. 39

From now, we can apply Kalman filter by following steps:

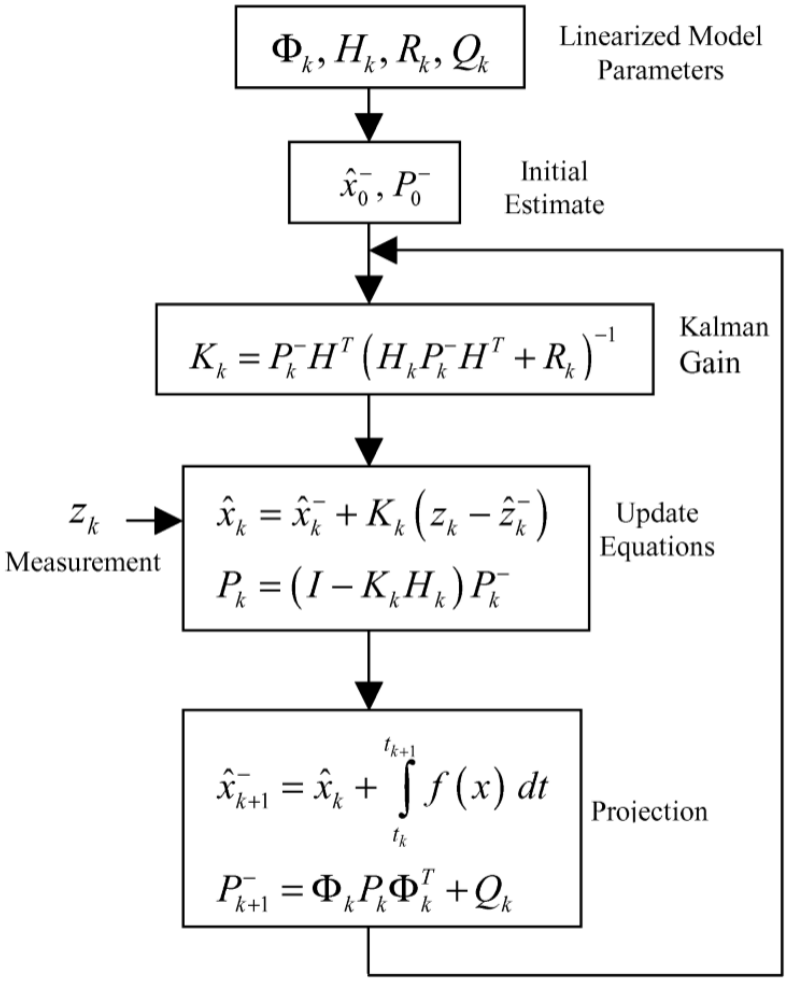


Figure 6: Kalman filter phases

Where:

- Φ_k (7x7) : the discrete state transition matrix.
- H_k (7x7) : the measurement equation matrix which is (7x7) identity.
- Q_k (7x7) : The process noise covariance matrix.
- R_k (7x7) : The measurement noise covariance matrix.
- $\hat{x}_k$ (7x1) : posteriori estimate matrix.
- P_k (7x7) : posteriori covariance matrix.
- $\hat{x}_{k+1}^-$ (7x1) : priori estimate matrix.
- P_{k+1}^- (7x7) : priori covariance matrix.

2.8 DCM-based KF

In this modeling, instead of 7 states, it reduce the number of state to three. KF1 estimate the state from gyrometer data and compare with the measurement from accelerometer data to correct pitch and yaw angle in term of DCM. After that, the KF1's result combines with magnetometer data to correct yaw angle.

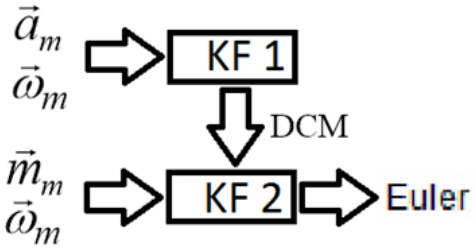


Figure 7: Quaternion-based EKF flowchart

The system for each filter can be described as:

$$\begin{cases} x_{k+1} = \Phi x_k + w_k \\ z_{k+1} = H_{k+1} x_{k+1} + v_{k+1} \end{cases} \quad \text{Eq. 40}$$

The discrete state transition matrix for both filters to update estimate state from gyrometer data is:

$$\Phi = I + \omega\otimes .\Delta T$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} . \Delta T \quad \text{Eq. 41}$$

$$= \begin{pmatrix} 1 & -\Delta T\omega_z & \Delta T\omega_y \\ \Delta T\omega_z & 1 & -\Delta T\omega_x \\ -\Delta T\omega_y & \Delta T\omega_x & 1 \end{pmatrix}$$

Where the state vector of KF 1 is:

$$x_k = \begin{matrix} R_{31} & R_{32} & R_{33} \end{matrix}^T \quad \text{Eq. 42}$$

The measurement vector is $z_k = \begin{pmatrix} \frac{a_x}{g} & \frac{a_y}{g} & \frac{a_z}{g} \end{pmatrix}$, while H_k is an identity matrix.

Where the state vector KF 2 is:

$$x_k = \begin{matrix} R_{21} & R_{22} & R_{23} \end{matrix}^T \quad \text{Eq. 43}$$

The measurement vector can be derived from EKF 1 estimation as following:

$$\begin{cases} \begin{pmatrix} DCM_1 \\ DCM_2 \\ DCM_3 \end{pmatrix} = \begin{pmatrix} R_{31} \\ R_{32} \\ R_{33} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta.\sin\phi \\ \cos\phi.\cos\theta \end{pmatrix} \\ X_h = m_x.\cos(\theta)+m_y.\sin(\theta).\sin(\phi)+m_z.\cos(\theta).\cos(\theta) \\ Y_h = m_y.\cos(\phi)+m_z.\sin(\phi) \\ \psi = \arctan 2(Y_h, X_h) \end{cases} \quad \text{Eq. 44}$$

$$\Rightarrow \begin{cases} \sin\theta & = & -DCM_1 \\ \cos\theta & = & \sqrt{1-DCM_1{}^2} \\ \sin\phi & = & DCM_2 \Big/ \sqrt{1-DCM_1{}^2} \\ \cos\phi & = & DCM_3 \Big/ \sqrt{1-DCM_1{}^2} \\ \sin\psi & = & -Y_h \Big/ (X_h{}^2 + Y_h{}^2) \\ \cos\psi & = & X_h \Big/ (X_h{}^2 + Y_h{}^2) \end{cases} \qquad \text{Eq. 45}$$

$$\Rightarrow \quad z_k \quad = \quad \begin{pmatrix} R_{21} \\ R_{22} \\ R_{23} \end{pmatrix} \quad = \quad \begin{pmatrix} \sin\psi.\cos\theta \\ \sin\phi.\sin\theta.\sin\psi + \cos\phi.\sin\psi \\ \sin\phi.\sin\theta.\sin\psi - \cos\psi.\sin\phi \end{pmatrix}$$

CHAPTER 3 IMPLEMENTATION

3.1 Matlab simulation

This section will show the simulation of 2 method for quaternion-based EKF's pre-filters, which are **Tilt-compensate** and **Gauss Newton method**. **AHRS**, which is a well-known and open sourced filter, will also be simulated for comparison. Quaternion based EKF simulation will be showed in 2 different approaches, the first one is **Quaternion based – gyro bias – EKF** and the second one is **Quaternion based – gyro rate – EKF.** The final filter is **DCM based KF**, which is the best among all algorithms interpreted.

All the algorithms will be simulated in 5 different testing circumstances including:

1. **Still.**
2. **Free move.**
3. **Free move in magnetic field.**
4. **Moving along x axis.**
5. **Turning around z axis.**

And they run on the same machine:

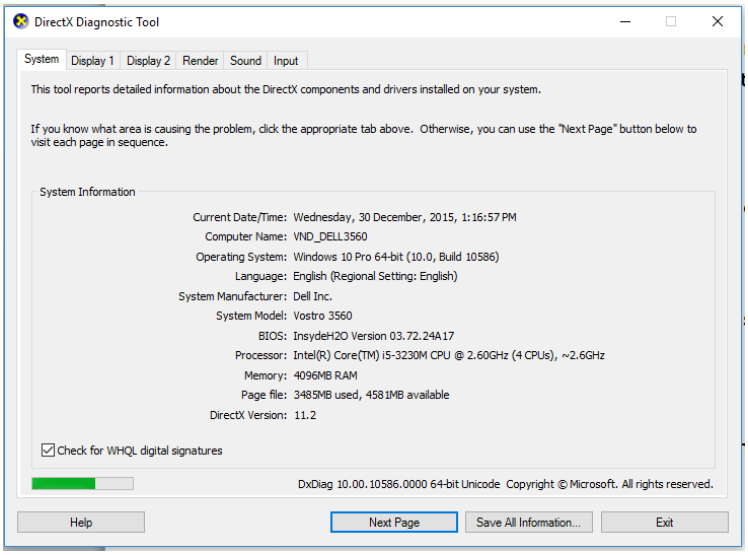


Figure 8: PC performance

In the first three graphs for each case, **the blue lines** show estimate results for the roll, pitch and yaw, compared to **the red lines** given by Xsens, which is the leading innovator in 3D motion tracking technology and products. Its sensor fusion technologies enable a seamless interaction between the physical and the digital world in consumer devices and professional applications such as 3D character animation, motion analysis, and industrial control & stabilization.

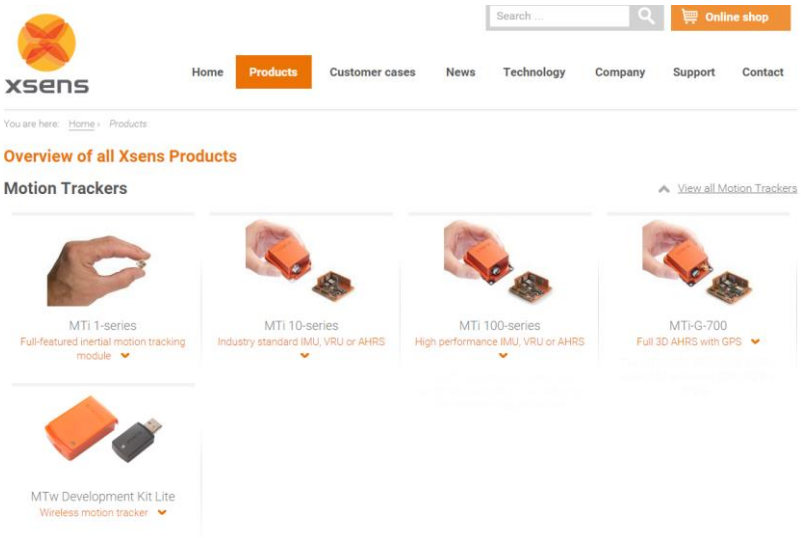


Figure 9: Xsens's website

The last three graphs show the estimation errors compared to Xsens data in RMS.

3.1.1 Tilt-compensation Method

Tilt-compensation is the method can be used as a pre-filter to feed into a quaternion based EKF. It is fast computed, specifically 43.87 micro-second per cycle. The simulation results are showed as below:

Tilt-compensation method RMS					
Conditions	**1**	**2**	**3**	**4**	**5**
Roll	0.06	5.88	6.58	1.57	1.38
Pitch	0.08	2.08	3.01	7.41	0.72
Yaw	0.33	7.95	15.51	6.91	14.50

Table 1: Tilt-compensation method RMS.

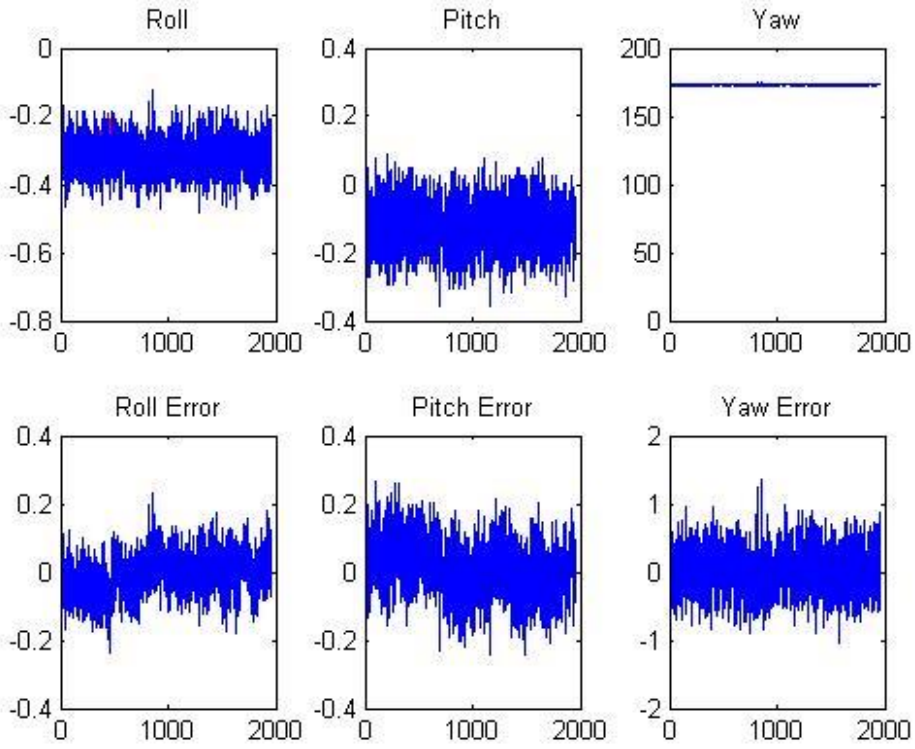


Figure 10: Simulation result for Tilt-compensation on condition 1.

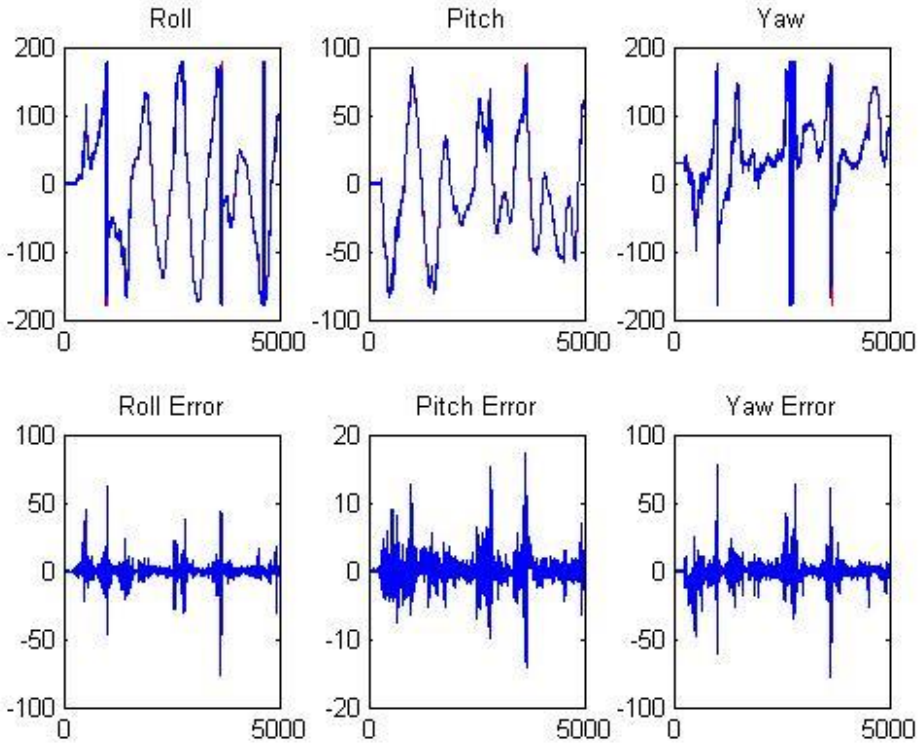


Figure 11: Simulation result for Tilt-compensation on condition 2.

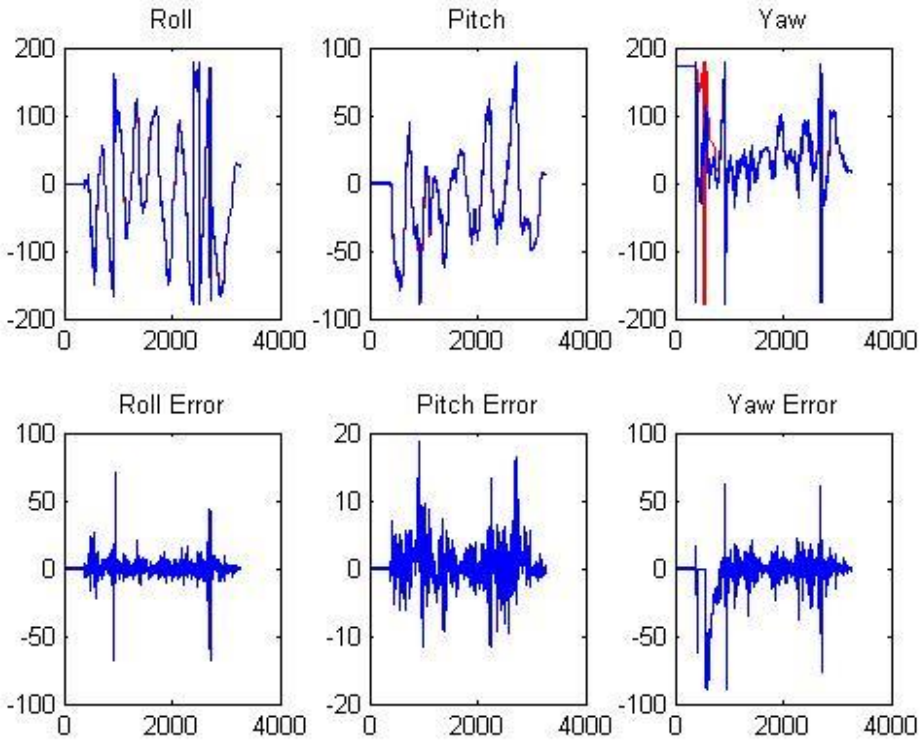


Figure 12: Simulation result for Tilt-compensation on condition 3.

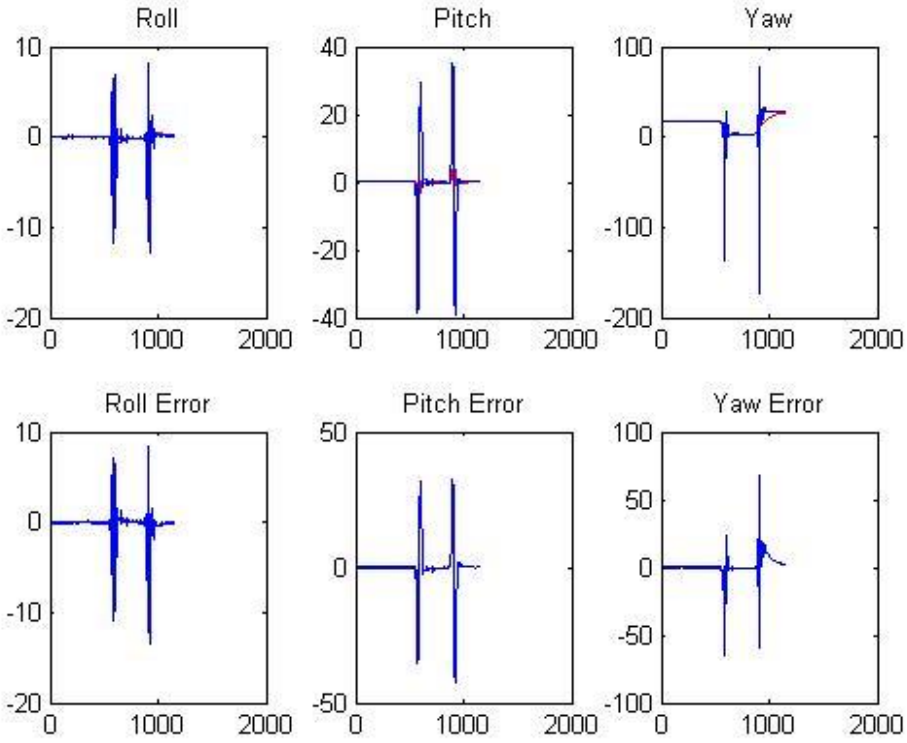


Figure 13: Simulation result for Tilt-compensation on condition 4.

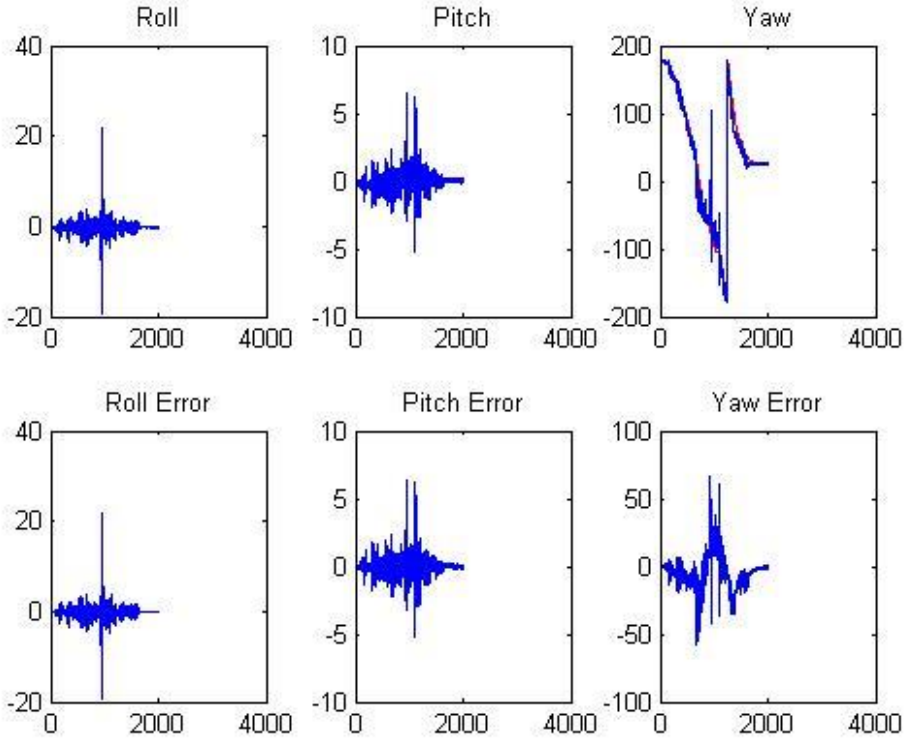


Figure 14: Simulation result for Tilt-compensation on condition 5.

