Quaternion-based Extended Kalman Filter for IMU tutorial

Dang Lam Tung
danglamtung199@gmail.com

Faculty of Electrical and Electronic Engineering
Ho Chi Minh University of Technology

September 27, 2019

Contents

1 Introduction

This is tutorial about the origin, theory and designing of Kalman Filter and Extended Kalman Filter, a useful tool for filtering and sensor fusion . In this tutorial we will cover the origins of Kalman Filter from linear systems origin, (the Bayes origin will be cover in later version). Then we will investigate how to design an extended Kalman Filter from an example for quarternion for IMU fusion problem. The writer not expected this document is a complete guide to Kalman Filter, but a useful note for everyone who want to implement an Extended Kalman Filter. I want to thank for the PIF club, the Student Research Club of Faculty of Electrical and Electronic Engineering of HCM University of Technology to give me the passion to complete this document. If there is any error, please feel free to correct me.

2 The Kalman Filter

2.1 Noise and filter

Imagine you have an sensor, any type maybe an heart rate sensor, and temperature sensor or even an expensive GPS tracker, no matter how advanced or costly they are, they will always be suffered from noise, may be from sensor itself or the environment. In fact, the more sensitive is the sensor, the more noise it gets. So in reality we always need filter to correct the noisy data. There are a lot of filter types for many type of noise. but base on probability there are 2 type of noise: white Gaussian noise and colored noise.

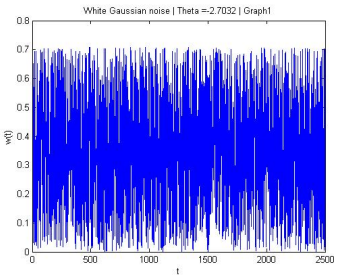

Fig 1: The white Gaussian noise

White Gaussian noise is non correlated to the distribution of the signal, zero mean and its distribution is Gaussian. WGN (white Gaussian noise) has a lot of sources and this is the most common noise source. It has that is because of the central probability theorem, if a distribution is prolong enough, it is considered the Gaussian distribution.

Colored noise is correlated with the signal, and its distribution may not be the Gaussian, to filter this kind of noise, we need special designed filter.

In term of frequency, we have high frequency noise and low frequency noise. An example of high frequency noise is the noise of the accelerometer in the

IMU system we will concern later, the low frequency noise is the noise of the gyroscope in the IMU system, it come from incremental effect of gyroscope, not from the sensor itself. To deal with noise, we have to use the filters like low-pass filter, high-pass filter, band-pass filter,...etc but in many application only one sensor is not enough cause the properties of each sensor is different, we can conduct the "sensor fusion" to get a better estimation, that is the point of complemetary filter and kalman filter is also be a very good (and popular) tool for the sensor fusion problem. So we will trace the creation of Kalman filter from its root: linear system and estimation.

2.2 Least squared estimation

The following theory of Kalman Filter and Extended Kalman Filter is based on [1]

Imagine there is a constant you want to measure, maybe height of a drone, the temperature of your house,... etc, we will call that constant is x. To measure an constant x, we will use some sort of sensors to measure x, and the result of that measurement is y, in this section we will consider the measurement of sensors is a linear combination of x and the measurement constraint $H_1, H_2, ...Hk$, and a noise of measurement is v So $y = H_1x + H_2x + ... + H_kx + v$

Generally, we can consider x is a constant state vector instead of a single constant, denote $x = [x_0, x_1, ..., x_n]$, and the k-element measurement vector is $y = [y_0, y_1, ...y_k]$ $(k, n > 0)$ (in this case we consider there is no noise in the measurement), the measurement equation is

$$\begin{aligned}
y_0 &= H_{11}x_0 + H_{12}x_1 + ... + H_{1n}x_n + v_0 \\
y_1 &= H_{21}x + H_{22}x_1 + ... + H_{2n}x_n + v_1 \\
&.... \\
y_k &= H_{k1}x + H_{k2}x_1 + ... + H_{kn}x_n + v_k
\end{aligned}$$

This combination can be write by this matrix form:

$$y = Hx \tag{1}$$

In which H is the observation matrix, has the dimension of (k, n).

$$H = \begin{pmatrix} H_{11} & H_{12} & H_{13} & ...H_{1n} \\ H_{21} & H_{22} & H_{23} & ...H_{2n} \\ . & .. & .. & . \\ H_{k1} & H_{k2} & .. & H_{kn} \end{pmatrix}$$

So base on measurement vector and the observation matrix H, we could estimate the state vector x, we will call the estimated vector by (1) is $\hat{x}$, we will have:

$$y = H\hat{x}$$

We define the different between the measurement and the $H\hat{x}$:

$$e_y = y - H\hat{x}$$

The most probable value of x is the vector $\hat{x}$ that minimizes the sum of squared between of the observed value y and the vector $H\hat{x}$. So we will try to compute the $\hat{x}$ that minimizes the cost function L, where L is given as

$$L = e_{y1}^2 + e_{y2}^2 + \ldots + e_{yk}^2$$

$$= e_y^T e_y$$

$$L = e_y^T e_y = (y - H\hat{x})(y - H\hat{x})^T = y^T y - \hat{x}^T H^T y - y^T H\hat{x} + \hat{x}^T H^T H\hat{x}$$

To minimize the cost function L respect to $\hat{x}$, we compute its partial derivative and set it equal to zero:

$$\frac{\partial L}{\partial \hat{x}} = -y^T H - y^T h + 2\hat{x}^T H^T H = 0$$

Solving this equation for $\hat{x}$ results in

$$H^T y = H^T H\hat{x}$$

$$\hat{x} = (H^T H)^{-1} H^T y$$

This can be the minimum of L or some other type of stationary point of L, we need to prove this is the minimum by computing the second derivative of L.