3.1.2 Gauss Newton Method

Gauss Newton method can be also used as a pre-filter to feed into a quaternion based EKF. It is slower computed than Tilt-compensation method, with 319.64 micro-second per cycle. The simulation results are showed as below:

Gauss Newton method RMS					
Conditions	**1**	**2**	**3**	**4**	**5**
roll	0.045	2.081	6.432	1.597	0.966
pitch	0.045	1.048	3.021	7.273	0.554
yaw	0.927	2.747	15.486	7.874	14.158

Table 2: Gauss Newton method RMS

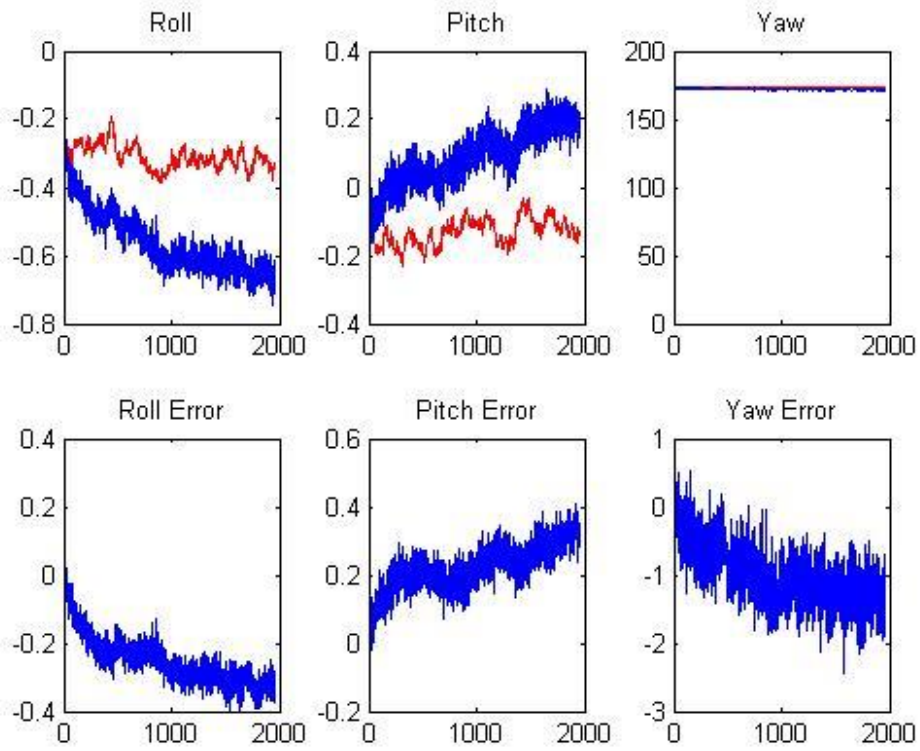


Figure 15: Simulation result for Gauss Newton method on condition 1.

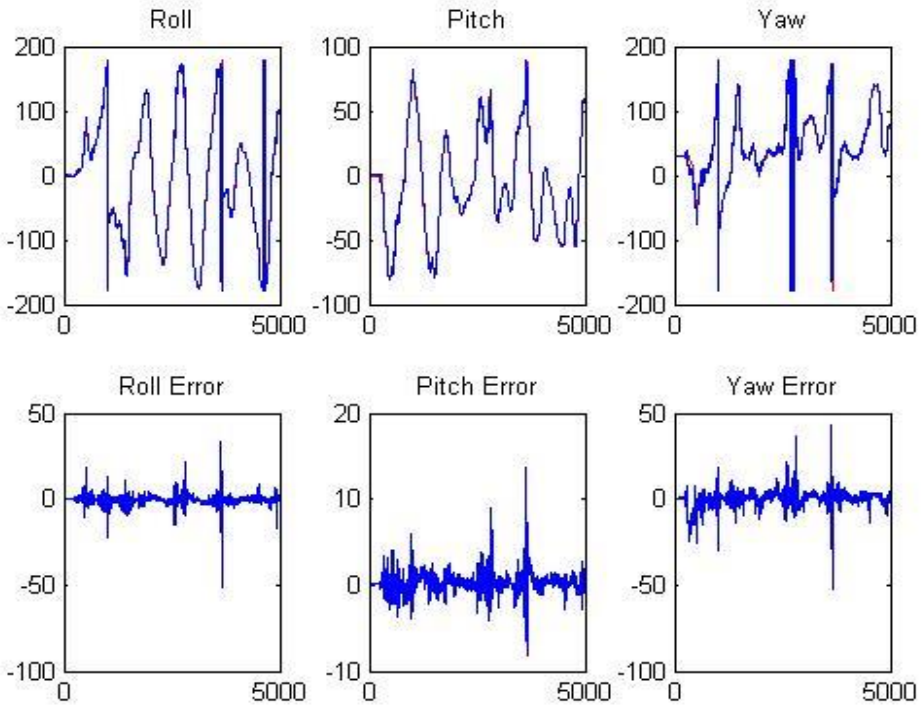


Figure 16: Simulation result for Gauss Newton method on condition 2.

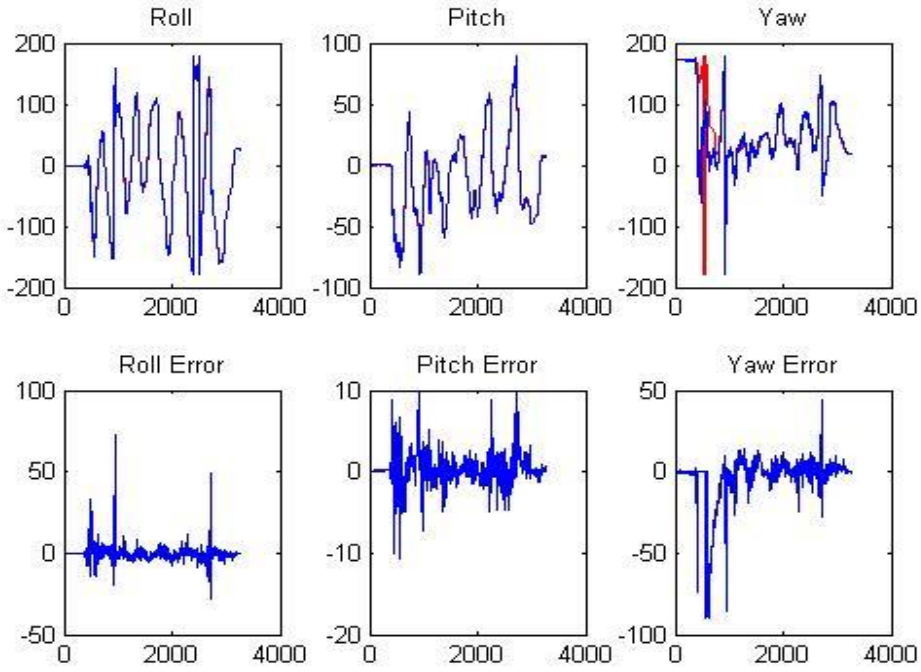


Figure 17: Simulation result for Gauss Newton method on condition 3.

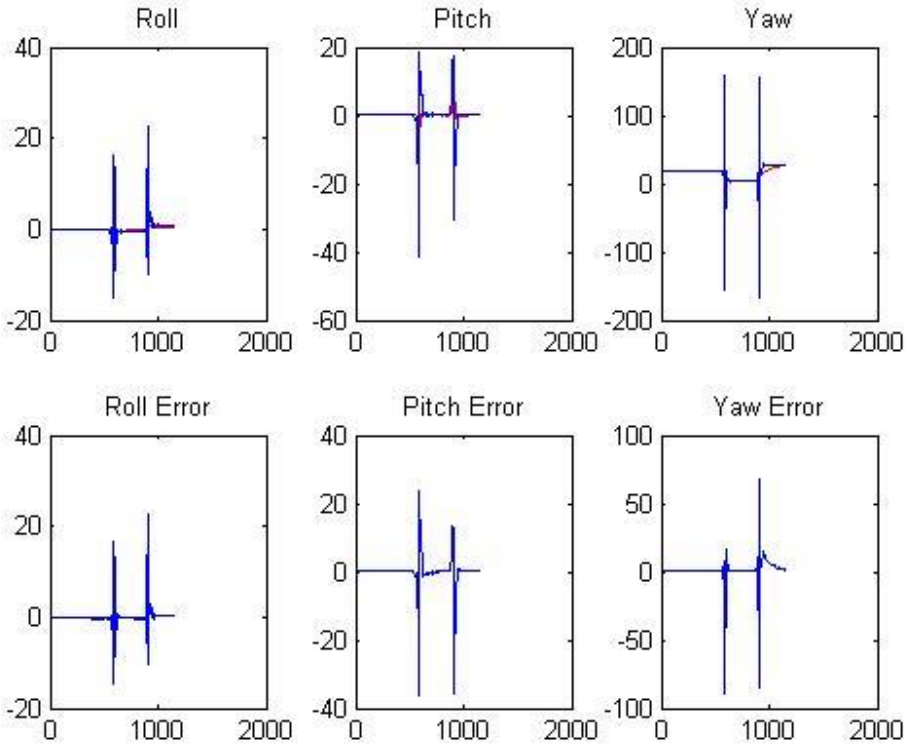


Figure 18: Simulation result for Gauss Newton method on condition 4.

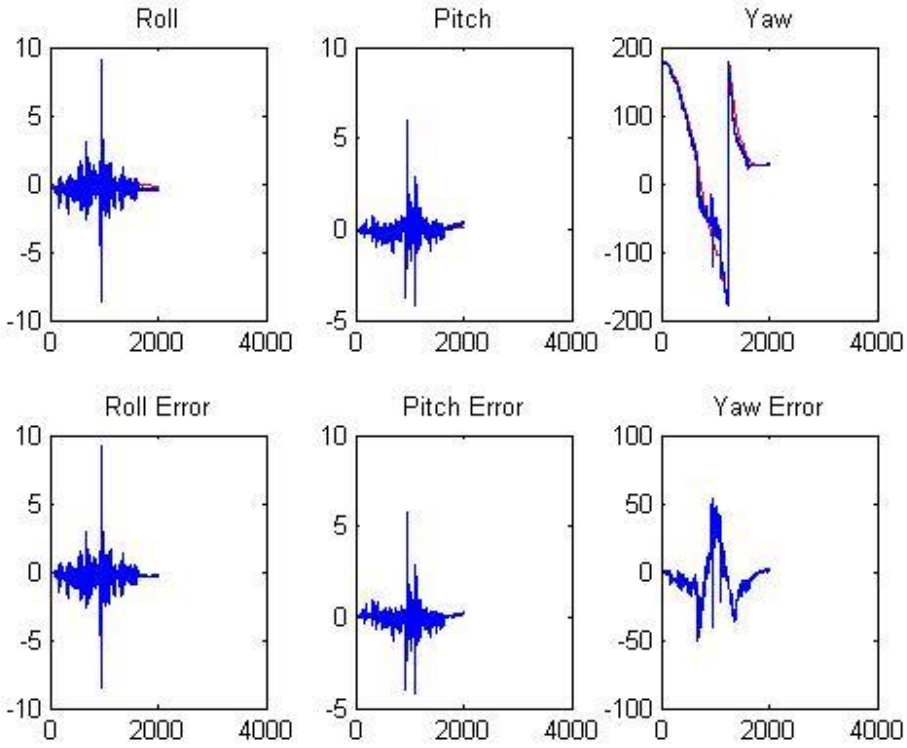


Figure 19: Simulation result for Gauss Newton method on condition 5.

3.1.3 AHRS

AHRS which is a famous way to estimate object orientation from 6-DOF sensor, can be also used as a pre-filter to feed into a quaternion based EKF. It is more accurate and faster computed than Gauss Newton method, with 188.02 micro-second per cycle. The simulation results are showed as below:

AHRS RMS					
Conditions	**1**	**2**	**3**	**4**	**5**
roll	0.085	2.134	8.840	1.069	1.721
pitch	0.048	0.623	8.501	3.522	2.642
yaw	0.488	2.616	21.405	2.192	36.336

Table 3: AHRS RMS.

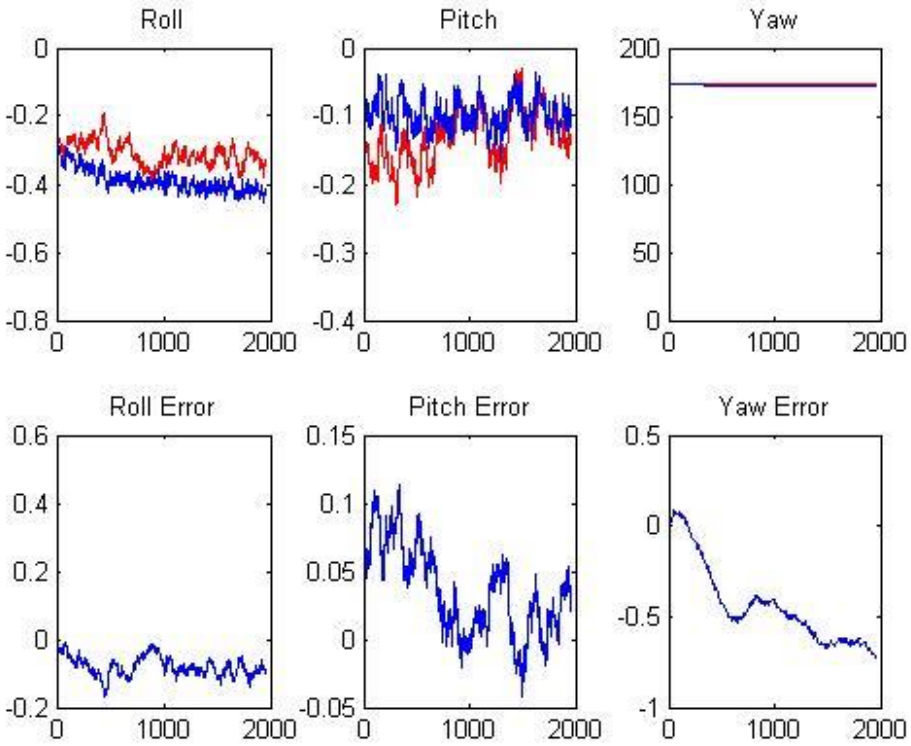


Figure 20: Simulation result for AHRS on condition 1.

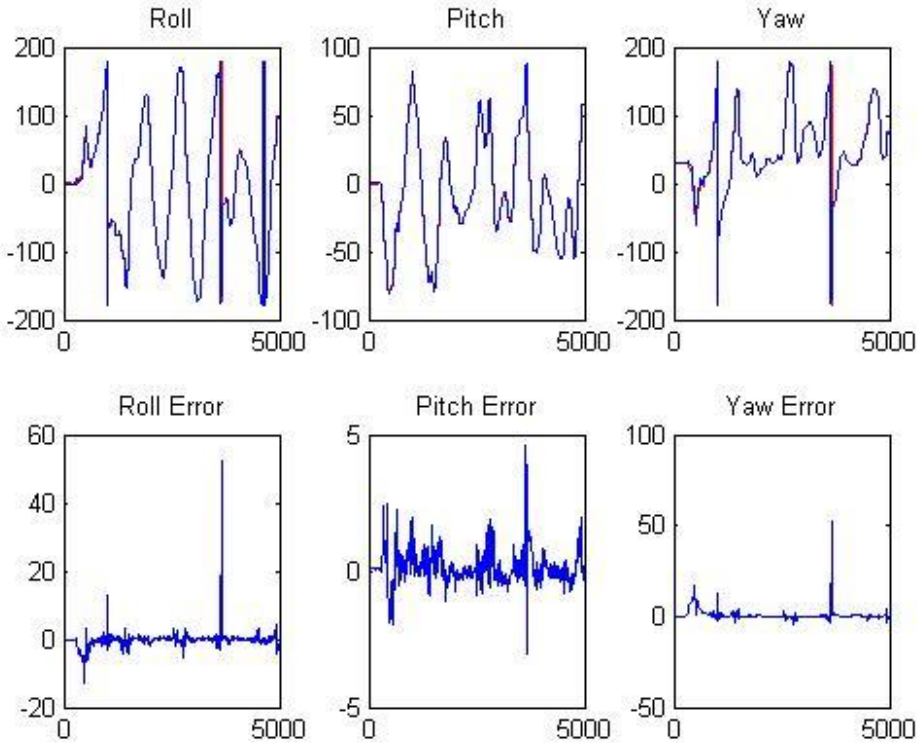


Figure 21: Simulation result for AHRS on condition 2.

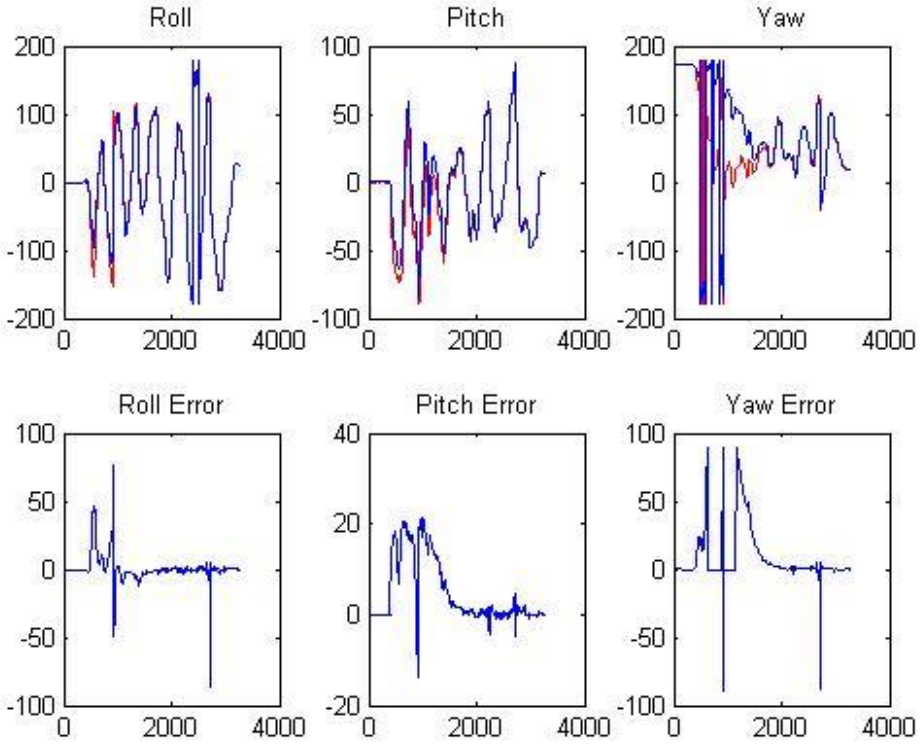


Figure 22: Simulation result for AHRS on condition 3.

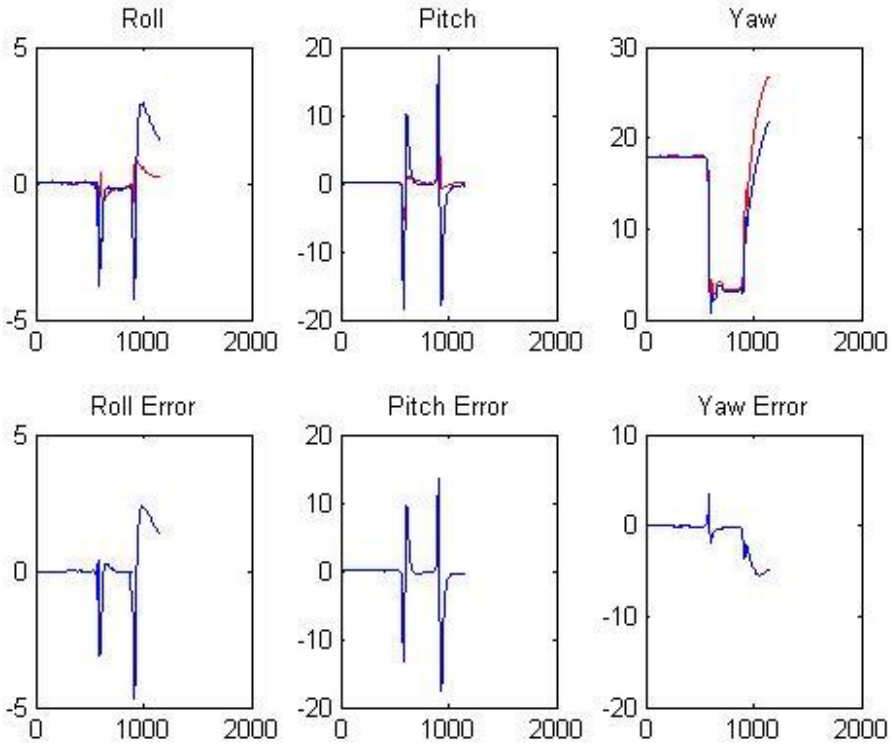


Figure 23: Simulation result for AHRS on condition 4.