But in recently case, the noise v is not considered, in real world there is nothing such as a non noisy system, the noise can come from many source, may be a bad sensor or a sleepy student forget something,... etc but each piece of information is precious, no matter how noisy it is, so in this section we will find a way to use that noisy information. We will assume that x is a constant vector, y is a k-element noisy measurement vector and has a linear combination with x, and the noise vector v has k-element is $v = [v_0, v_1, ..., v_k]$

We will assume that v has zero mean and independent Gaussian distribution, the covariance matrix is

$$R = E(vv^T) = \begin{pmatrix} \sigma_1^2 & 0 & 0 & ...0 \\ 0 & \sigma_2^2 & 0 & ...0 \\ 0 & .. & .. & 0 \\ 0 & .. & .. & \sigma_k^2 \end{pmatrix}$$

Our goal is to minimize the loss function

$$L = e_1^2/\sigma_1^2 + e_2^2/\sigma_2^2 + \ldots + e_n^2/\sigma_n^2 \tag{2}$$

But why? Let consider the i measurement which has the covariance of noise is σ_i, we can see that the larger the σ_i is the more unreliable the i measurement is, so the loss function of this measurement is smaller, mean wah, we don't even need to minimize this measurement anymore cause it is bad anyway and vice versa, if the measurement is not very noisy then its information can be trustworthy.

The matrix form of the loss function is

$$\begin{aligned} L &= e^T R^{-1} e \\ &= (y - H\hat{x})^T R^{-1} (y - H\hat{x}) \\ &= y^T R^{-1} y - \hat{x}^T H^T R^{-1} y - y^t R^{-1} H x' + \hat{x}^T H^T R^{-1} H \hat{x} \end{aligned}$$

To find the best $\hat{x}$ possible, we compute the derivative of L by $\hat{x}$

$$\begin{aligned} \frac{\partial L}{\partial \hat{x}} &= -2y^T R^{-1} H + 2\hat{x}^T H^T R^{-1} H = 0 \\ &\rightarrow \hat{x} = (H^T R^{-1} H)^{-1} H^T R^{-1} y \end{aligned} \tag{3}$$

So this this how we find the best estimation $\hat{x}$ for the given system, but in reality, to compute $\hat{x}$ this way, we has to recompute the equation (3) each time, and that is not very memory and time efficiency, so we has to find a way to recursively computing the $\hat{x}$:

$$\begin{aligned} y_k &= H_k x + v_k \\ \hat{x}_k &= \hat{x}_{k-1} + K_k(y_k - H\hat{x}_{k-1}) \end{aligned}$$

In which k is the iteration step, K_k is called the estimator gain matrix.

So the estimation error's mean is

$$\begin{aligned} E(e_x, k) &= E(x–\hat{x}_k) \\ &= E(x - \hat{x}_{k-1} - K_k(y_k - H_k\hat{x}_{k-1}) \\ &= E(e_{x,k-1} - K_k(H_k x + v_k - H_k x' k - 1)) \\ &= E(e_{x,k-1} - K_k H_k(x - \hat{x}_{k-1}) + K_k v_k) \\ &= (I–K_k H_k)E(e_{x,k-1}) + K_k E(v_k) \end{aligned}$$

Our loss function in this case is

$$\begin{aligned} J_k &= E((x_1 - \hat{x}_1)^2) + E((x_2 - \hat{x}_2)^2) + \ldots + E((x_k - \hat{x}_k)^2) \\ &= E[(e_{x_1,k})^2 + \ldots . + (e_{x_n,k})^2] \\ &= E[e_{x,k}^T e_{x,k}] \\ &= E(Tr(e_{x,k}, e_{x,k}^T)) \end{aligned}$$

The transition of P_k through time steps is

$$P_k = E[(I - K_k H_k)e_{x,k-1}–K_k v_k)(I - K_k H_k)e_{x,k-1}–K_k v_k)^T] \tag{4}$$

$$\begin{aligned} P_k = (I–K_k H_k)E(e_{x,k-1} e_{x,k-1}^T)(I–K_k H_k)^T \\ - K_k E(v_k e_{x,k-1}^T)(I - K_k H_k)^T \\ - (I–K_k H_k)E(e_{x,k-1} v^T)K_k^T \\ + K_k(v_k v_k^T)K_k^T \end{aligned}$$

Because the probability independence of two covariance, we have:

$$E(e_{x,k-1}v_k^T) = 0$$

So P_k update equation is actually

$$Pk = (I–K_kH_k)P_{k-1}(I–K_k * H_k)^T + K_k * R_k * K_k^T \tag{5}$$

This is how we can calculate the P_k recursively, and P_k is the covariance matrix of the state $\hat{x}$, all of its values must be positive. Now we have to find the value of K_k based on P_k to make the loss function small as possible (to know why please read the appendix A)

$$\begin{aligned} <=> \frac{\partial J_k}{\partial K_k} &= 2(I–K_kH_k)P_{k-1}(-H_k^T) + 2K_kR_k = 0 \\ <=> K_kR_k &= (I - K_kH_k)P_{k-1}H_k^T \\ <=> K_k(R_k + H_kP_{k-1}H_k^T) &= P_{k-1}H_k^TK_k = P_{k-1}H_k^T(R_k + H_kP_{k-1}H_k^T)^{-1} \end{aligned}$$

To summary the full algorithm will be:
1. Initialize the estimator when k = 0:

$$\begin{aligned} x_0 &= E(x_0) \\ P_0 &= E[(x_0–\hat{x_0})(x - \hat{x_0})^T] \end{aligned}$$

2. $k > 0$ we have:

$$y_k = H_kx + v_k$$

v_k is zero-mean random vector with covariance Rk (WGN)
Update equation is:

$$\begin{aligned} K_k &= P_{k-1}H_k^T(R_k + H_kP_{k-1}H_k^T)^{-1} \\ P_k &= (I–K_kH_k)P_{k-1}(I–K_kH_k)^T + K_kR_kK_k^T \\ \hat{x_k} &= \hat{x_{k-1}} + K_k(y_k - H_kx_{k-1}) \end{aligned}$$

So, this is the least squared estimator.

2.3 Discrete-time systems update

Suppose we have a Discrete-time system:

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1}$$

Where u_k is the input and w_k is the zero-mean white Gaussian noise with covariance of Q_k. We will find out how x_k state change with time We call $\bar{x_k}$ is

the expected value of x_k then because the system is linear, we can compute the $barx_k$ through time like this (we don't consider the noise factor here):

$$\bar{x}_k = F_{k-1}\bar{x}_{k-1} + G_{k-1}u_{k-1}$$

The covariance of x_k though time is

$$\begin{aligned}(x_k – \bar{x_k})(xk – \bar{x_k})^T &= (F_{k-1}x_{k-1} + Gu_{k-1} + w_{k-1} - \bar{x_k})(..)^T \\ &= [F_{k-1}(x_{k-1} - \bar{x_{k-1}}) + w_{k-1}][...]^T \\ &= F_{k-1}(x_{k-1} - \bar{x}_{k-1})(x_{k-1} - \bar{x}_{k-1})^T F_{k-1}^T + w_{k-1}w_{k-1}^T + ...\end{aligned}$$

The rest are not show because $(x - \bar{x})$ and w_{k-1} are probability independent so when compute the covariance matrix P_k they are equals to zero:

$$=> P_k = E[(x_k – \bar{x_k})(xk – \bar{x_k})^T] = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}$$

This is called the Lyapunov equation.

2.4 Discrete-time Kalman Filter

Kalman Filter is the combination of the least squared estimation and a (non) linear model of the system, the effect of this combo is that when an unexpected accident happen to measurement method, the model will "correct" the error happened and vice versa.

To prove the Kalman filter, suppose we have a linear discrete system as:

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

The noise w_{k-1}, v_k is white gaussian, non-correlated and has covariance:

$$\begin{aligned}w_k &= (0, Q_k) \\ v_k &= (0, R_k) \\ Q_k = E(w_k w_k^T) &= \begin{pmatrix} Q_1 & ... & 0 \\ . & . & . \\ 0 & ... & Q_n \end{pmatrix} \\ R_k = E(v_k v_k^T) &= \begin{pmatrix} R_1 & ... & 0 \\ . & . & . \\ 0 & ... & R_n \end{pmatrix}\end{aligned}$$

We denote the x_k^+ is the posteriori state estimation, the estimation of xk base on the linear model of the system up to time k (include time k):

$$x_k^+ = E[x_k / y_1, y_2, ...y_k]$$

If we have all of the measurement before the time k, then we can form a priori estimation state

$$x_k^- = E[x_k/y_1, y_2, ...y_{k-1}]$$

Summary: x_k^- = estimation of x_k before we process the measurement at time k
x_k^+ = estimation of x_k a after we process the measurement at time k To smooth our measurement or predict , we can use a prediction and combine it with our measurement state estimation:

$$x_{k/k+N} = E[x_k/y_1, y_2, ...y_{k+n}]$$
$$x_{k/k-N} = E[x_k/y_1, y_2, ...y_{k-n}]$$

We use the term P_k to denote the covariance of the estimation error, P_k^- denote the covariance of the estimation error of x_k^- and P_k^+ denote the covariance of x_k^+:

$$P_k^- = E[(x_k - \hat{x_k}^-)(x_k - \hat{x_k}^-)^T]$$
$$P_k^+ = E[(x_k - \hat{x_k}^+)(x_k - \hat{x_k}^+)^T]$$

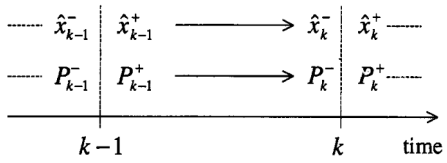

Figure 5.2 Timeline showing *a priori* and *a posteriori* state estimates and estimation-error covariances.

Fig 2 :The relationship of priori and posteriori can be seen in this image. *This is a picture from the Optimal State Estimation book, Dan Simon* So, we begin with x_0, we initial the value of the filter like this

$$x_0^+ = E(x_0)$$

This is make sense because we want to initial the x_0 the starting value of the system. Next we want to find the time update equation of P, the covariance of x_0. If we know the starting state perfectly, $P_0^+ = 0$, if we have absolutely no idea about the value of x_0 then $P_0^+ = \infty I$. In general, P_0 represents the uncertainty in our initial estimation of x_0.

$$P_0^+ = E[(x_0 - \hat{x_0}^+)(x_0 - \hat{x_0}^+)^T]$$

Then based on our discrete-time linear system model we have:

$$x_k^- = F_{k-1}x_{k-1}^+ + G_{k-1}u_{k-1}$$
$$P_k^- = F_{k-1}P_{k-1} + F_{k-1}^T + Q_{k-1}$$

This is called the update equation for $\hat{x}$ and P . From the time $(k-1)^+$ to k^- we don't have any state estimation for $\hat{x}$ so we have to update the state estimate based on our knowledge of the system dynamics in the linear model form.