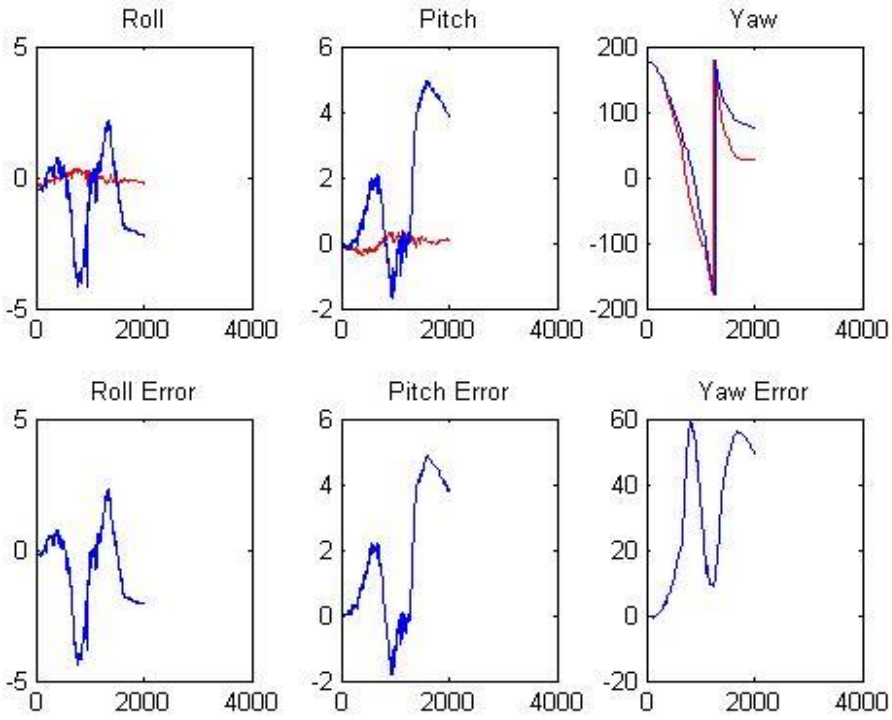


Figure 24: Simulation result for AHRS on condition 5.

3.1.4 Quaternion based – gyro bias – EKF

In this model, the state vector is: $x = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 & b_x & b_y & b_z \end{pmatrix}$

Where $q = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 \end{pmatrix}$ is the quaternion represents the sensor's orientation and $b = \begin{pmatrix} b_x & b_y & b_z \end{pmatrix}$ is the bias of gyrometer data.

The advantage of this model is that it could monitor the bias rate of the gyrometer data, compared to the estimation, so the system can avoid drift effect of most low priced gyrometers. While the lower part of Q matrix is for bias rate, process noise covariance matrix is chose to be:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-9} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-9} \end{pmatrix}$$

The Measurement noise covariance matrix is showed below with the upper four lines and the rest are referred to quaternion and bias rate respectively:

$$R = \begin{pmatrix} 10^{-6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-4} \end{pmatrix}$$

It takes 103.41 micro-second for each cycle. The simulation results are showed as below:

Quaternion based – gyro bias – EKF					
Conditions	**1**	**2**	**3**	**4**	**5**
roll	0.046	1.879	9.058	0.178	0.256
pitch	0.046	0.678	1.772	0.612	0.426
yaw	0.928	2.892	13.862	2.594	17.417

Table 4: Quaternion based – gyro bias – EKF.

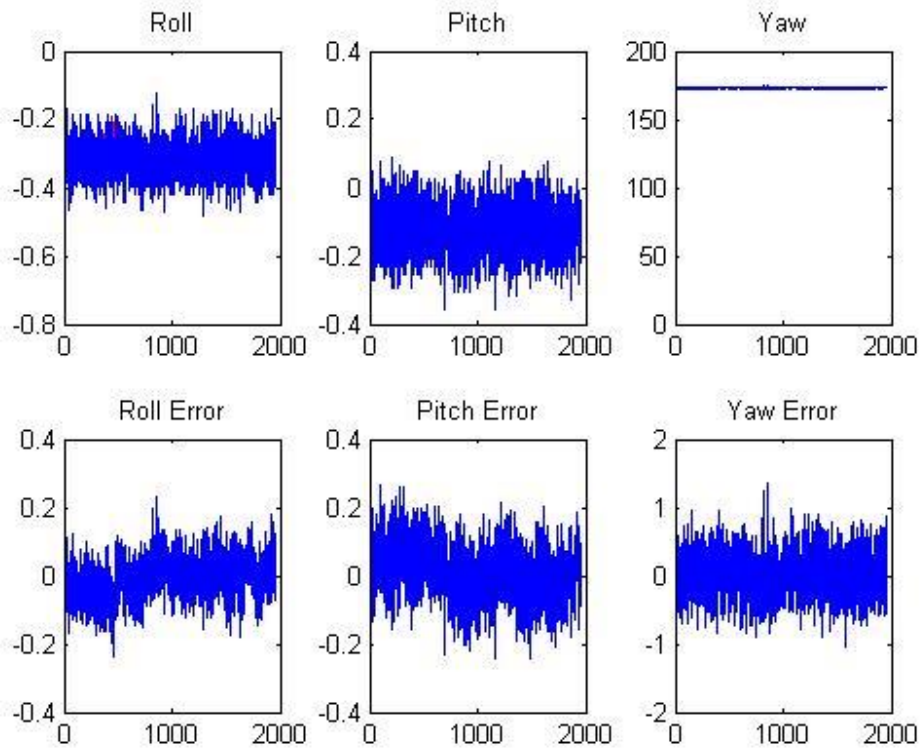


Figure 25: Simulation result for Qua based – gyro bias – EKF on condition 1.

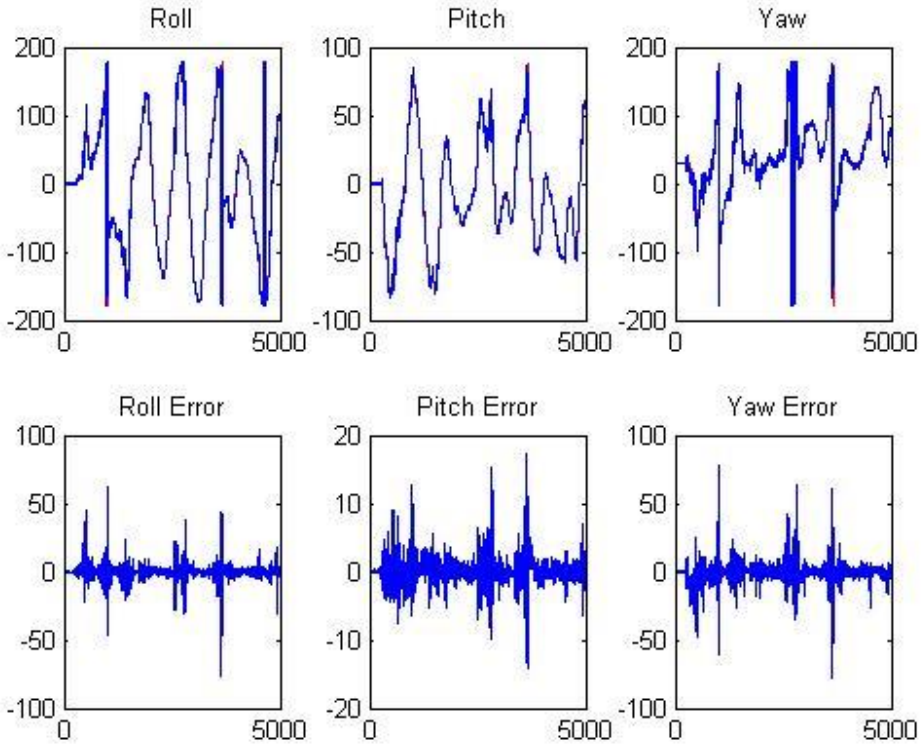


Figure 26: Simulation result for Qua based – gyro bias – EKF on condition 2.

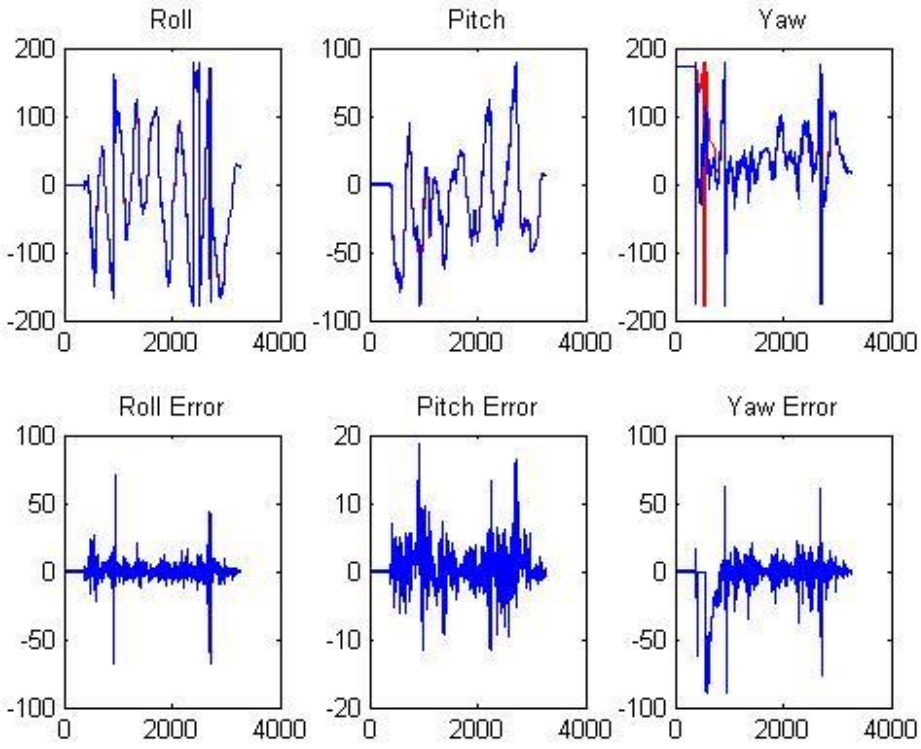


Figure 27: Simulation result for Qua based – gyro bias – EKF on condition 3.

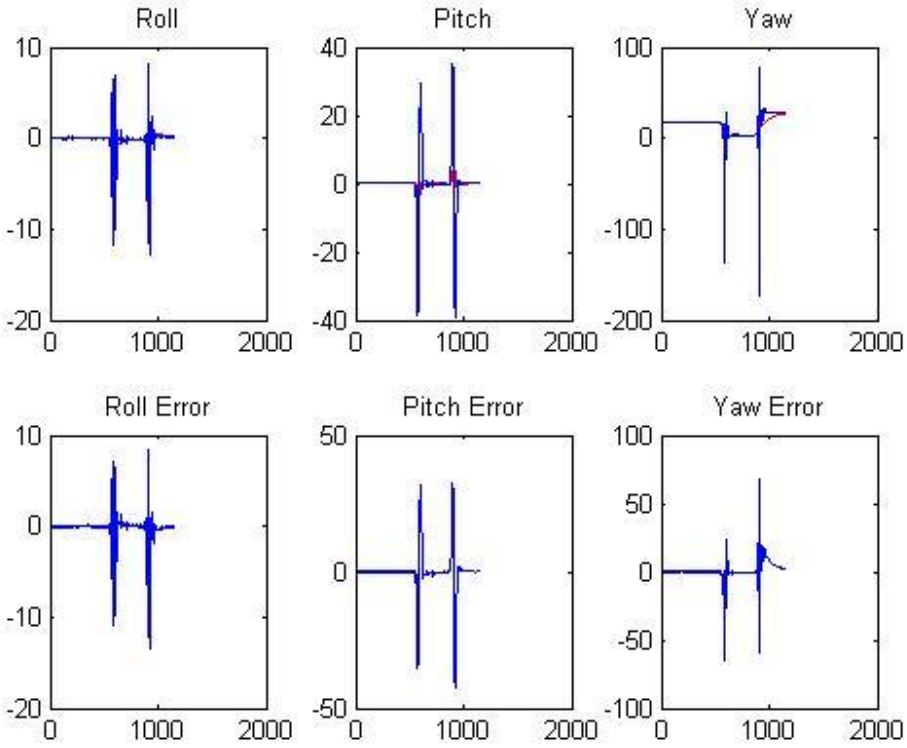


Figure 28: Simulation result for Qua based – gyro bias – EKF on condition 4.

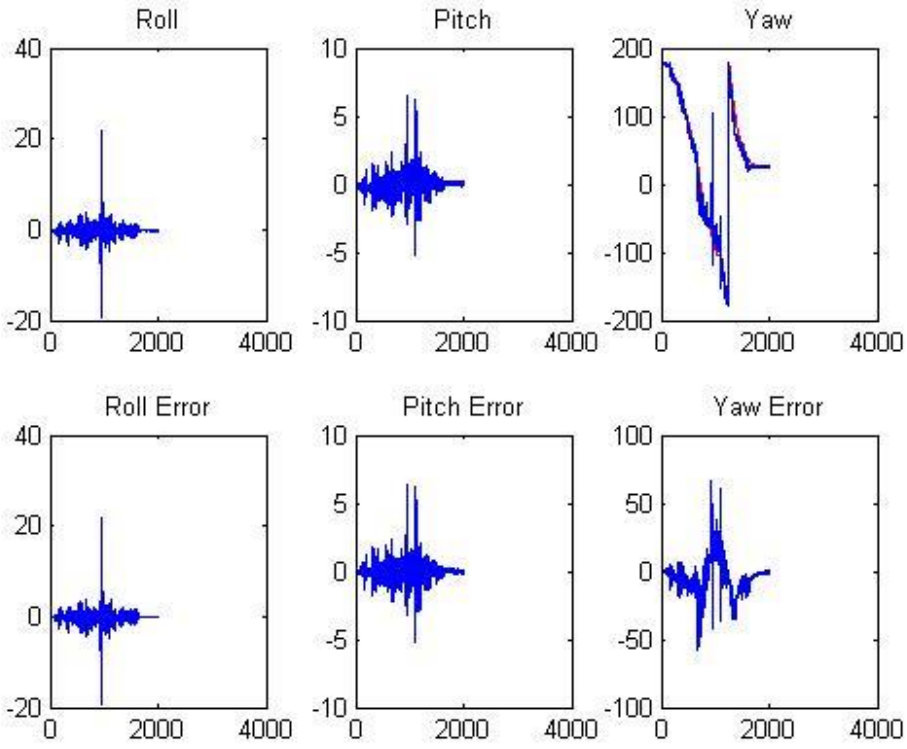


Figure 29: Simulation result for Qua based – gyro bias – EKF on condition 5.

3.1.5 Quaternion based – gyro rate – EKF

In this model, the state vector is: $x = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 & g_x & g_y & g_z \end{pmatrix}$

Where $q = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 \end{pmatrix}$ is the quaternion represents the sensor's orientation and $g = \begin{pmatrix} g_x & g_y & g_z \end{pmatrix}$ is the angular rate data from gyrometer.

Instead of monitoring bias rate of the gyrometer data, this system looks at the angular rate data and compare to the estimation. While the lower part of Q matrix is for bias rate, process noise covariance matrix is chose to be:

$$Q = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-9} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-9} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-9} \end{pmatrix}$$

The Measurement noise covariance matrix is showed below with the upper four lines and the rest are referred to quaternion and bias rate respectively:

$$R = \begin{pmatrix} 10^{-6} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10^{-4} \end{pmatrix}$$

It takes 109.68 micro-second for each cycle. The simulation results are showed as below:

Quaternion based – gyro rate – EKF RMS					
Conditions	**1**	**2**	**3**	**4**	**5**
roll	0.2391	1.49326	3.204693	0.29247	0.694984
pitch	0.20965	0.53013	0.859146	0.4439	0.899367
yaw	0.11438	1.56441	6.8912	1.32024	24.11162

Table 5: Quaternion based – gyro rate – EKF.

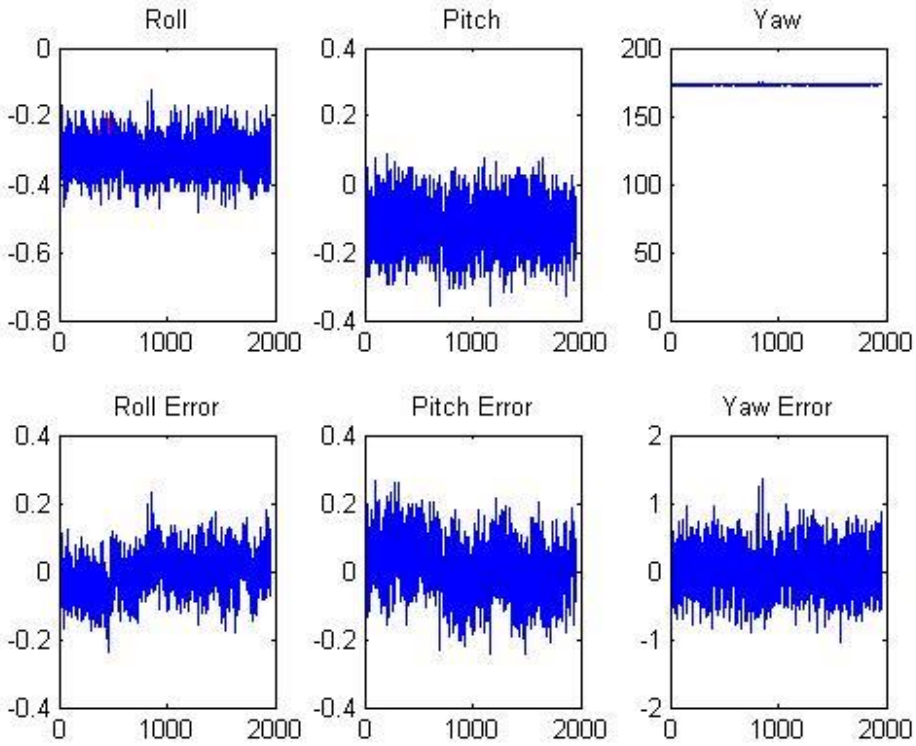


Figure 30: Simulation result for Qua based – gyro rate – EKF on condition 1.

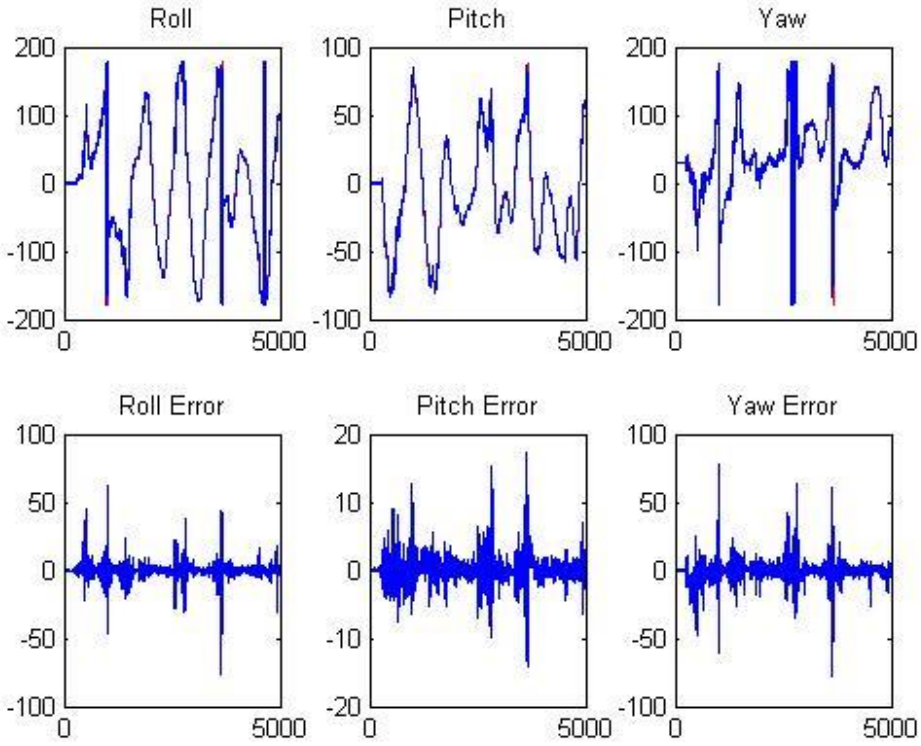


Figure 31: Simulation result for Qua based – gyro rate – EKF on condition 2.

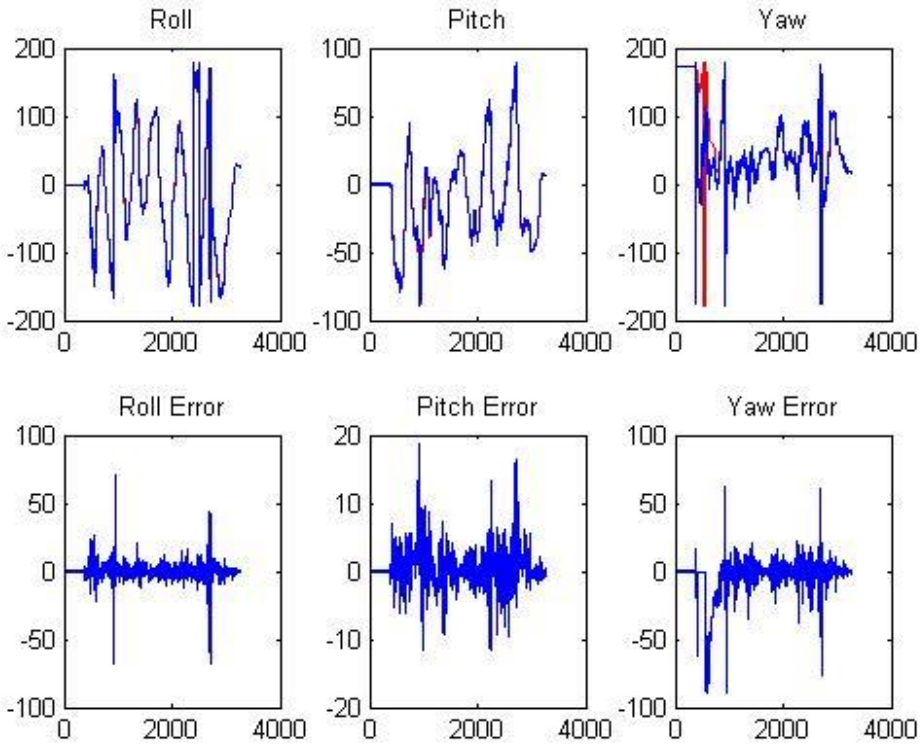


Figure 32: Simulation result for Qua based – gyro rate – EKF on condition 3.

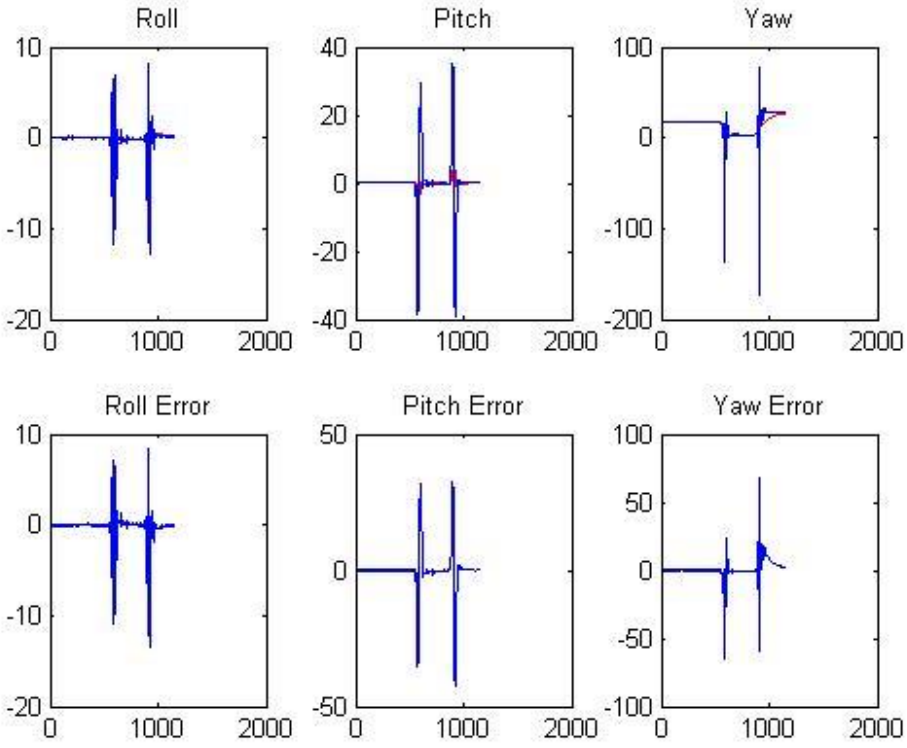


Figure 33: Simulation result for Qua based – gyro rate – EKF on condition 4.

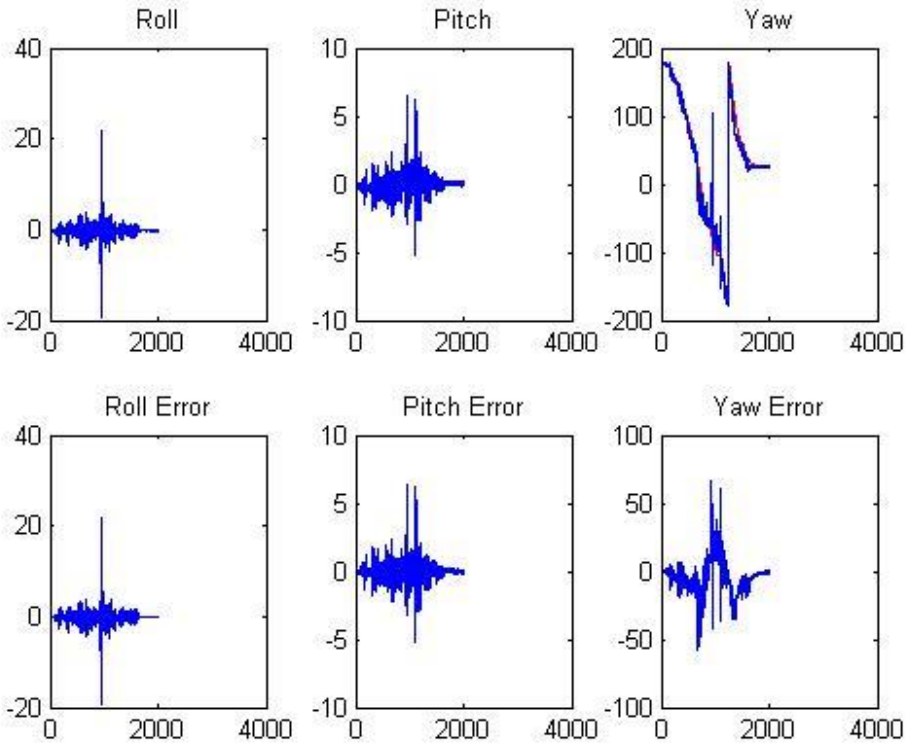


Figure 34: Simulation result for Qua based – gyro rate – EKF on condition 5.

3.1.6 DCM based KF

Choosing process noise and measurement noise covariance matrix to be:

$$Q_1 = R_1 = Q_2 = R_2 = \begin{pmatrix} 10^{-5} & 0 & 0 \\ 0 & 10^{-5} & 0 \\ 0 & 0 & 10^{-5} \end{pmatrix}$$

Each cycle takes 90.87 micro-second. The simulation results are below:

DCM based KF RMS					
Conditions	**1**	**2**	**3**	**4**	**5**
roll	0.17207	1.27154	1.742496	0.27642	0.234564
pitch	0.15025	0.41687	0.64271	0.20385	0.120836
yaw	0.65095	1.63914	2.049651	1.47216	0.80049

Table 6: DCM based KF RMS.

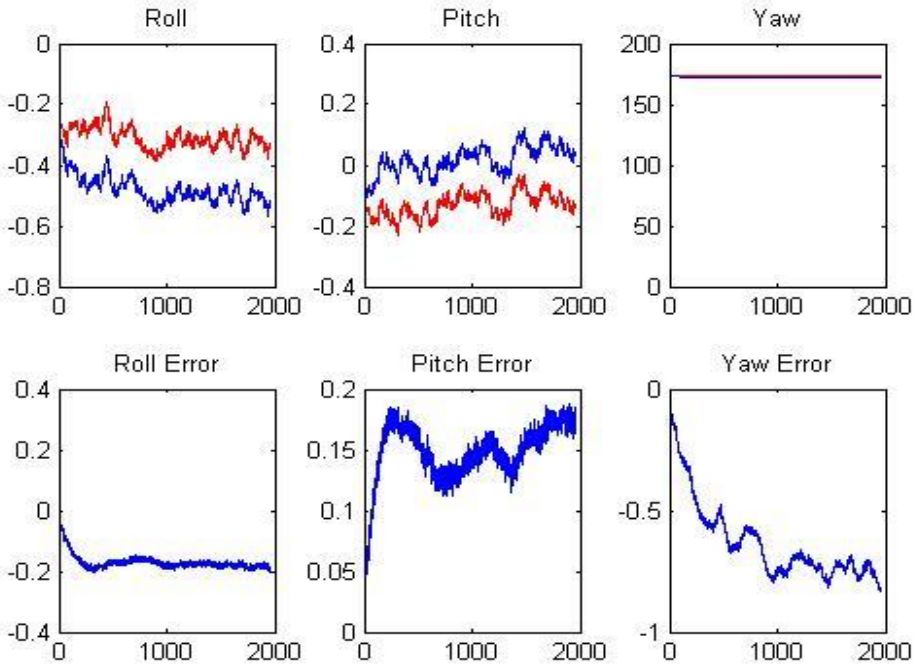


Figure 35: Simulation result for DCM based – KF on condition 1.

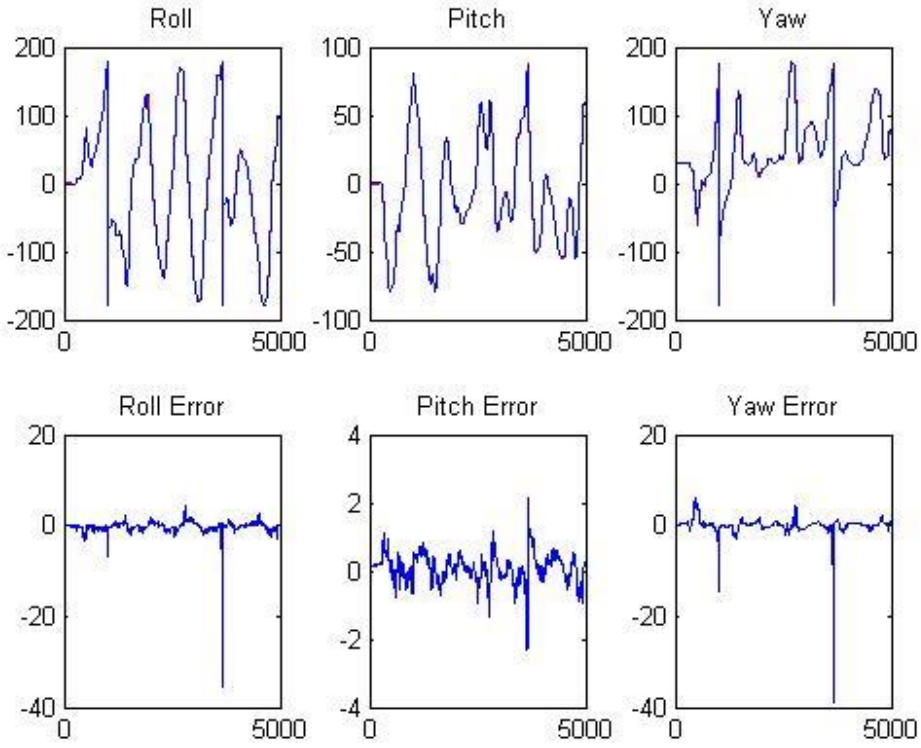


Figure 36: Simulation result for DCM based – KF on condition 2.

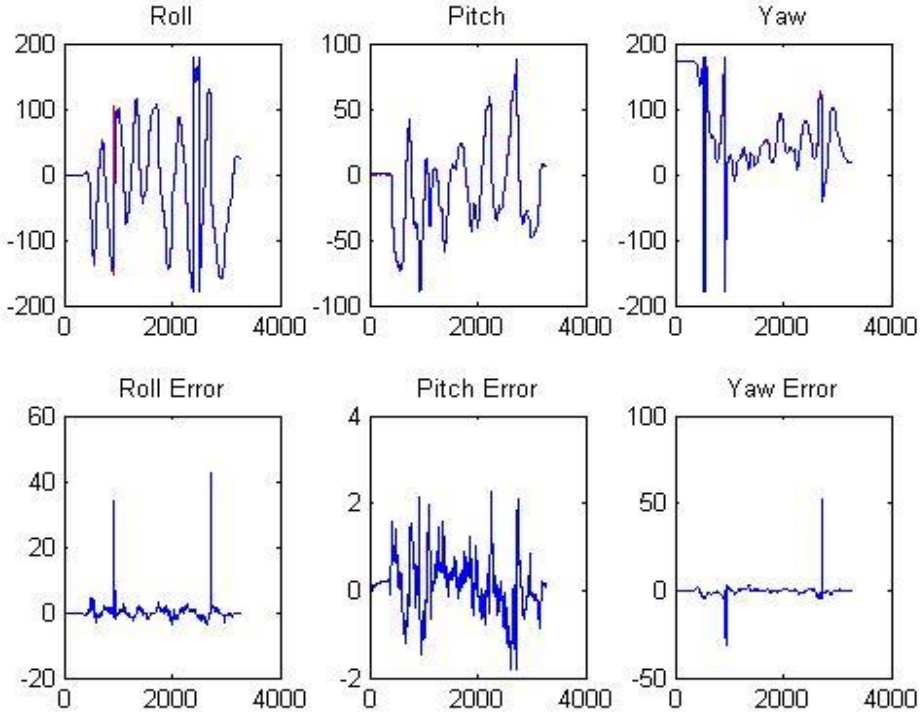


Figure 37: Simulation result for DCM based – KF on condition 3.

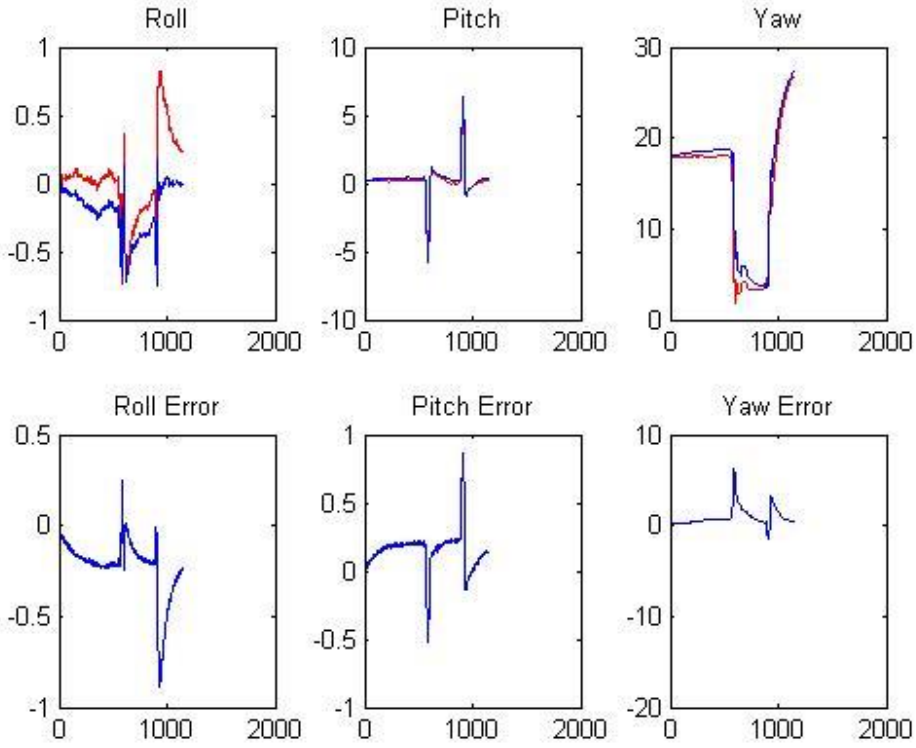


Figure 38: Simulation result for DCM based – KF on condition 4.

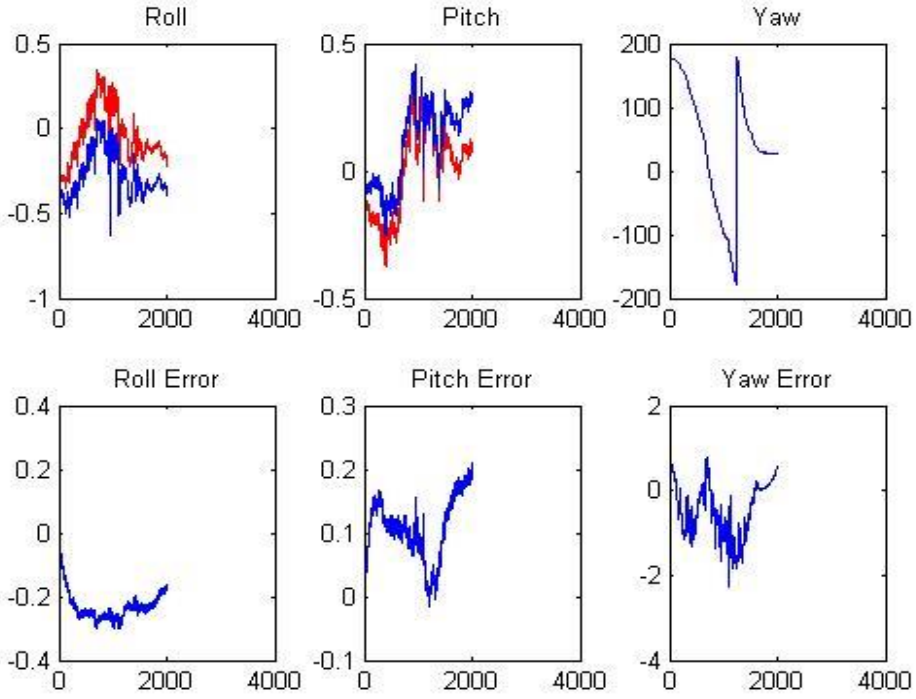


Figure 39: Simulation result for DCM based – KF on condition 5.

3.1.7 Result

The simulations proved Kalman filter gives much improvement than other calculation methods and a popular 6-axis filter AHRS:

RMS table							
Conditions		TILT-COMP	GAU NEW	AHRS	Tilt EKF bias	Tilt EKF gyro	DCM KF
1	roll	0.065	0.046	0.085	0.046	0.239	0.172
	pitch	0.084	0.046	0.048	0.046	0.210	0.150
	yaw	0.335	0.928	0.488	0.928	0.114	0.651
2	roll	5.882	2.082	2.134	1.879	1.493	1.272
	pitch	2.084	1.048	0.623	0.678	0.530	0.417
	yaw	7.953	2.747	2.616	2.892	1.564	1.639
3	roll	6.580	6.433	8.840	9.058	3.205	1.742
	pitch	3.010	3.022	8.501	1.772	0.859	0.643
	yaw	15.514	15.486	21.405	13.862	6.891	2.050
4	roll	1.575	1.598	1.069	0.178	0.292	0.276
	pitch	7.413	7.273	3.522	0.612	0.444	0.204
	yaw	6.911	7.874	2.192	2.594	1.320	1.472
5	roll	1.380	0.967	1.721	0.256	0.695	0.235
	pitch	0.716	0.554	2.642	0.426	0.899	0.121
	yaw	14.497	14.152	36.336	17.417	24.112	0.800
Cycle (us)		43.870	319.640	188.020	103.410	109.680	90.870

Table 7: RMS table

3.2 Bluetooth communication

Communication link is for data transmit between the filter system and the PC base station. Therefore, it is necessary to design compact transmit method to save CPU resources. The serial communication is designed as below:

- PC: to receive data stream and process to visualize on user interface. It can also send command to the board for updating new parameter or doing tasks.
- Board: It executes PC's commands and processes sensor then and put push the result onto data stream to PC.

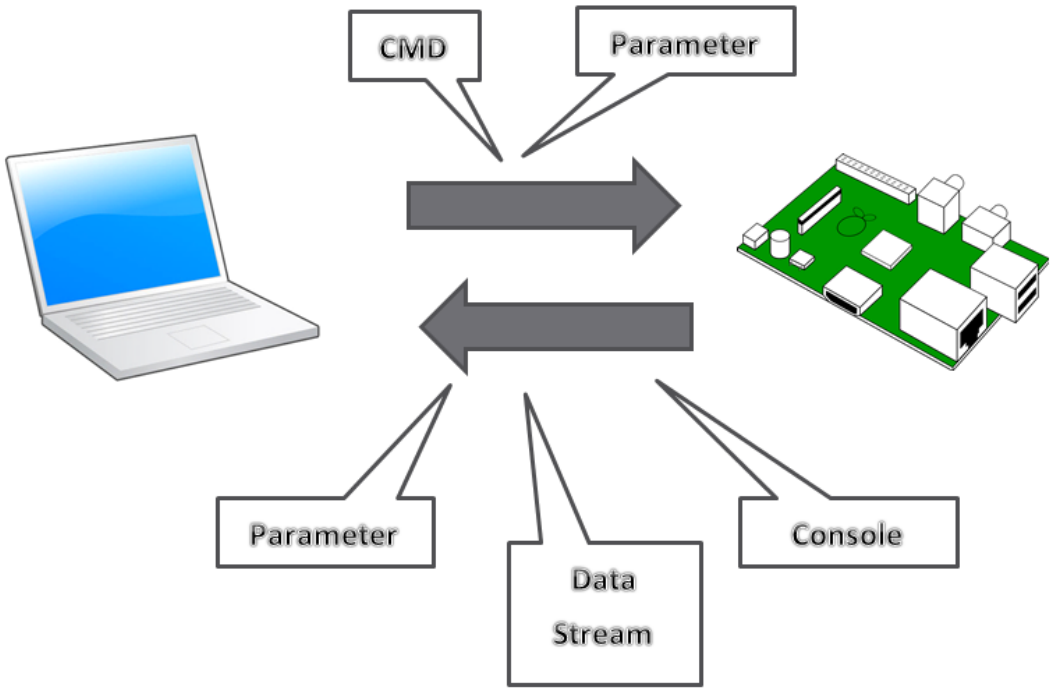


Figure 40: Bluetooth communication

Console mode		CMD mode	
bit length	data type	bit length	data type
1 byte	header - 'V'	1 byte	header - 'V'
1 byte	header - 'N'	1 byte	header - 'N'
1 byte	header - 'D'	1 byte	header - 'D'
1 byte	indicate - '*'	1 byte	indicate - '*'
64 byte	char	1 byte	CDM

Table 8: Serial Protocol in console mode and CMD mode.