From the time k^- to k^+, we conduct the estimation state with the least squared estimation which take the new measurement y_k in to account:

$$\begin{aligned}
K_k &= P_{k-1}^- H_k^T (H_k P_{k-1} H_k^T + R_k)^{-1} \\
x_k^+ &= \hat{x_k}^- + K_k(y_k - H_k \hat{x}_k^-) \\
P_k^+ &= (I–K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T
\end{aligned}$$

And that end the update cycle of the predict - measurement or we can call the prediction - correction system:

To summarized the Discrete-time Kalman Filter: 1. The dynamic system is given by the following equations:

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

2. The Kalman Filter is initialized as follows:

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

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

The prediction update:

$$\begin{aligned}
x_k^- &= F_{k-1} x_{k-1}^+ + G_{k-1} u_{k-1} \\
P_k^- &= F_{k-1} P_{k-1} + F_{k-1}^T + Q_{k-1}
\end{aligned}$$

The measurement update:

$$\begin{aligned}
K_k &= P_{k-1}^- H_k^T (H_k P_{k-1} H_k^T + R_k)^{-1} \\
x_k^+ &= \hat{x_k}^- + K_k(y_k - H_k \hat{x}_k^-) \\
P_k^+ &= (I–K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T
\end{aligned}$$

2.5 Properties of Kalman Filter

+ P will converge through time and P is the way to evaluate the "correctness" of the measurement + Kalman filter is the best linear filter because it has the combination of the model and the measurements. There maybe a better non

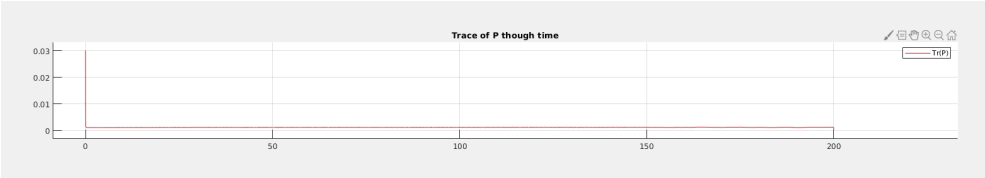

Fig 3: The converge of P

linear filter but Kalman filter is the best linear filter even if the noise is not Gaussian.

+ For non linear systems, there are approximate solution like Taylor expansion.

+ In fact the Kalman Filter can be derived as a filter that whiten the measurements and extract the maximum information possible.

2.6 Divergence issues

If your Kalman filter does not work, try:

+ Recheck the model F, Q, H and R, the model is important for the converge of the P, if P does not converge :3 try recompute the Q and R. Note that the filter consider Q and R is white, zero-mean and completely uncorrelated, if some of that condition does not meet, the filter won't work.(actually there is a kind of Kalman Filter can handle this situation, but not our interest).

+ Kalman filter can suffer from the arithmetic precision.

+ Initialize P properly, P is too large or too small can lead to unstable in the state estimation.

3 Quaternion Based Extended Kalman Filter for estimating orientation.

3.1 The Discrete-time Extended Kalman Filter

In reality, there is no thing such as a "linear system", all system are non linear, but we can use the Kalman Filter to estimate these non-linear system by using the Taylor Expansion to linearize the equation. Suppose we have the non-linear system like this:

$$x_k = f_k(x_{k-a}, u_{k-1}, w_{k-1}) \tag{6}$$

$$y_k = h_k(x_k, v_k) \tag{7}$$

$$w\ (0, Q_k) \tag{8}$$

$$v\ (0, R_k) \tag{9}$$

We perform the first order Taylor expandsion for the state equation around $x_{k-1} = \hat{x}_{k-1}^+$ and $w_{k-1} = 0$ to obtain:

$$x_k = f_k(x_{k-1}^+, u_{k-1}, 0) + \frac{\partial f_{k-1}}{\partial x}|_{\hat{x}_{k-1}^+}(x_{k-1} - \hat{x}_{k-1}^+) + \frac{\partial f_{k-1}}{\partial w}|_{\hat{x}_{k-1}^+}(w_{k-1} - 0)$$

Denote $\frac{\partial f_{k-1}}{\partial x}|_{\hat{x}_{k-1}^+} = F_{k-1}$, $\frac{\partial f_{k-1}}{\partial w}|_{\hat{x}_{k-1}^+} = L_{k-1}$:

$$\begin{aligned} x_k &= f_k - 1(x_{k-1}^+, u_{k-1}, 0) + F_{k-1}(x_{k-1} - x_{k-1}^+) + L_{k-1}w_{k-1} \\ &= F_{k-1}x_{k-1} + [f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) - F_{k-1}\hat{x}_{k-1}^+] + L_{k-1}w_{k-1} \\ &= F_{k-1}x_{k-1} + \bar{u}_{k-1} + \bar{w}_{k-1} \end{aligned}$$