Streaming mode			Parameter mode	
bit length	data type	multiply by	bit length	data type
1 byte	header - 'V'	x	1 byte	header - 'V'
1 byte	header - 'N'	x	1 byte	header - 'N'
1 byte	header - 'D'	x	1 byte	header - 'D'
1 byte	indicate - '_'	x	1 byte	indicate - '$'
2 bytes	mag x	10^2	2 bytes	mag offset x
2 bytes	mag y	10^2	2 bytes	mag offset y
2 bytes	mag z	10^2	2 bytes	mag offset z
2 bytes	Euler roll	10^2	2 bytes	mag gain x
2 bytes	Euler pitch	10^2	2 bytes	mag gain y
2 bytes	Euler yaw	10^2	2 bytes	mag gain z
3 bytes	quaternion q1	10^6	1 byte	Compl Roll
3 bytes	quaternion q2	10^6	1 byte	Compl Pitch
3 bytes	quaternion q3	10^6	1 byte	Compl yaw
3 bytes	quaternion q4	10^6	2 bytes	AHRS Kp
3 bytes	dcm R0	10^6	2 bytes	AHRS Ki
3 bytes	dcm R1	10^6	3 bytes	EKFQUA r
3 bytes	dcm R2	10^6	3 bytes	EKFQUA q
3 bytes	dcm R3	10^6	3 bytes	EKFDCM q1
3 bytes	dcm R4	10^6	3 bytes	EKFDCM r1
3 bytes	dcm R5	10^6	3 bytes	EKFDCM q2
3 bytes	dcm R6	10^6	3 bytes	EKFDCM r2
3 bytes	dcm R7	10^6	2 bytes	dt
3 bytes	dcm R8	10^6		

Table 9: Serial Protocol in streaming mode and parameter mode.

3.3 Hardware

3.3.1 Components

a. Microprocessor - STM32F407xx

- Core: ARM 32-bit Cortex™-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/ 1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions.
- Clock, reset and supply management – 1.8 V to 3.6 V application supply and I/Os – POR, PDR, PVD and BOR – 4-to-26 MHz crystal oscillator – Internal 16 MHz factory-trimmed RC (1% accuracy) – 32 kHz oscillator for RTC with calibration – Internal 32 kHz RC with calibration.
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support.
- Up to 17 timers: up to twelve 16-bit and two 32bit timers up to 168 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input.
- Up to 3 × I2C interfaces (SMBus/PMBus).
- Debug mode – Serial wire debug (SWD) & JTAG interfaces – Cortex-M4 Embedded Trace Macrocell™.

b. EPPROM - 24LC256

The Microchip Technology Inc. 24LC256 is a 256Kb (32K x 8) Serial Electrically Erasable PROM (EEPROM), capable of operation across a broad voltage range (1.7V to 5.5V). It has been developed for advanced, low-power applications such as personal communications or data acquisition. This device also has a page write capability of up to 64 bytes of data. This device is capable of both random and sequential reads up to the 256K boundary. Functional address lines allow up to eight devices on the same bus, for up to 2 Mbit address space. This device is available in the standard 8-pin plastic DIP, SOIC, TSSOP, MSOP and DFN packages.

c. Gyroscope and Accelerometer - MPU6050

Gyroscope Features the triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec.
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization.
- Integrated 16-bit ADCs enable simultaneous sampling of gyros.
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration.
- Improved low-frequency noise performance.
- Digitally-programmable low-pass filter.
- Gyroscope operating current: 3.6mA.
- Standby current: 5µA.
- Factory calibrated sensitivity scale factor User self-test

Accelerometer Features The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g and ±16g.
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer.
- Accelerometer normal operating current: 500µA.
- Low power accelerometer mode current: 10µA at 1.25Hz, 20µA at 5Hz, 60µA at 20Hz, 110µA at 40Hz.
- Orientation detection and signaling.
- Tap detection.
- User-programmable interrupts.
- High-G interrupt.
- User self-test.

d. 3-Axis Digital Compass - HMC5883L

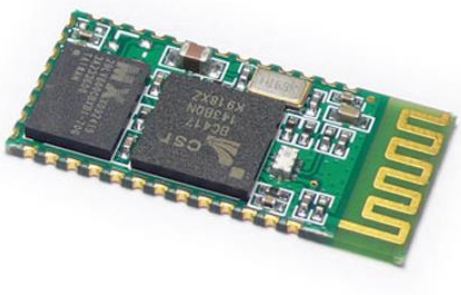
The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometer. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I2C serial bus allows for easy interface.

The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.

The HMC5883L utilizes Honeywell's Anisotropic Magnetoresistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors' solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth's magnetic fields, from milli-gauss to 8 gauss. Honeywell's Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

e. Bluetooth module - HC05

HC05 is a serial port Bluetooth module, drop-in replacement for wired serial connections, transparent usage. It can be used simply for a serial port replacement to establish connection between MCU and other devices. HC-05 could be setting to Master or Slave by user but this project is focus on Slave mode only



.

Figure 41: HC05 Bluetooth module

In order to use this module, configuring step need to pass through. There are two AT modes:

- AT Mode 1: After power on, it can enter the AT mode by triggering PIN34 with high level. Then the baud rate for setting AT command is equal to the baud rate in communication, for example: 9600.
- AT mode 2: First set the PIN34 as high level, or while on powering the module set the PIN34 to be high level, the Baud rate used here is 38400 bps.

All AT commands can be operated only when the PIN34 is at high level. Only part of the AT commands can be used if PIN34 doesn’t keep the high level after entering to the AT mode. Through this kind of designing, set permissions for the module is left to the user’s external control circuit, which makes the application of HC-05 is very flexible.

3.3.2 Schematic

The board's processor is STM32F4, which is later discussed in following section.

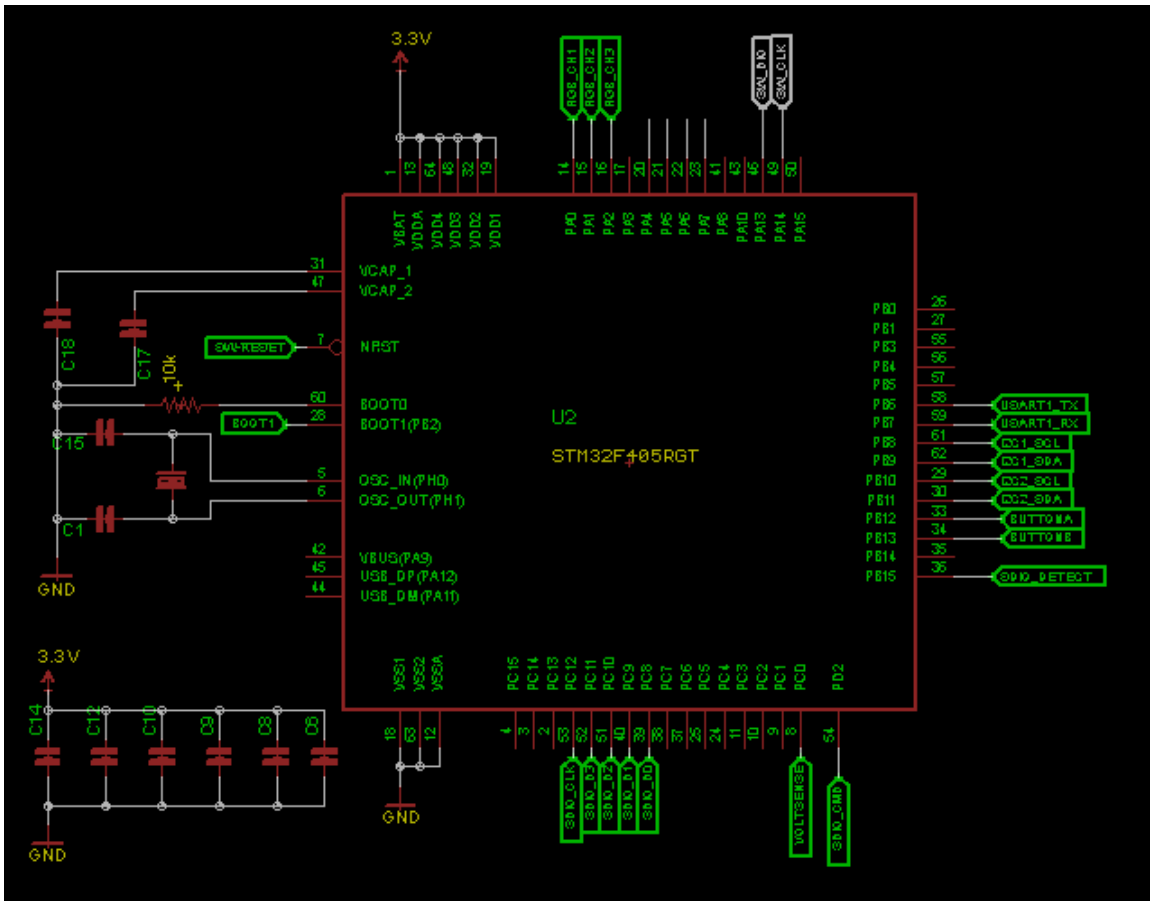


Figure 42: Schematic – STM32F40X processor

With SWD connector for programming and debugging:

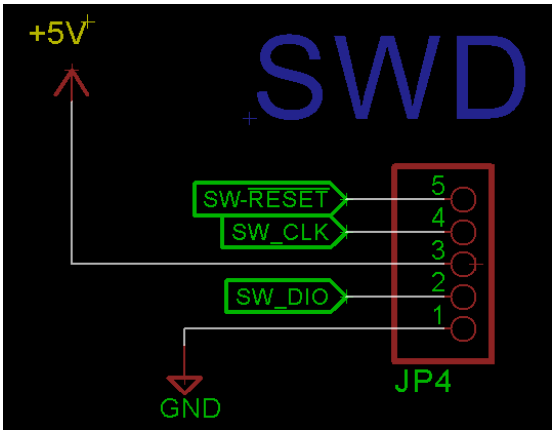


Figure 43: Schematic – SWD port

HC05 Bluetooth module is connected to STM32F40X USART1 port. Led are mounted to indicate different modes of the module and PIO11 is grounded to make sure the module is always in connection mode.

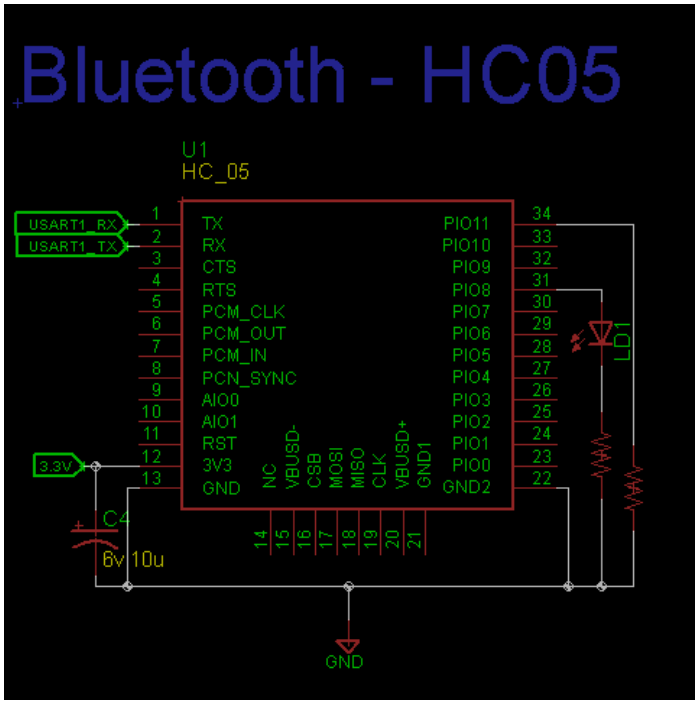


Figure 44: Schematic – HC05 module

SD card socket and EPPROM is also mounted:

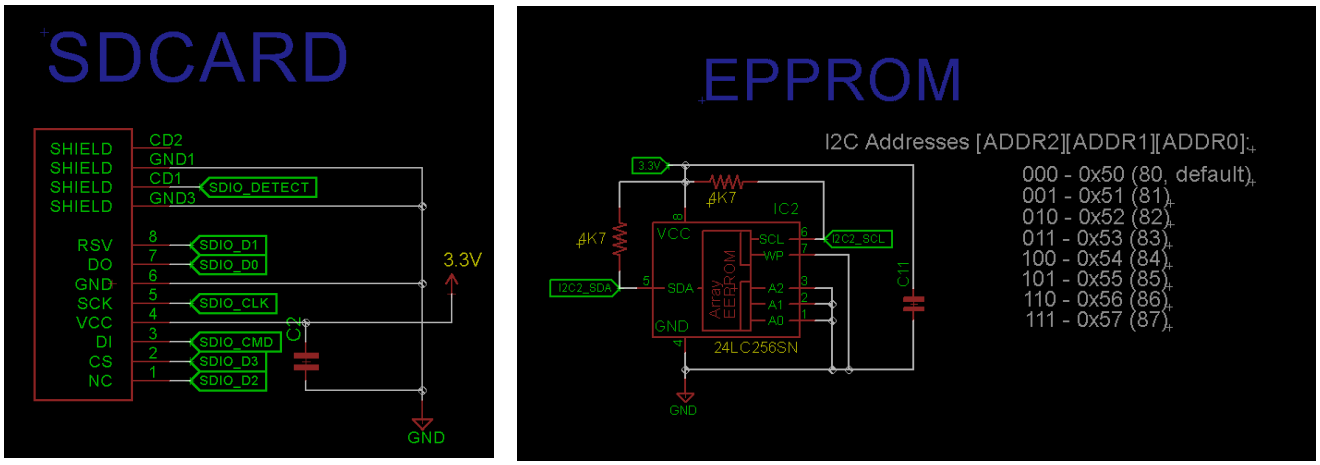


Figure 45: Schematic – SD card socket and EPPROM

HMC5883 and MPU6050 are connected to I2C1 port. Couple resister R10 and R11 are used to pull up I2C line to high level.

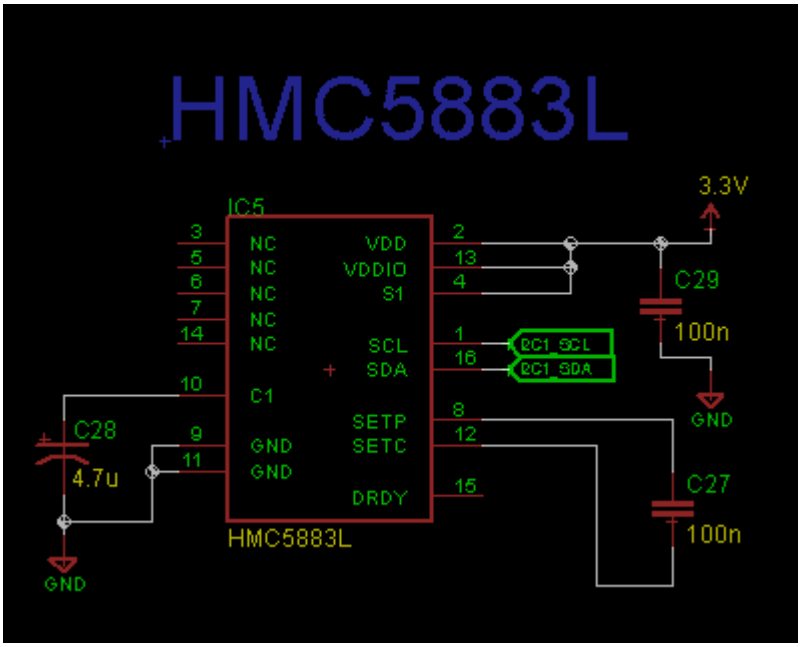


Figure 46: Schematic – HMC5883L

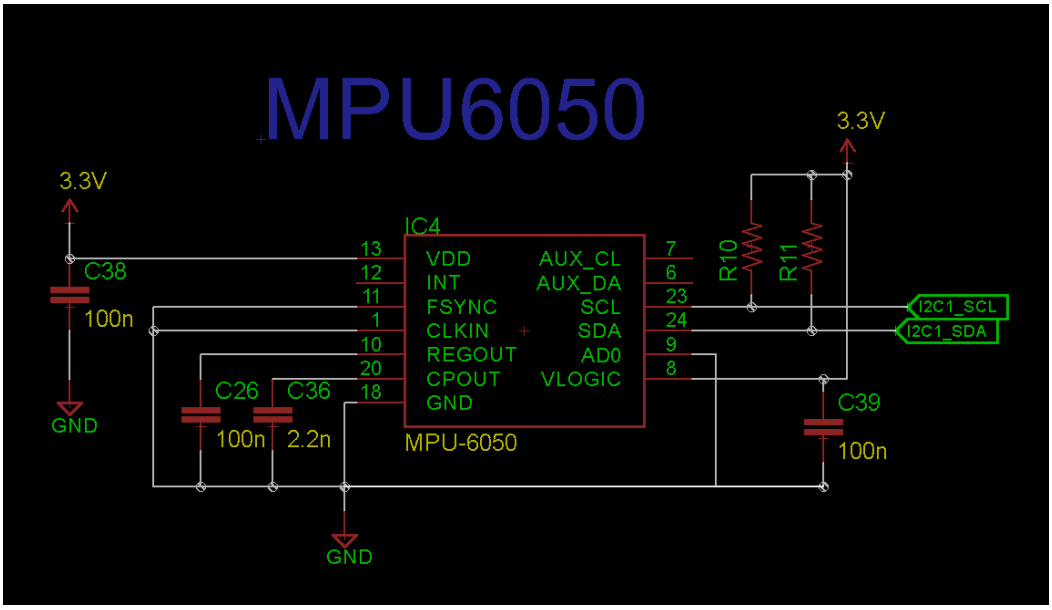


Figure 47: Schematic – MPU6050

3.3.3 Layout

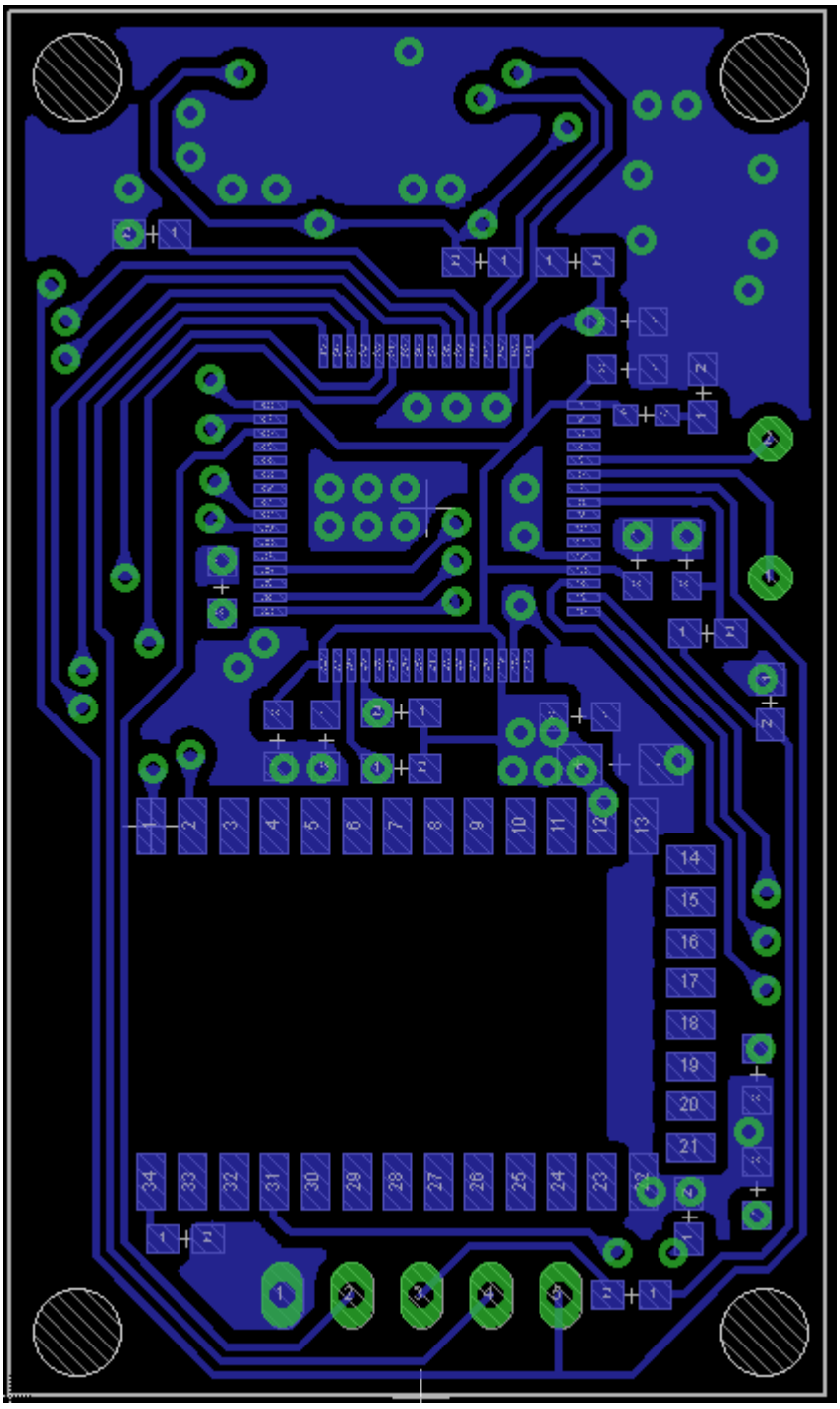


Figure 48: Layout – bottom.