In which $\bar{u}_{k-1} = [f_{k-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) - F_{k-1}\hat{x}_{k-1}^+]$, $\bar{w}_{k-1} = L_{k-1}w_{k-1}$ The signal $\bar{u}_k$ and the noise signal u_k are defined as:

$$\bar{u}_k = f_k(\hat{x_k}^+, u_k, 0) - F_k\hat{x_k}^+$$

$$\bar{w}_k \sim (0, L_kQ_kL_k^T)$$

Which measurement equation, we use the first order Taylor expansion around $x_k = \hat{x}_k^-$ and $v_k = 0$ to obtain:

$$\begin{aligned} y_k &= h(\hat{x}_k^-, 0) + \frac{\partial h_k}{\partial x}|_{\hat{x}_k^-}(x_k - \hat{x}_k^-) + \frac{\partial h_k}{\partial x}(v_k - 0) \\ &= h_k(\hat{x}_k^-, 0) + H_k(x_k - \hat{x}_k^-) + M_kv_k \\ &= H_kx_k + [h_k(\hat{x}_k^-, 0) - H_k\hat{x}_k^-] + M_kv_k \end{aligned}$$

Just like the state equation, we define $z_k = [h_k(\hat{x}_k^-, 0) - H_k\hat{x}_k^-]$

$$= H_kx_k + z_k + \bar{v}_k$$

z_k and $\bar{v}_k$ is defined as:

$$z_k = h_k(\hat{x_k}^-, 0) - H_k\hat{x_k}^-$$

$$\bar{v}_k \sim (0, M_kR_kM_k^T)$$

So the Discrete-time Extended Kalman Filter can be summarized as: 1. The dynamic system is given by the following equations:

$$\begin{aligned} x_k &= f_k(x_k, u_k, k) + w_k \\ y_k &= h_k(x_k, k) + v_k \\ w_k &\sim (0, Q_k) \\ v_k &\sim (0, R_k) \end{aligned}$$

2. The Kalman Filter is initialized as follows:

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

3. The Kalman Filter is given by the following equations, which are computed for each time step k = 1,2,...: a) Compute the partial derivative matrices of the state equation:

$$F_{k-1} = \frac{\partial f_{k-1}}{\partial x}|_{\hat{x}^+_{k-1}}$$

$$L_{k-1} = \frac{\partial f_{k-1}}{\partial w}|_{\hat{x}^+_{k-1}}$$

b) Perform the time update of the state estimate and estimation error covariance:

$$x^-_k = f_k(x^+_{k-1}, u_{k-1}, 0)$$

$$P^-_k = F_{k-1}P_{k-1} + F^T_{k-1} + L_{k-1}Q_{k-1}L^T_{k-1}$$

c) Compute the partial derivative matrices of the measurement equation:

$$M_k = \frac{\partial h_k}{\partial x}|_{\hat{x}^+_{k-1}}$$

$$L_{k-1} = \frac{\partial h_k}{\partial w}|_{\hat{x}^+_{k-1}}$$

d)Perform the measurement update of the state estimate and state measurement covariance as follows:

$$K_k = P^-_{k-1}H^T_k(H_kP_{k-1}H^T_k + M_kR_kM^T_k)^{-1}$$
$$x^+_k = \hat{x_k}^- + K_k(y_k - h_k(\hat{x}^-_k, 0))$$
$$P^+_k = (I–K_kH_k)P_k$$

And that is our Extended Kalman Filter.

So, in reality, we have:

3.2 The quaternion Extended Kalman Filter for IMU

The EKF model in thos section is based on the [2].

The IMU or the inertial measurement unit is a combination of sensor which give us the information of the object's orientation in 3D space, an IMU system consist of (a) gyroscopes, (an) accelerometers and maybe (a) magnetometer. The fusion data of these sensor give us the orientation or even better, which high-end sensors, we can even measure the accelerate and integral it to achieve the reference location of an object (the inertial navigation system), which often go with an GPS system. IMU is a must-have in many application, like SLAM, self driving car, drone,.... and Extended Kalman Filter is the best way for fusing IMU sensors. That why we will use this problem for an example for Extended Kalman Filter.

The state vector is the combination of quaternion (if you don't know what is quaternion, see appendix A) q and gyroscope bias b_ω (if you don't know what is gyroscope bias, please look at appendix B), total is 7 elements (and if you

think 7 is many, remember that in Ardupilot, their EKF has a 22 variable state vector).

$$x = [q, b_\omega] \tag{10}$$

From kinematic equation, we have

$$\phi = \phi + (\omega - b_\omega)dt \tag{11}$$

We know that:

$$\frac{dq}{dt} = 1/2q\omega \tag{12}$$

From the definition of gradient, we have

$$q' = (q_k - q_{k-1})/dt = \frac{1}{2}q_{k-1}\omega$$
$$=> q_k = (I + \frac{1}{2}\omega\Delta t)q_{k-1}$$

We know from previous section that the quaternion multipication can be represent by a matrix multipication, we denote as

$$\frac{1}{2}q\omega = \frac{1}{2}S(q)q = \frac{1}{2}S(\omega)\omega \tag{13}$$

Which

$$S(q) = \begin{bmatrix} -b & -c & -d \\ a & -d & c \\ d & a & -b \\ -c & -b & a \end{bmatrix}$$

Then we can create the state transition matrix:

$$A = \begin{pmatrix} I_{4x4} & -0.5 * S(q) * dt \\ 0_{3x3} & I_{3x3} \end{pmatrix}$$

$$B = \begin{pmatrix} 0.5 * Sq * dt \\ 0_{3x3} \end{pmatrix}$$

$$\omega_k = [Gx_k, Gy_k, Gz_k]$$

If this seen magical to you, just write down the equation and you will see the magic.