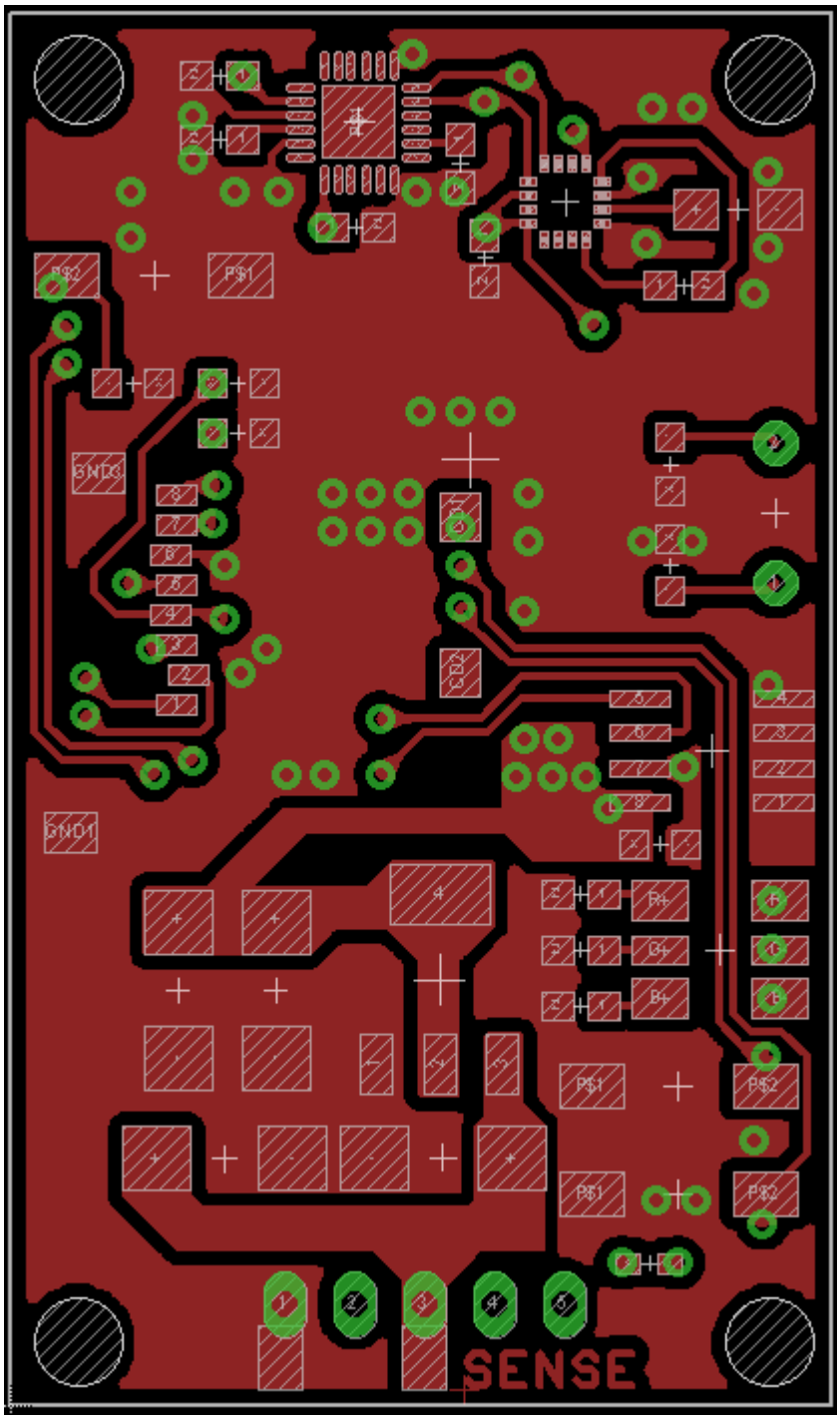


Figure 49: Layout – top.

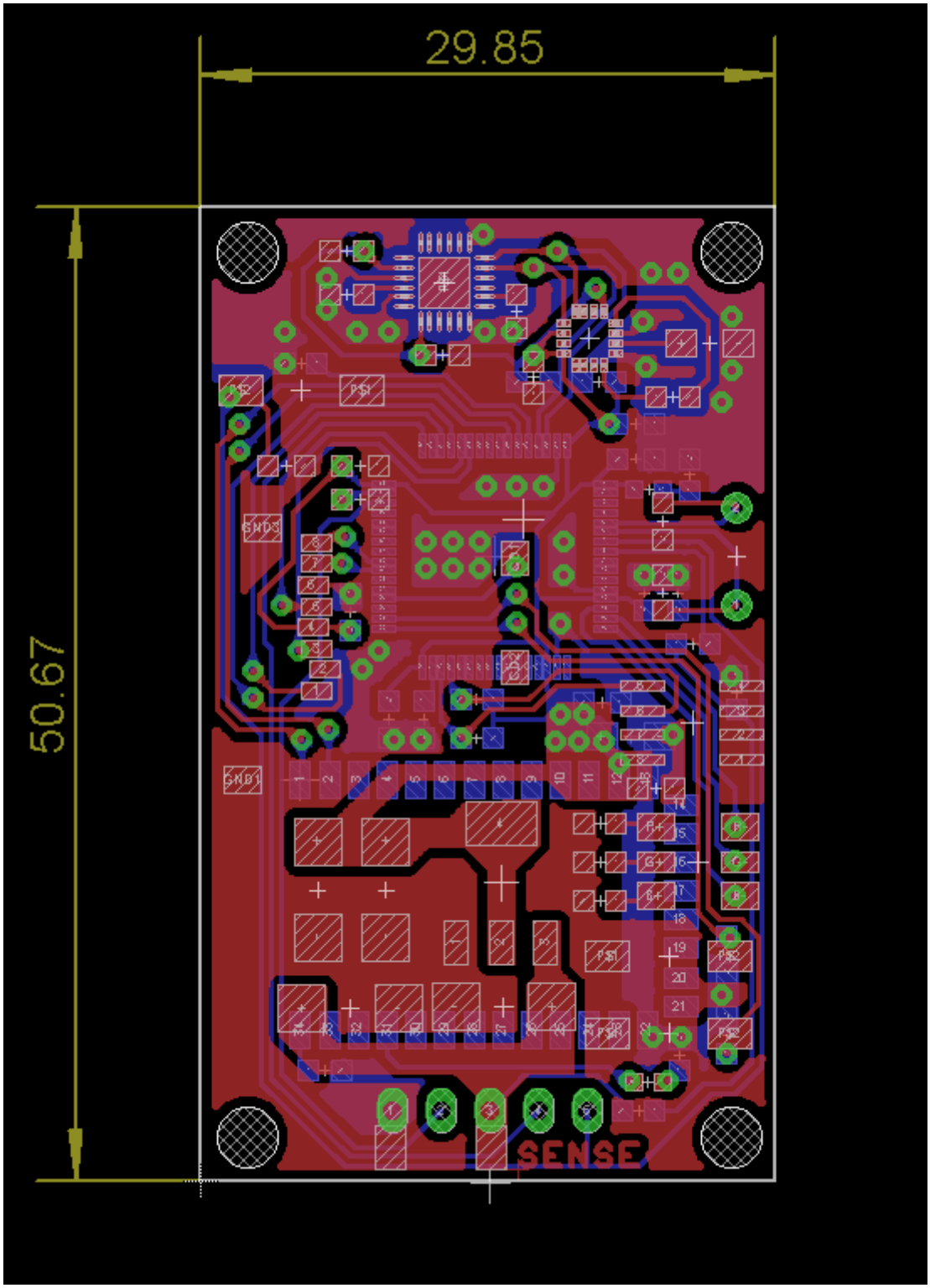


Figure 50: Layout – top and bottom.

3.3.4 The Sense

The first prototype is built on the BalaPi, which supports I2C ports and uSD card:

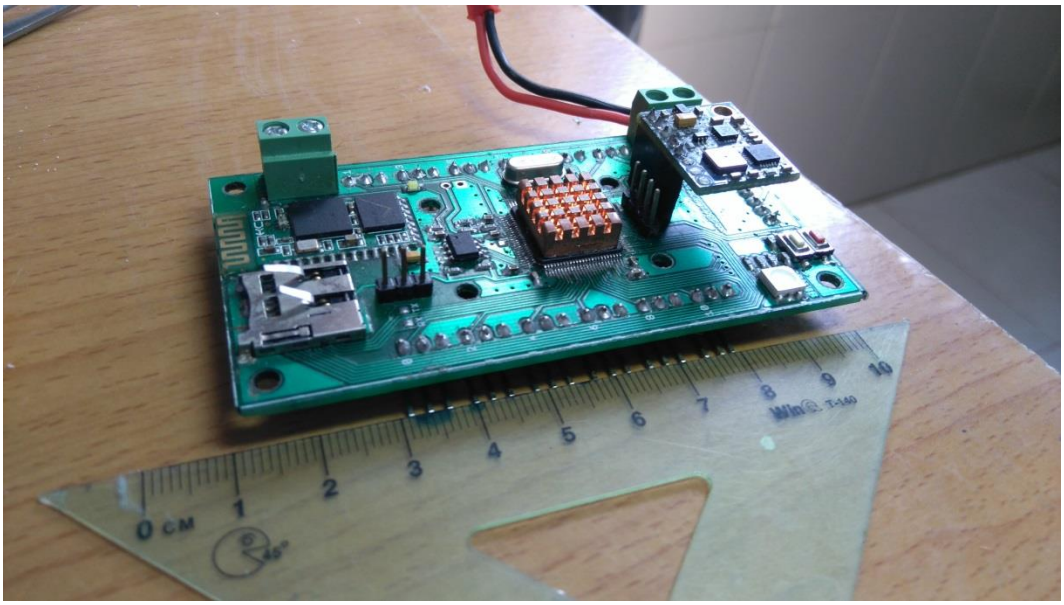


Figure 51: The BalaPi

After that, the system is re-designed to be more compact, which is the SENSE:

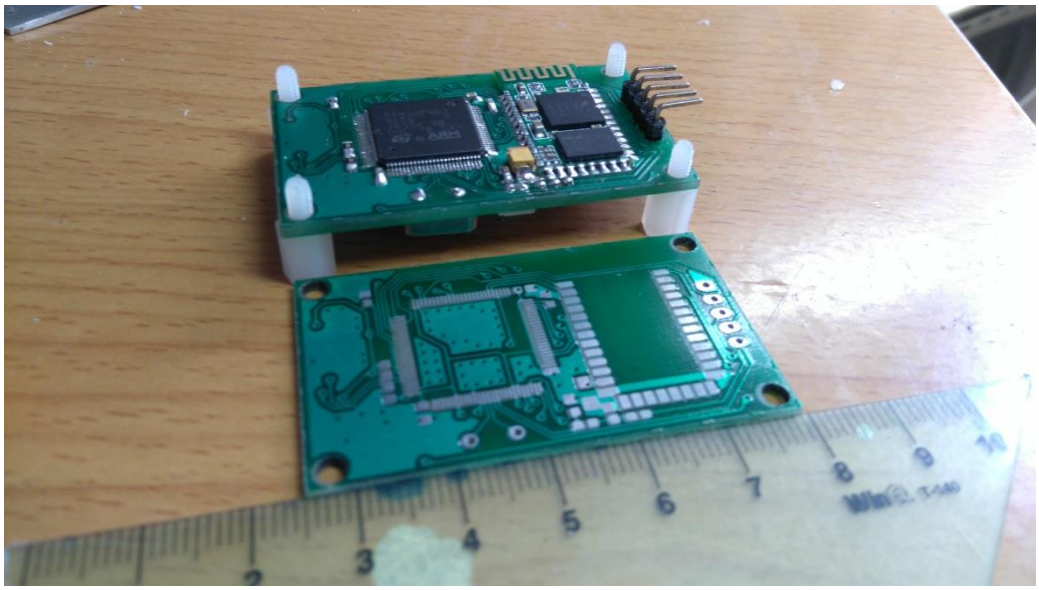


Figure 52: The SENSE (back side)

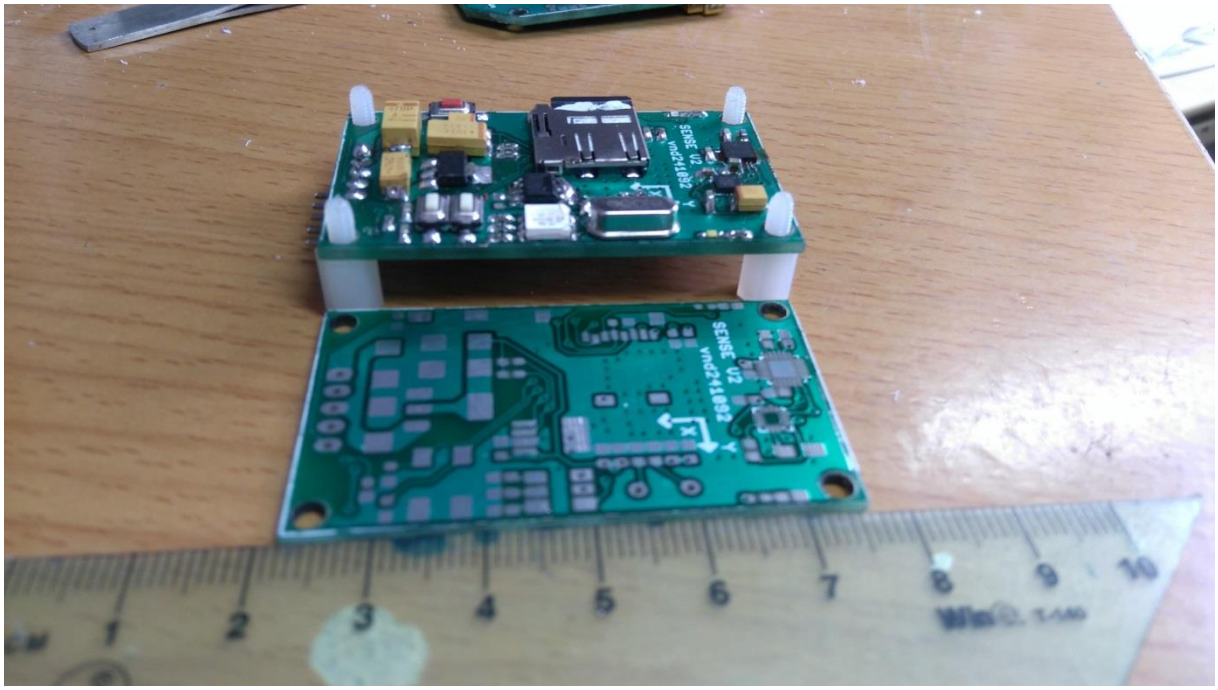


Figure 53: The SENSE (front side)

And then we have it in mica shield:

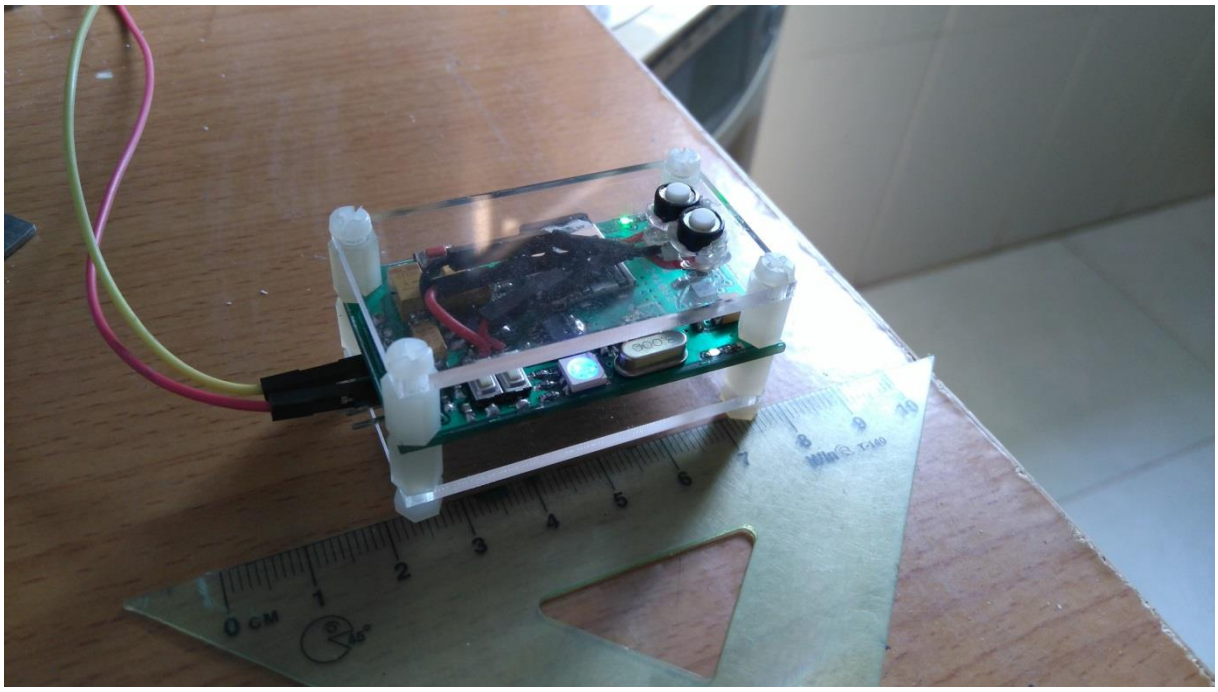


Figure 54: The SENSE (in mica shield 1)

Or a shield with more spaces for battery:

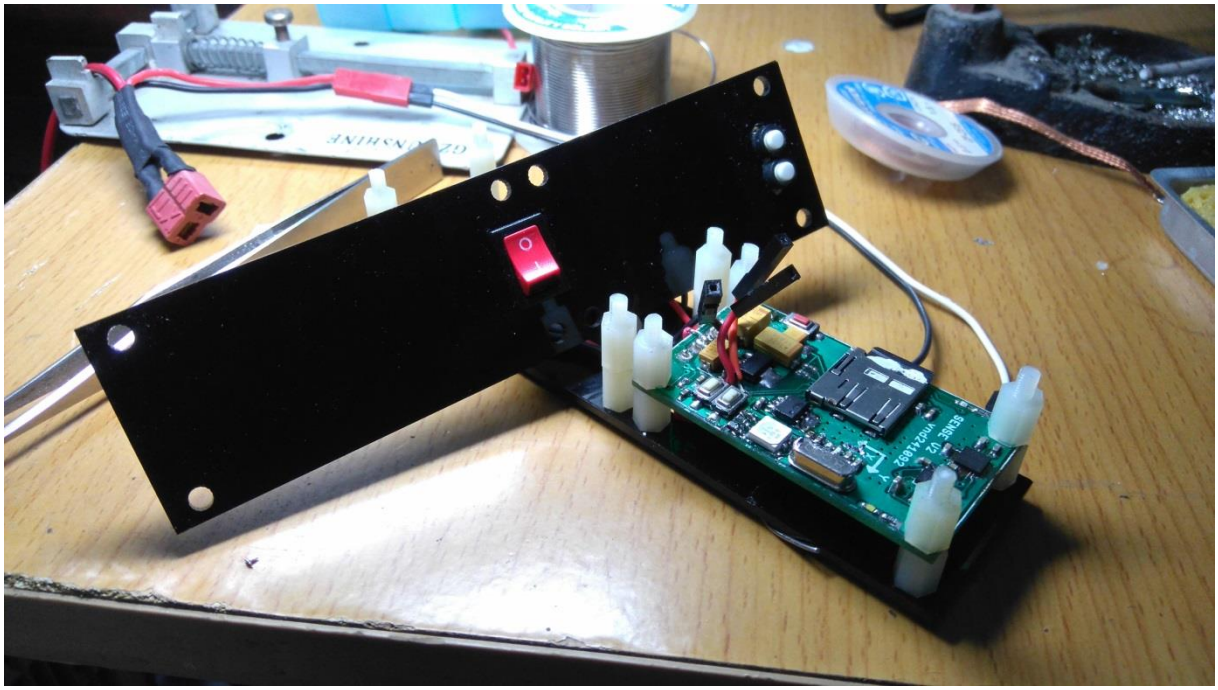


Figure 55: The SENSE (in mica shield 2)

The Sense with 9V battery:

Figure 56: The SENSE with 9V battery

3.4 Calibration

Some cheap magnetometer modules like HMC5883L cannot be used without calibration. Measurement of magnetic field will be subjected to distortion. There are two categories of these distortions: the hard iron distortions and the soft iron distortions. The hard iron errors refer to the presence of magnetic fields around the sensor (magnets, power supply wires) and are related to measurement offset errors, while the soft iron errors refer to the presence of ferromagnetic materials around the sensor, which skew the density of the Earth's magnetic field locally and are related to scaling offset errors.

The uncalibrated data could be the shifting ellipse:

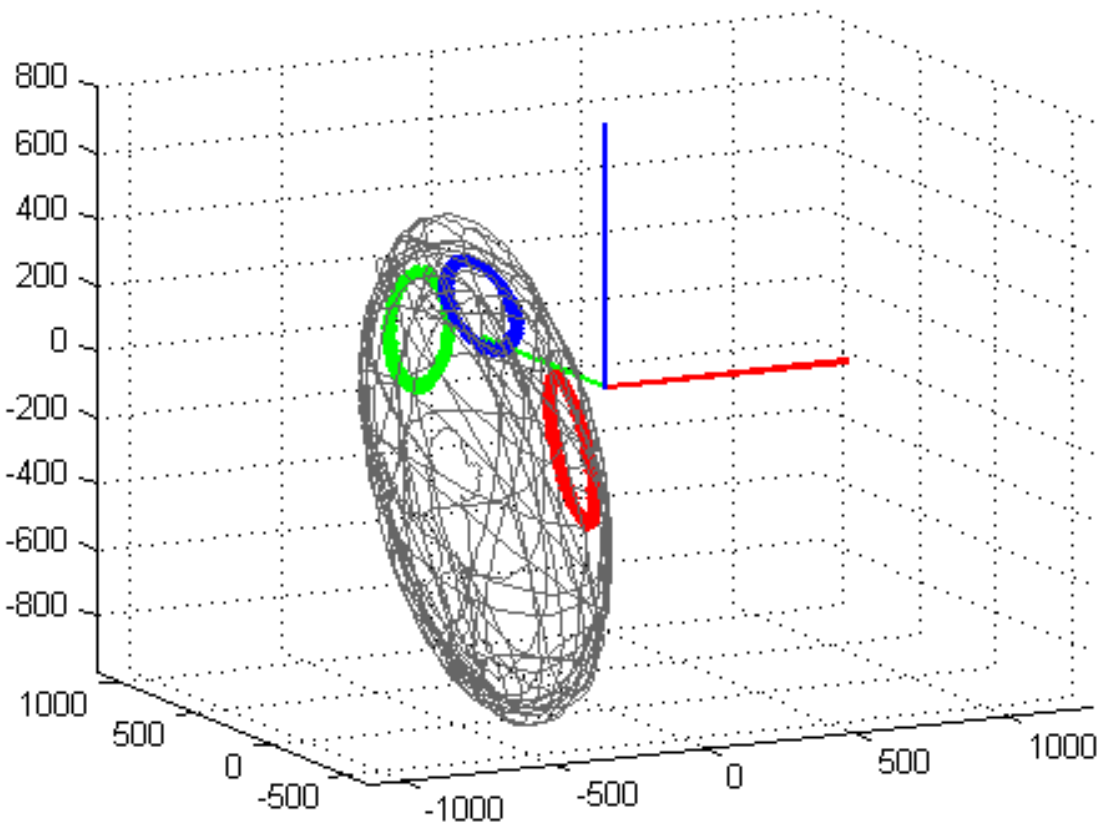


Figure 57: Uncalibrated magnetometer data

In other words, to get the correct magnetometer data it should be calibrated to have the spherical form similar to Figure 57:

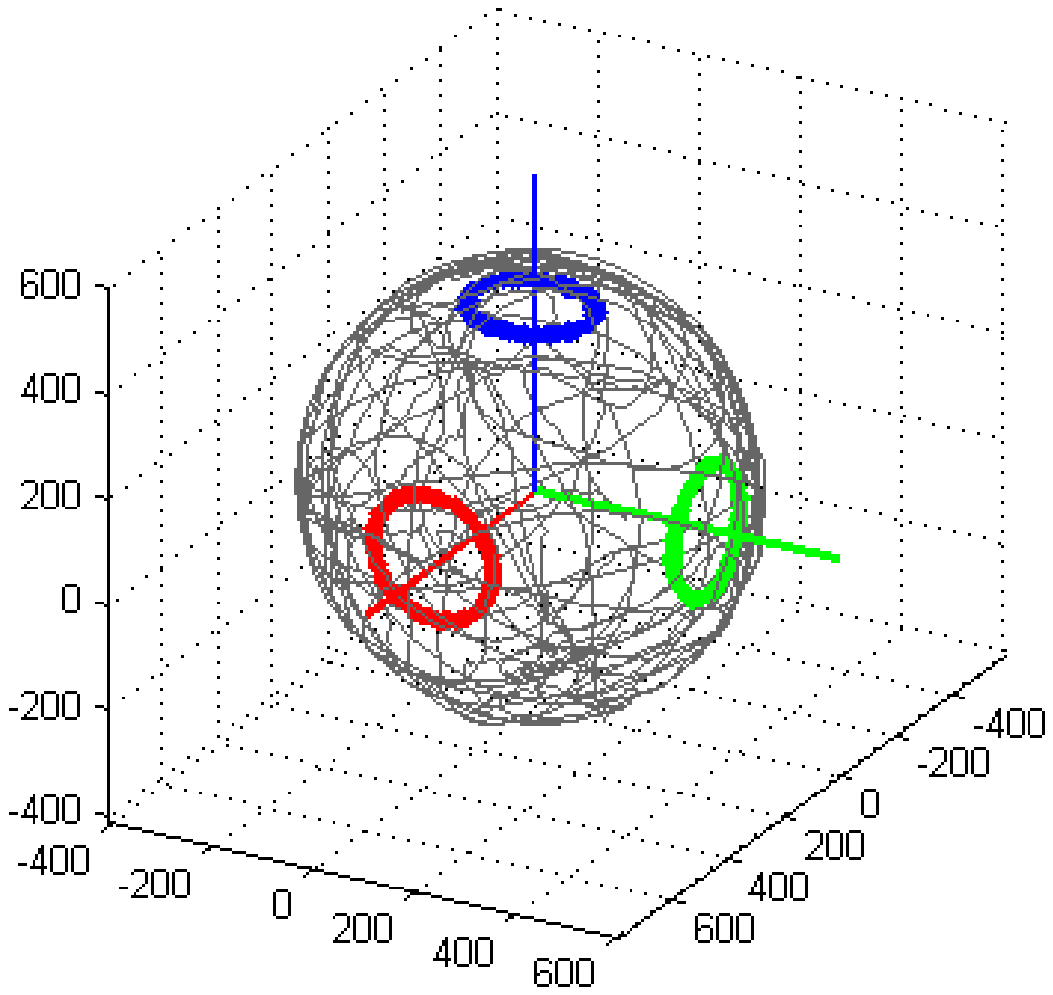


Figure 58: Calibrated magnetometer data

The magnetometer can be calibrated by adding offset and multiplying with specific gain.

```
//////////////////////////////////////////////
//  Magn
//add offset and scale to circle
Magn->x =   ((MagOffset.x + (float)mx) *   MagScalerGain.x);
Magn->y =   ((MagOffset.y + (float)my) *   MagScalerGain.y);
Magn->z =   ((MagOffset.z + (float)mz) *   MagScalerGain.z);
//////////////////////////////////////////////
```

Figure 59: Magnetic calibration

The magnetic gain, which reforms the magnetometer data on spherical surface:

```
SENSOR_Status SENSOR_Estimate_MagnGain(void)
{
  int16_3D Tmp_Mag;
  uint64_t zero_time;
  float Highlimit, Lowlimt;

  if(HMC5883_Configuration_A(HMC5883_SAMPLEAVERAGE_8, HMC5883_RATE_15, HMC5883_BIAS_POSITIVE)!= I2C_STATUS_OK)
    return SENSOR_MagCalibGain_PosBias_Fail;

  // Disregarding the first data
  if(HMC5883_Get_RawMagn(&Tmp_Mag.x, &Tmp_Mag.y, &Tmp_Mag.z)!= I2C_STATUS_OK)
    return  SENSOR_MagCalibGain_PosBias_ReadFail;

  // Read positive bias values
  zero_time = Get_msTick();
  Highlimit = 200/HMC5883_Get_Gain();
  while (Magn_PosBias.x < Highlimit | Magn_PosBias.y < Highlimit | Magn_PosBias.z < Highlimit)
  {
    if(Check_msTick(zero_time,10000))         return SENSOR_MagCalibGain_PosBias_Timeout;
    if(HMC5883_Get_RawMagn(&Magn_PosBias.x, &Magn_PosBias.y, &Magn_PosBias.z)!= I2C_STATUS_OK)
      return  SENSOR_MagCalibGain_PosBias_ReadFail;
  }
  if(HMC5883_Configuration_A(HMC5883_SAMPLEAVERAGE_8, HMC5883_RATE_15, HMC5883_BIAS_NEGATIVE)!= I2C_STATUS_OK)
    return SENSOR_MagCalibGain_NegBias_Fail;

  // Disregarding the first data
  if(HMC5883_Get_RawMagn(&Tmp_Mag.x, &Tmp_Mag.y, &Tmp_Mag.z)!= I2C_STATUS_OK)
    return  SENSOR_MagCalibGain_NegBias_ReadFail;   ;

  // Read negative bias values
  zero_time = Get_msTick();
  Lowlimt = -200/HMC5883_Get_Gain();
  while (Magn_NegBias.x > Lowlimt | Magn_NegBias.y > Lowlimt| Magn_NegBias.z > Lowlimt)
  {
    if(Check_msTick(zero_time,10000))         return SENSOR_MagCalibGain_NegBias_Timeout;
    if(HMC5883_Get_RawMagn(&Magn_NegBias.x, &Magn_NegBias.y, &Magn_NegBias.z)!= I2C_STATUS_OK)
        return  SENSOR_MagCalibGain_NegBias_ReadFail;
  }

  MagScalerGain.x = (float) EKFQUA_MAGN_LIMIT/((Magn_PosBias.x  - Magn_NegBias.x)/2);
  MagScalerGain.y = (float) EKFQUA_MAGN_LIMIT/((Magn_PosBias.y  - Magn_NegBias.y)/2);
  MagScalerGain.z = (float) EKFQUA_MAGN_LIMIT/((Magn_PosBias.z  - Magn_NegBias.z)/2);

  if (PARAM_EEPROM_WRITE(MAGN_SCALEGAIN)!= PARAM_OK) return SENSOR_MagCalibGain_SaveFail;

  return SENSOR_OK;
}
```

Figure 60: magnetic gain estimation

The magnetic offset, which move the magnetometer data sphere to the origin:

```
SENSOR_Status SENSOR_Estimate_MagnOffset( uint8_t Stopbutton, RGB_Type led)
{
  int16_3D Tmp_Mag;
  uint64_t SENSOR_Lastime = Get_msTick();

  //setup for avr
  if(HMC5883_Set_SAMPLEAVERAGE(HMC5883_SAMPLEAVERAGE_8)!= I2C_STATUS_OK)
      return  SENSOR_MagCalibOffset_Fail;

  // Disregarding the first few data
  while(Get_msTick() - SENSOR_Lastime <1000)
  {
    if(HMC5883_Get_RawMagn(&Tmp_Mag.x, &Tmp_Mag.y, &Tmp_Mag.z)!= I2C_STATUS_OK)
      return  SENSOR_MagCalibOffset_ReadFail;
  }

  // colect data
  RGB_Set(led);
  while(Button_Check(Stopbutton) == IsReleased)
  {
    if(HMC5883_Get_RawMagn(&Tmp_Mag.x, &Tmp_Mag.y, &Tmp_Mag.z)!= I2C_STATUS_OK)
      return  SENSOR_MagCalibOffset_ReadFail;
      else
        {
          DATA_OUT.Magnx =  Tmp_Mag.x ;
          DATA_OUT.Magny =  Tmp_Mag.y ;
          DATA_OUT.Magnz =  Tmp_Mag.z ;

          Magn_max.x = max(Magn_max.x , Tmp_Mag.x);
          Magn_max.y = max(Magn_max.y , Tmp_Mag.y);
          Magn_max.z = max(Magn_max.z , Tmp_Mag.z);

          Magn_min.x = min(Magn_min.x , Tmp_Mag.x);
          Magn_min.y = min(Magn_min.y , Tmp_Mag.y);
          Magn_min.z = min(Magn_min.z , Tmp_Mag.z);
        }
  }

  // compute scale offset
  MagOffset.x = (((float)(Magn_max.x - Magn_min.x)/2)- Magn_max.x);
  MagOffset.y = (((float)(Magn_max.y - Magn_min.y)/2)- Magn_max.y);
  MagOffset.z = (((float)(Magn_max.z - Magn_min.z)/2)- Magn_max.z);

  if (PARAM_EEPROM_WRITE(MAGN_OFFSET)!= PARAM_OK)
    return SENSOR_MagCalibOffset_SaveFail;

  return SENSOR_OK;
}
```

Figure 61: magnetic offset estimation

The gyrometer can be also prevented from drifting by subtracting the bias:

```
////////////////////////////////////////////////
//  Gyro
//Convert to rad/s
Gyro->x = ((float)gx - GyroBias.x*(*SampleTime))  / (float)EKFQUA_GYRO_Sensitivity * DEG_TO_RAD ;
Gyro->y = ((float)gy - GyroBias.y*(*SampleTime)) / (float)EKFQUA_GYRO_Sensitivity * DEG_TO_RAD;
Gyro->z = ((float)gz - GyroBias.z*(*SampleTime)) / (float)EKFQUA_GYRO_Sensitivity * DEG_TO_RAD;
////////////////////////////////////////////////
```

Figure 62: gyrometer bias subtraction

Gyrometer bias can be estimate by following method:

```
SENSOR_Status SENSOR_Estimate_GyroBias(uint32_t gyro_second, uint8_t gyroButton, RGB_Type led)
{
  static int16_t gx, gy, gz;
  static int16_t ax, ay, az;
  float_3D bias;

  uint64_t SENSOR_Lastime = Get_msTick();
  uint64_t SENSOR_LastSample = Get_msTick();

  // Disregarding the first few data
  while(Get_msTick() - SENSOR_Lastime <3000)
  {
    if (MPU6050_Get_rawMotion (&gx, &gy, &gz, &ax, &ay, &az) != MPU6050_STATUS_OK)
      return SENSOR_GyroBias_Read_fail;
  }

  // colect data
  RGB_Set(led);
  SENSOR_Lastime = Get_msTick();
  while(Get_msTick() - SENSOR_Lastime <gyro_second)
  {
    if (Button_Check(gyroButton)== IsPressed) return SENSOR_OK;

    if (Get_msTick()-SENSOR_LastSample >= 10)
    {
      SENSOR_LastSample = Get_msTick();
      if (MPU6050_Get_rawMotion (&gx, &gy, &gz, &ax, &ay, &az) != MPU6050_STATUS_OK)
        return SENSOR_GyroBias_Read_fail;
      bias.x += gx;
      bias.y += gy;
      bias.z += gz;
    }
  }

  // compute scale offset
  GyroBias.x = (bias.x/gyro_second);
  GyroBias.y = (bias.y/gyro_second);
  GyroBias.z = (bias.z/gyro_second);

  if (PARAM_EEPROM_WRITE(GYRO_BIAS)!= PARAM_OK)
    return SENSOR_MagCalibOffset_SaveFail;

  return SENSOR_OK;
}
```

Figure 63: Gyrometer bias estimation

3.5 On-board software

The most feature of STM32F40x line is DMA, which stands for “Direct Memory Access”. As the name suggests, it has something to do with accessing memories, or more precisely with moving data between different locations in “memory space”. A very typical example would be reading the value of a receive register of a communication module such as a UART and storing it in a variable or array in RAM. Of course it can be done this with the CPU, so why do we need a special module for this transfer? The problem with using the CPU for this type of work is that it can actually tie up the CPU quite a bit. We would either have to constantly check if new data is available in the peripheral by reading and checking a status bit (or bits), or an interrupt (with the appropriate handler) could be used to alert us to the arrival of new data and to perform the transfer. This might be perfectly fine in the case of transferring a small number of bytes, but what if we would like to transfer larger amounts of data? In that case even the interrupt method can become quite inefficient, because there is some overhead associated with entering and exiting the interrupt handlers. So wouldn’t it be nice if there was a little helper module that we could tell to watch out for incoming bytes on the peripheral and to store them in some chunk of memory as they come in, and to only alert the CPU (if at all) if a certain number of bytes have been received? This is what a DMA controller is for: it allows us to transfer data without the involvement of the CPU. Essentially it allows us to use the resources in our microcontroller system more efficiently. It frees up the CPU for other tasks while data are being shuffled back and forth between memories and peripherals (or between memories).

Therefore, UART3 is configured with DMA for HC05 communication, TX and RX lines are on DMA1 stream 3 and stream 1 respectively. I2C1 and I2C2 are also configured with DMA1 stream 0, stream 2, stream 6 and stream 7 for MPU6050, HMC5583 and EPPROM 24LC256: DMA2 Stream 0 is for storing the DAC signal of voltage feed back.

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S2_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Figure 64: DMA 1 - Setup

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2				DCMI
Channel 2	ADC3	ADC3				CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4			USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX				USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3			TIM8_CH4 TIM8_TRIG TIM8_COM

Figure 65: DMA 2 - Setup

The program is design as below:

```
int main(void)
{
  Initialize_SerialLink();

  Initialize_Timer();

  Button_Initialize();

  RGB_Initialize();

  Initialize_PARAMETER();

  Voltage_FB_Initiallize();

  SENSOR_Calibrate(5*1000*60,0,1);

  while (1)
    {
      Algorithm_Orientiation();
      SerialLink_INPUT_Process();
    }
}
```

Figure 66: Main function

Serial Link to HC05 is firstly configured to send board events to the PC console:

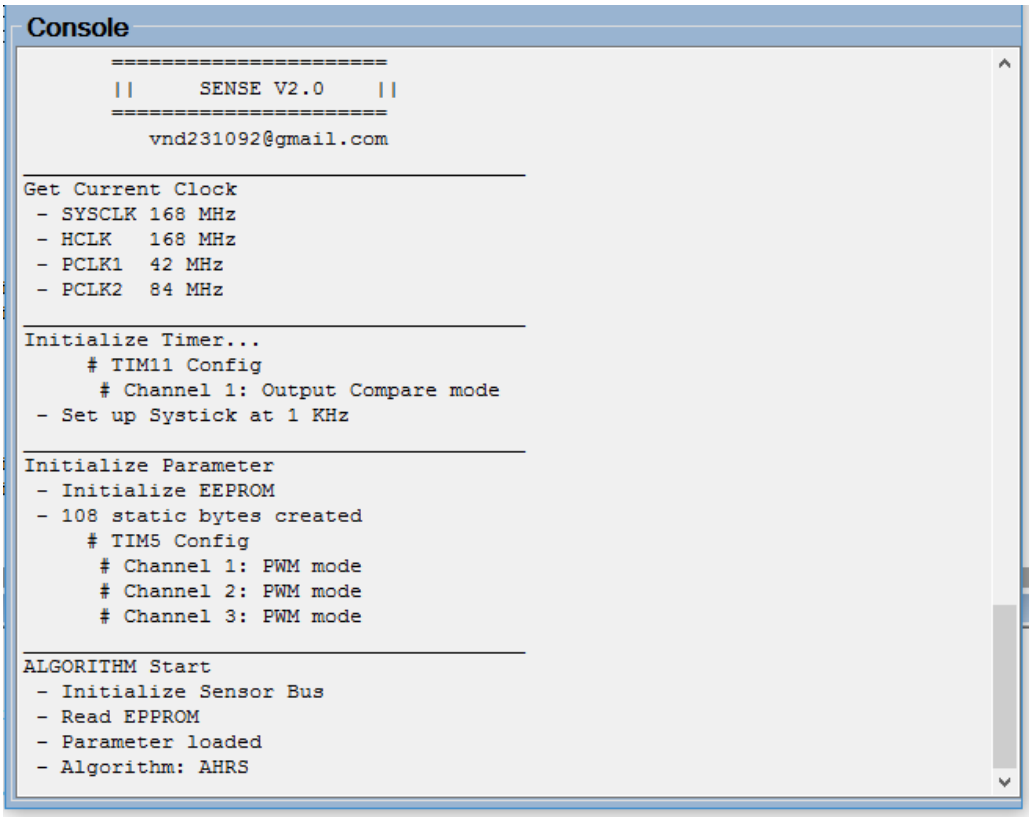


Figure 67: the SENSE sends debug lines to PC's console.

```
ALGORITHM_Status Algorithm_Process (void)
{
  switch (Algorithm)
  {
      case ALGORITHM_TiltComp   :////////////////////////////////////////////
                                Tilt_Compensate(&EulerRAD, &Raw_Acce, &Raw_Magn);
                                Convert_Rad_Quaternion(&EulerRAD, &Qua);
                                Convert_Rad_Deg(&EulerRAD, &EulerDEG);
                                break;
      case ALGORITHM_GyroInteg  :////////////////////////////////////////////
                                Quaternion_UpdateRate (&Qua, &Raw_Gyro, &Qua, Algorithm_dt);
                                Normalize_Quaternion(&Qua);
                                Convert_Quaternion_Deg(&Qua, &EulerDEG);
//                                Gyro_Integration(&EulerRAD, &Raw_Gyro, Algorithm_dt);
//                                Convert_Rad_Quaternion(&EulerRAD, &Qua);
//                                Convert_Rad_Deg(&EulerRAD, &EulerDEG);
                                break;
      case ALGORITHM_Comple     :////////////////////////////////////////////
                                Complementary_Estimate(&EulerRAD, &Raw_Gyro, &Raw_Acce, &Raw_Magn, Algorithm_dt)
                                Convert_Rad_Quaternion(&EulerRAD, &Qua);
                                Convert_Rad_Deg(&EulerRAD, &EulerDEG);
                                break;
      case ALGORITHM_AHRS       :////////////////////////////////////////////
                                AHRS_EstimateAttitude(&Qua, &Raw_Gyro, &Raw_Acce, &Raw_Magn, Algorithm_dt);
                                Convert_Quaternion_Deg(&Qua, &EulerDEG);
                                break;
      case ALGORITHM_QUAekf     :////////////////////////////////////////////
                                EKFQUA_Estimate(&Qua, &Raw_Gyro, &Raw_Acce, &Raw_Magn, Algorithm_dt);
                                Convert_Quaternion_Deg(&Qua, &EulerDEG);
                                break;
      case ALGORITHM_DCMekf     :////////////////////////////////////////////
                                EKFDCM_Estimate(&dcm, &Raw_Gyro, &Raw_Acce, &Raw_Magn, Algorithm_dt);
                                Convert_Dcm_Deg(&dcm, &EulerDEG);
                                Convert_Deg_Quaternion(&EulerDEG, &Qua);
                                break;
  }

  if ((Button_Check(0)== IsPressed) && (Button_Check(1)== IsPressed)) RGB_Flash(RGB_WHITE);
  else if (Button_Check(0) == IsPressed) RGB_Flash(RGB_GREEN);
  else if (Button_Check(1) == IsPressed) RGB_Flash(RGB_BLUE);
  SenseBat = Get_Voltage();

  // Serial Link Out
  SENSOR_DataAlign(Algorithm, &Raw_Acce, &Raw_Gyro, &Raw_Magn, &EulerDEG, &Qua, &dcm, &Algorithm_dt, &SenseBat);
      SDLOG_UpdateSensor(
                          Algorithm_dt,
                          Raw_Gyro.x, Raw_Gyro.y, Raw_Gyro.z,
                          Raw_Acce.x, Raw_Acce.y, Raw_Acce.z,
                          Raw_Magn.x , Raw_Magn.y , Raw_Magn.z
                         );
  return ALGORITHM_OK;
}
```

Figure 68: Different algorithms can be chose from PC application.