Model equation:

$$x_k = A_k x_{k-1}^+ + B_k\omega_k$$
$$P_k^- = A_k P_{k-1} + A_k^T + Q_k$$

For the measurement route: We have the rotation matrix with quaternion:

$$R(q) = \begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & a^2 - b^2 - c^2 + d^2 \end{pmatrix}$$

$$a_k^i = C_a(q_k) + b_a + e_a$$

With the a_k^i is the inner frame accelerate of the sensor.
$C_a(q_k)$ is the quaternion rotation matrix at time k multiplied with the reference accelerate of the gravity.
b_a is the bias of the accelerometer.
e_a is the noise of the accelerometer.
For accelerometer, we have the gravitation force downward vector is [0 0 1] (consider the range of accelerometer is [0:1], this can be [0 0 g] based on the range of the accelerometer)

$$C_a(q_k)g = -\begin{bmatrix} 2(bd–ac) \\ 2(cd+ab) \\ a^2-b^2-c^2+d^2 \end{bmatrix}$$

To linearlize this Equation, we use the Taylor expansion:

$$f(x)|_{x-a} = f(a) + f'(x)(x-a) + \frac{f''(x)}{2}(x-a)^2 + ... + \frac{f^n(x)(x-a)^n}{n!}$$

Using the Taylor expansion for $C_a(q_k)$ we achieve:

$$C_a(q_k)|_{q_k=q_{k-1}} = C_a(q_{k-1}) + C_a(q_k)'(q_k - q_{k-1}) + ...$$

$$C_a(q_k)' = \frac{\partial C_a(q_k)}{\partial q_k} = \begin{pmatrix} \frac{\partial C_{a1}}{\partial a} & \frac{\partial C_{a1}}{\partial b} & \frac{\partial C_{a1}}{\partial c} & \frac{\partial C_{a1}}{\partial d} \\ \frac{\partial C_{a2}}{\partial a} & \frac{\partial C_{a2}}{\partial b} & \frac{\partial C_{a2}}{\partial c} & \frac{\partial C_{a2}}{\partial d} \\ \frac{\partial C_{a3}}{\partial a} & \frac{\partial C_{a3}}{\partial b} & \frac{\partial C_{a3}}{\partial c} & \frac{\partial C_{a3}}{\partial} \end{pmatrix}$$

This is called the Jacobian matrix of C_a, with the help of Matlab, the result is:

$$\frac{\partial C_a(q_k)}{\partial q_k} = -2\begin{pmatrix} 2c & 2d & 2a & 2b \\ -2b & -2a & 2d & 2c \\ 2a & -2b & -2c & 2d \end{pmatrix}$$

The way to calculate the Jacobian matrix is using the jacobian function of matlab:

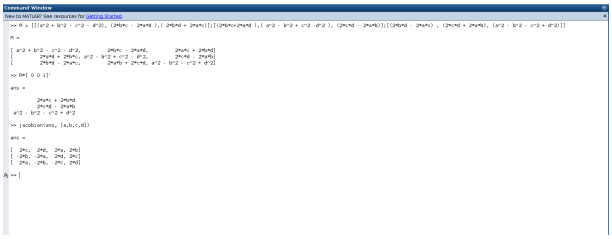

Fig 4: Computing the jacobian

So the measurement model is:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_k = \left(\begin{bmatrix} 2(bd–ac) \\ 2(cd+ab) \\ a^2-b^2-c^2+d^2 \end{bmatrix}_k + \begin{bmatrix} 2c & 2d & 2a & 2b \\ -2b & -2a & 2d & 2c \\ 2a & -2b & -2c & 2d \end{bmatrix}_k \right) \left(\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_k - \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_{k-1} \right)$$

We give $y = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_k$, $C'_{a_k} = \begin{bmatrix} 2c & 2d & 2a & 2b \\ -2b & -2a & 2d & 2c \\ 2a & -2b & -2c & 2d \end{bmatrix}_k$ and the rest of equation is considered the z_k and v_k, and we can sum up two matrices into R_k (this is the Extended Kalman Filter properties)

For magnetometer, we have the north magnetic field northward is [0 1 0] (note that this is not the true representation of the magnetic field, you can see appendix B for a proper explanation, the [010] vector is chosen for the sake of simplicity). We also have to normalize the magnetic field by $m_{norm} = m/||m||$

The measurement magnetometer model is processed like the accelerometer, but note that the data from magnetometer must be calibration like in appendix B to get the correct result The measurement equation is:

$$m_k^i = R_k(q_k)m + b_a + e_a$$

With the a_k^i is the inner frame accelerate of the sensor.
$C_m(q_k)$ is the quaternion rotation matrix at time k multiplied with the reference magneto vector.
b_a is the bias of the magnetometer.
e_a is the noise of the magnetometer.
m_k^i is the inertial magneto of the sensor.

Process the equation like the accelerometer, we achieve the measurement matrix:

$$C_{m_k} = \begin{pmatrix} 2(bc+ad) \\ a^2-b^2+c^2-d^2 \\ (cd-ab) \end{pmatrix}$$ The Jacobian matrix of C_{m_k} is:

$$C'_{m_k} = \begin{bmatrix} -2d & 2c & 2b & -2a \\ 2a & -2b & 2c & -2d \\ 2b & 2a & 2d & 2c \end{bmatrix}$$

So the measurement equation is: $H_k = \begin{pmatrix} C'_{a_k} & 0_{3x3} \\ C'_{m_k} & 0_{3x3} \end{pmatrix}$ The rest of the EKF is follow the EKF equation:

$$K_k = P_k^- H_k^T (H_k P_k - H_k^T + R_k)^{-1}$$
$$x_k = x_{k-1} + + K_k(y_k – H_k x_{k-1}^+)$$
$$Pk+ = (I – K_k H_k) P_k^- (I – K_k H_k)^T + K_k R_k K_k^T$$

The value of R_k and Q_k is chosen by getting the stationary value and covariance, the need of using z_k in the original Extended Kalman Filter equation is

not important. Design an Extended Kalman Filter is mostly designing its state equations and especially the covariance matrix R_k and Q_k is very important for the filter to converge, so sensor calibration is important part of the work, this is explained in the appendix B. The rest of the work now is mostly applying the proven EKF equation. Plotting the value of P matrix is the most helpful tool to make sure the filter work. In the Ardupilot EKF, a complementary filter is used to check as if the EKF is fail.

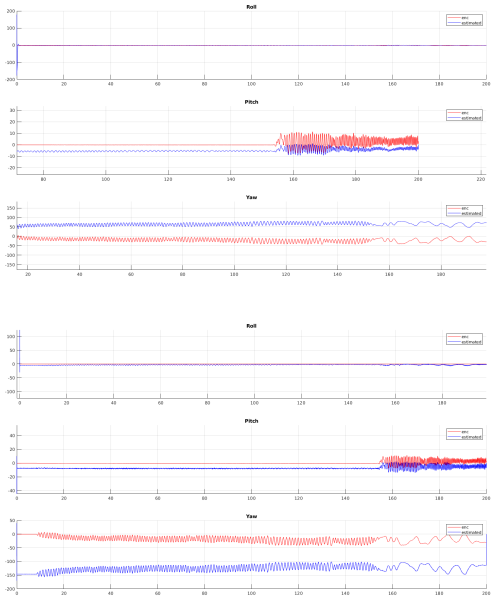

Fig 5: The result of the EKF with a real life data

We can see that the yaw and pitch estimation is wrong, but this can be fix because the estimation error is linear :3 this come from the wrongly magneto value has been chosen, but overall the filter work quiet good with difference about 1.5 - 2 degrees.

A Appendix A

Quaternion is inverted by Hamilton, it is the way to represent 3D orientation by using complex number. We all know about the famous equation: ij = jk = ki = -1, this denote the cross product, a useful tool to compute volume of a object, quaternion is actually an expansion from the cross product idea. [3] We call $q = a + bi + cj + dk$ is a quaternion, a,b,c,d is scalar numbers, i,j,k is complex numbers represent 3 axis of the 3D coordinates system

This quaternion can be discribe as $q = A + j * B$

Give quaternion p and q like this:

$$q = a + bi + cj + dk$$

$$p = a' + b'i + c'j + d'k$$

1. Quaternion adding:

$$p + q = (a + a') + (b + b')i + (c + c')j + (d + d')k$$

2. Quaternion multiphy

$$p * q = (a + bi + cj + dk) * (a' + b'i + c'j + d'k)$$

Remember $ij = jk = ki = -1$, we have the result:

$$\begin{aligned} p * q = aa' - bb' - cc' - dd' + (ab' + ba' + cd' - dc')i \\ +(ac' - bd' + ca' + db')j \\ +(ad' + bc' - cb' + da')k \end{aligned}$$

We consider the quaternion a 4x1 vector then we can split the result to get the multiphy matrix like is:

$$H_k = \begin{pmatrix} C'_{a_k} & 0_{3x3} \\ C'_{m_k} & 0_{3x3} \end{pmatrix}$$

3. Norm of the quaternion:

$$||q|| = \sqrt{(a^2 + b^2 + c^2 + d^2)}$$

4.Quaternion unit: Quaternion unit is the quaternion which has norm equal 1, unit quaternion represent the 3D coordinates system:

$$||q|| = \sqrt{(a^2 + b^2 + c^2 + d^2)} = 1$$

5. Inverse of a quaternion Give $q = a + bi + cj + dk$

$$q^{-1}q = 1$$

$$q^{-1} = (a - bi - cj - dk)/||q||^2$$

6. 3D rotation which quaternion: To represent an 3D orientation which quaterion, the best way to do so is using an unit quaternion, which is a quaternion which norm = 1 when we talk about rotation, always remember that we are talking about unit quaternion and the Kalman Filter on this tutorial has to perform normalization to return a uint quaternion.

To rotate a quaternion p with an quaternion q, we have:

$$p_{rot} = q * p * q^{-1}$$

[4] Proof of this equation will come in the next version of this document, or you can read more [5]. The simple explain is that the quaternion multiplication is

actually a rotation from 3D space into the 4D, and the multiplication with the inverse of that quaternion will rotate the quaternion again, from the 4D space to the 3D space.

6. Rotation model: Give a quaternion q, angular velocity ω, the rotation equation is:

$$\frac{dq}{dt} = -\frac{1}{2} * q * \omega * dt$$

Solve this derivative equation:

$$q(t) = e^{-\frac{1}{2}\omega\Delta t} * q_0$$

Using Taylor Expansion we have:

$$q = q + \frac{q\Delta t\omega}{2} + \frac{q\Delta t^2\omega}{8} + ...$$

Normally, we only need first or second order Taylor Expansion quaternion multiplication.