3.6 PC software

Microsoft Visual Studio is the best tool to develop win form application:

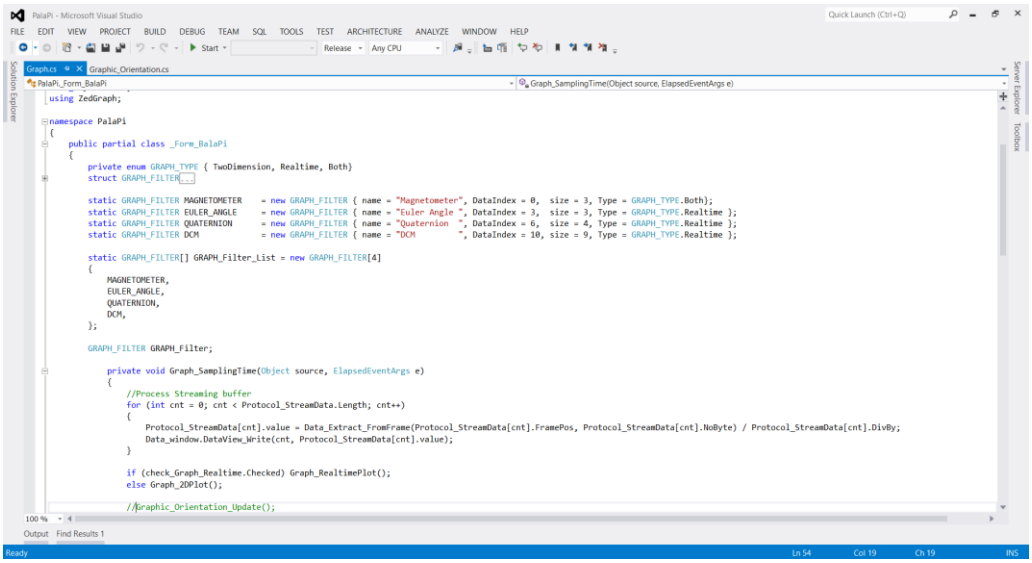


Figure 69: PC Software – Microsoft Visual Studio.

The win form application is divided into different part, doing specific tasks:

Firstly, the configuration is for adjust and connect to serial link:

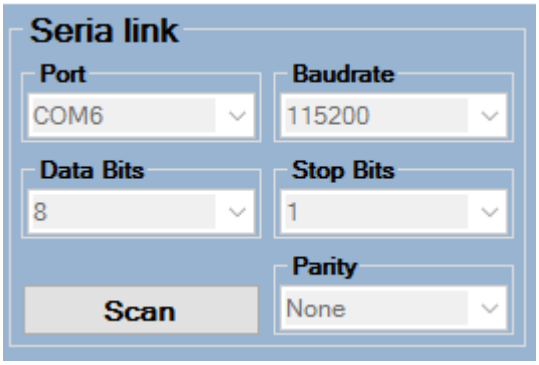


Figure 70: PC Software – configuration

Secondly, Parameter is for adjust filter parameters “on-the-fly”. “Refresh” means request parameters from the board, “Update” means apply new parameters to the board and ”Save “means tell the board to save current parameters to EPPROM:

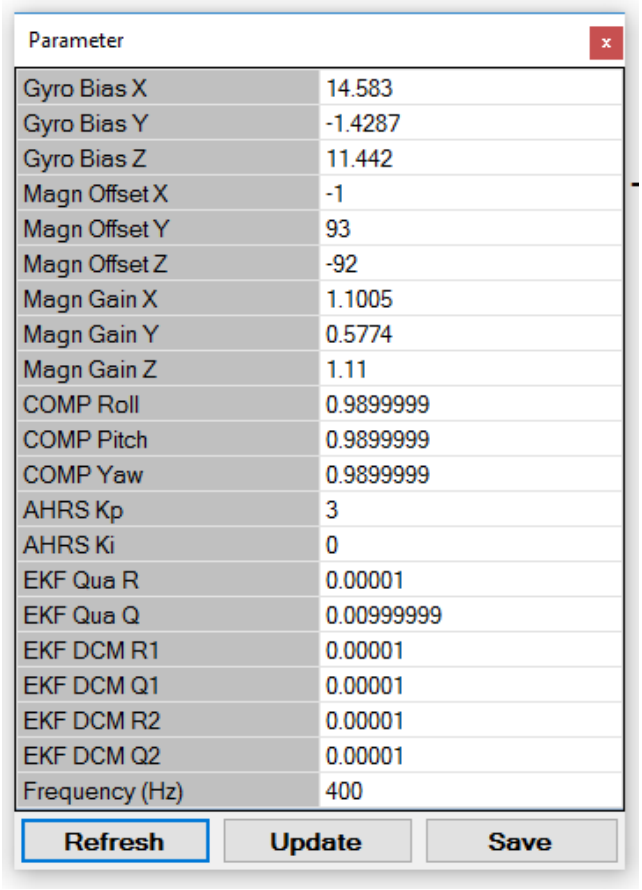


Figure 71: PC Software – Parameter

Thirdly, from “Algorithm” we can also change different filters that the board is using:

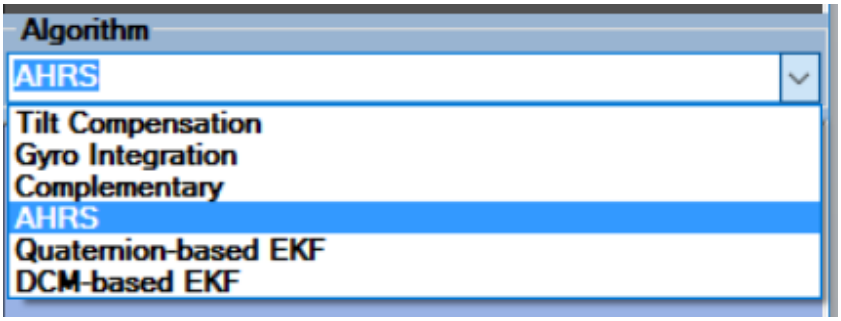


Figure 72: PC Software – Algorithm

Fourthly, data can be interpreted with different settings:

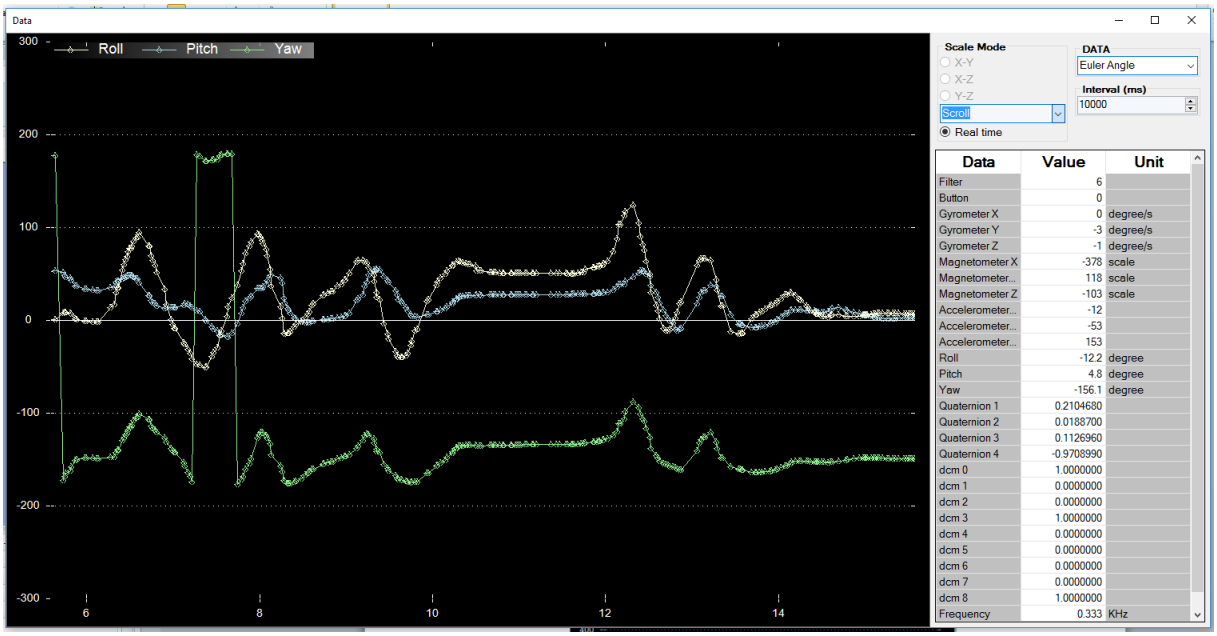


Figure 73: PC Software – graph and data

Finally, visualization is for showing algorithm result from board. X Y Z indicate the body frame axis of the sensor:

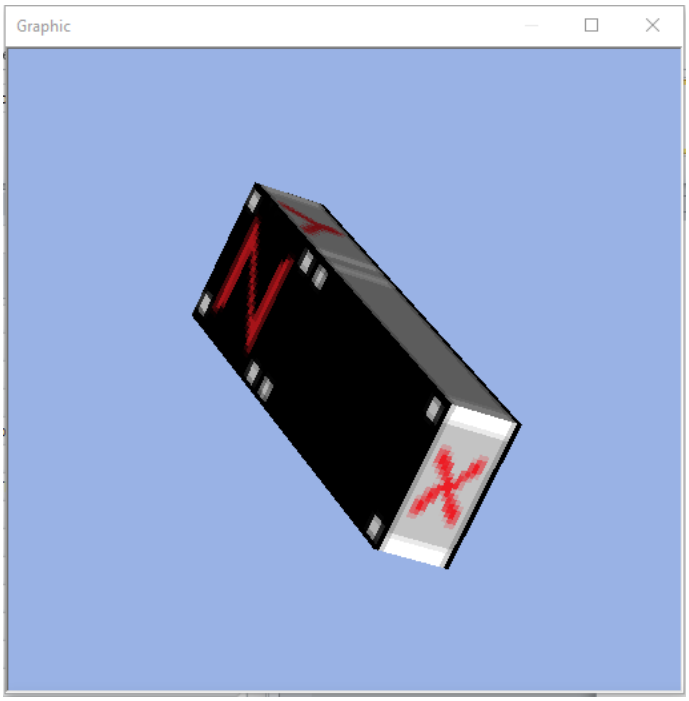


Figure 74: PC Software – visualization

The combination between The Sense and PC works stably:

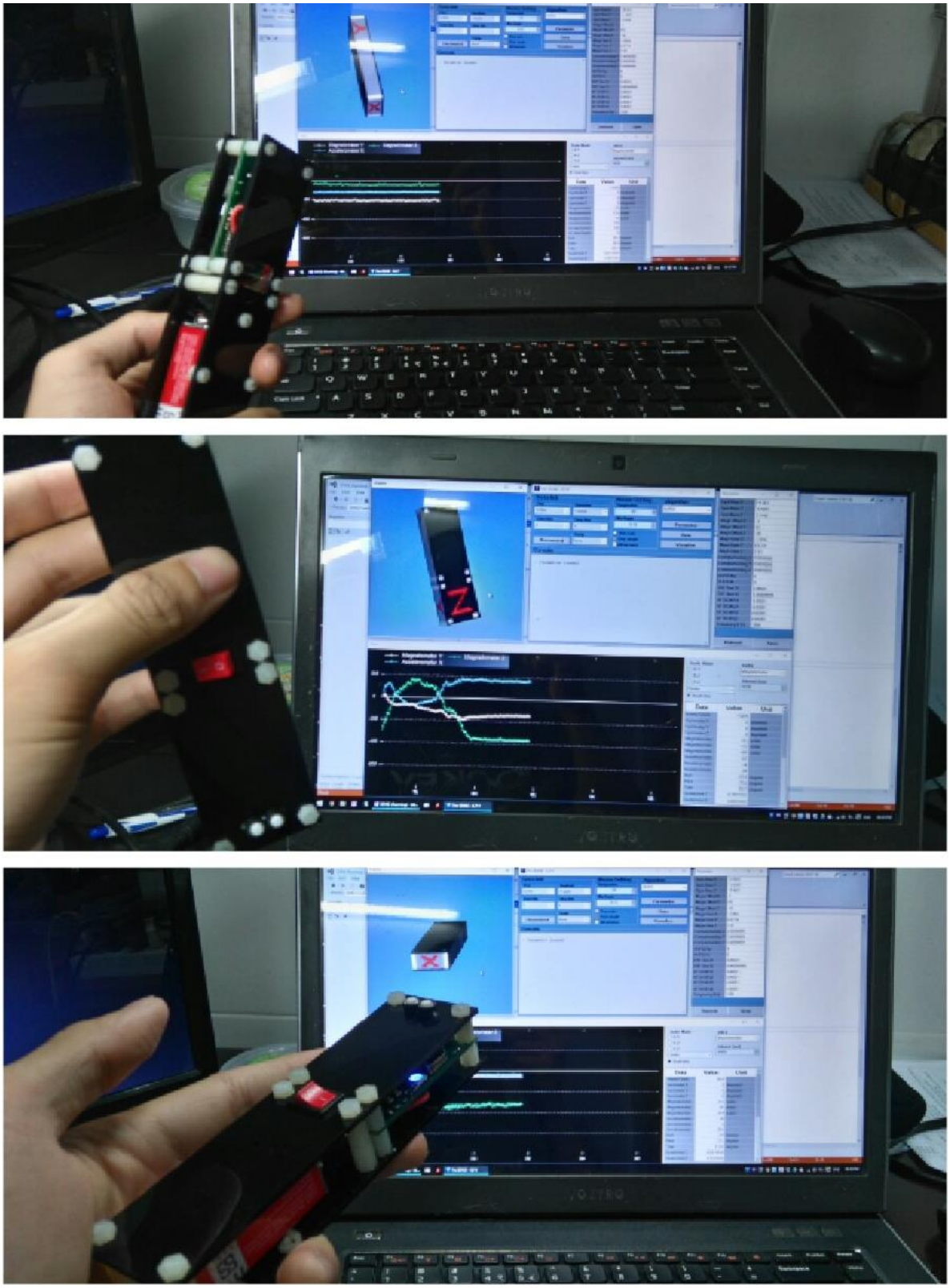


Figure 75: The combination between The Sense and PC

CHAPTER 4 CONCLUSION

4.1 Conclusion

The simulations proved Kalman filter gives much improvement than other calculation methods and a popular 9-axis filter AHRS.

Tilt-compensation is the method can be used as a pre-filter to feed into Quaternion-based EKF. Because of the computation is based on Euler system, it may cause gimbal lock problem. However, in some applications such as car and military truck's sensor, Tilt-compensation method is acceptable for its fast and simple.

Gauss Newton Method in contrast, can overcome the gimbal lock issue, however, it takes about 3 iterations, which is seventh times, to reach the level of Tilt-compensation method RMS result. It is necessary to mention that it is generally more accurate and stable than AHRS although it does not use gyrometer data.

The two **quaternion-based EKFs** are also compared using same modeling measurement, which is Tilt-compensation output. Obviously, they are both more accurate than mentioned calculation methods and filters. They have similar time consumptions. Personally, I prefer **Quaternion based – gyro rate – EKF** because it is slightly better performed than **Quaternion based – gyro bias – EKF** in most of the cases**.** In practice, **quaternion-based EKF** can be consider to use different modeling measurement depending on which application it is, such as QUEST, Gradient Descent, AHRS, which are in quaternion form but slow computed compared to Tilt-compensation.

DCM based KF is the super-fast with 90 micro-seconds. The reason is that it does not need linearizing and discretizing step like quaternion-based EKF. Moreover, it uses only three states for each sub-filter thanks to orthogonal characteristic. DCM based KF also performs the most accurate among those simulated. It firstly filters DCM based on angles rotate around X and Y axis, then use filtered data to correct the angle rotates around Z axis, while the quaternion-based EKF uses accelerometer and the magnetometer data to fix the drifting of gyrometer which lead to remarkable error in yaw. It can be clearly

seen from the simulation that the DCM based KF can suffer from the magnetic field environment.

The attempt to approach the real system is also successful that the board work stably. All components including STM32F40X, EPPROM, HC05 module, SD logging … work well without errors. The communication between the Sense and PC software was optimized, It can plug-and-play, user can adjust parameter while it working and observe data and its plots.

4.2 Future work

Because of the time limitation, this thesis focuses only on the basic level of Kalman filter. There are other approaches which is better than the discussed one, such as s the unscented Kalman filter, which is a relatively recent development that provides improved performance over the extended Kalman filter and widely used in Quadrotor:

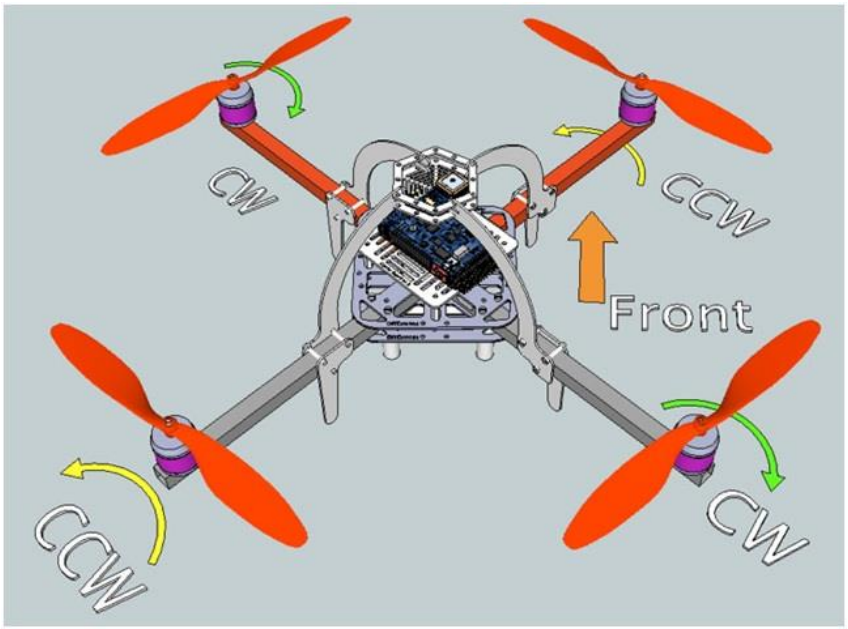


Figure 76: Quadrotor

REFERENCES

[1] Dan Simon (2006). Optimal State Estimation Kalman and Nonlinear Approaches. John Wiley & Sons.

[2] Edwin K.P. Chong, Stanislaw H.Zak (2008) An Introduction to Optimization. John Wiley & Sons.

[3] Greg Welch, and Gary Bishop (2006). An Introduction to the Kalman Filter.

[5] Daniel Roetenberg (2006). Inertial and Magnetic Sensing of Human Motion. PhD Thesis. University of Twente.

[5] Sebastian O.H. Madgwick (2010). Estimation of IMU and MARG orientation using a gradient descent algorithm. http://www.x-io.co.uk/.

[6] E. R. Bachmann et al (1999). "Orientation Tracking for Humans and Robots Using Inertial Sensors". In 1999 International Symposium on Computational Intelligence in Robotics & Automation (CIRA99).

[7] João Luís Marins et al. "An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors". In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[8] Xiaoping Yun et al (2005). "Implementation and Experimental Results of a QuaternionBased Kalman Filter for Human Body Motion Tracking". In Proceedings of the 2005 IEEE International Conference on Robotics and Automation Spain 2005.

[9] Xiaoping Yun et al (2006). "Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking". In IEEE Transactions On Robotics, 2006.

[10] Nguyen Ho Quoc Phuong et al. "Study On Orientation Estimation With Three Different Representations". In Proceedings of the International Symposium on Electrical & Electronics Engineering 2007, Vietnam.