7.Quaternion rotation matrix: We already known that the 3D rotation with quaternion has the equation $p_{rot} = qpq^{-1}$ Then convert this operation into the matrix form as $p = R(q)p$ we have the quaternion matrix rotation:

$$R(q) = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{pmatrix}$$

8.Conversion from quaternion to euler angle: When compare the quaternion rotation matrix with the euler rotation matrix, we have the quaternion to euler angle conversion[6]:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(2(ab+cd), 1-2(b^2+c^2)) \\ asin(2(ac-db)) \\ atan2(2(ad+bc), 1-2(c^2+d^2)) \end{bmatrix}$$

The quaternion rotation group is actually the SU(2), mean the quaternion rotation double cover the SO(3) rotation group[7], which prevent Gimbal Lock happen and make the rotation return to start point when complete a circle rotation. The advantaged of quaternion is also in the computing performance of rotation:

Performance comparison of rotation chaining operations			
Method	Multiplies	Add/subtracts	Total operations
Rotation matrices	27	18	45
Quaternions	16	12	28

*Remember that the Gimbal Lock proof of quaternion is because of the SU(2) rotation group, if you decide to convert quaternion to Euler angle – for a readable output, the gimbal lock effect will come back.

B Appendix B

B.1 Gyroscope and its properties:

Gyroscope is a device that will retain its orientation no matter how the outer coordinate change. This ability come from the Coriolis force of the heavy/ */ make it retain the orientation, based on the inner orientation, we can calculate the orientation of the outer system. There are a lot of theory for this device, but it is not our concern now.

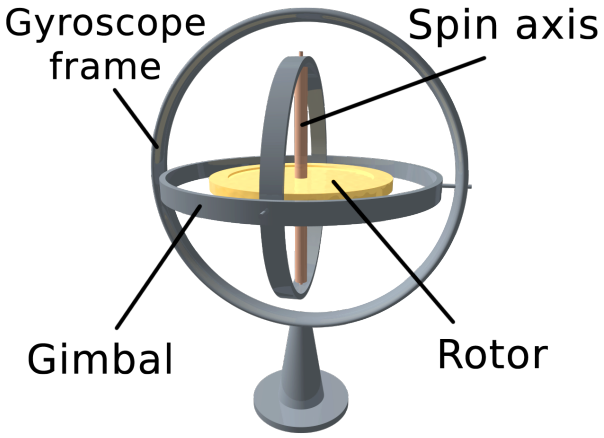

Fig 6: A gyroscope (Wikipedia)

The Gyroscope has a long history and nowadays they are used in many application of modern life, so to fit with a wide range of application like that, there also a wide range of gyroscope.

The most precise gyroscope is the navigation grade gyroscope, which is used in the ship INS, its has the precision of about 1.6 km drifting a day.

Next is the tactical grade, is used in the fighter plane or self driving car. And there are 3 rank upper until the commercial rank, with is our MPU-6050.

The commercial grade Gyroscope will return the angle velocity of the system instead of the absolute angle, this is because the MEMS technology behind this device. We can see that the raw data from low-cost commercial IMU always has the bias and always suffer from strong white Gaussian noise.

To calibrate the sensor, we should get the measurement of the sensor when its is non moving, by that we can measure the bias b_ω and the covariance σ on 3 axes of the sensor.

We already know about how to compute the orientation with the angle velocity by integral the sensor value: Euler:

$$\theta = \theta + (\omega - b_\omega) * \Delta t$$

Quaternion:

$$q' = \frac{1}{2} * q * (\omega - b_\omega)$$

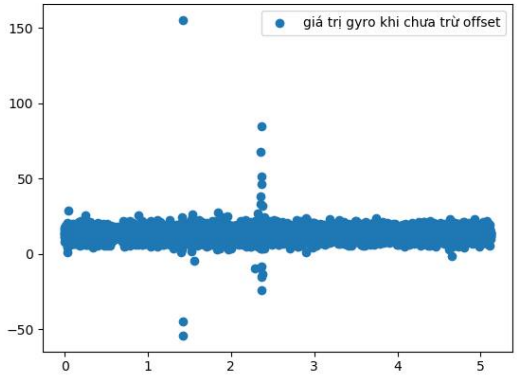

Fig 7: A gyroscope raw data, get from MPU-6050

This orientation is relative to a reference point, this is not the absolute orientation, so orientation data get from gyroscope always tend to be "drift".

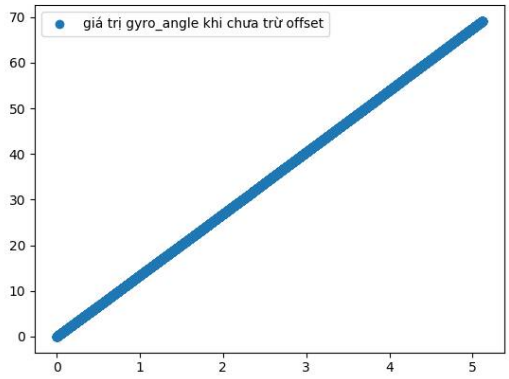

Fig 8: Gyroscope integral drifting, get from MPU-6050

As we discuss previously, the raw data has no drifting, the integration of sensor data makes the orientation data "drift". Eliminating the "drift" is impossible, even with our best navigation grade gyroscope. That is the reason gyroscope always goes with an absolute sensor like Accelerometer and Magnetometer.

B.2 Accelerometer:

An MEMS accelerometer is like the Gyroscope, the accelerometer suffer from sensor bias and the white Gaussian noise, but instead of getting the orientation

by integral the raw data, the accelerometer using the gravitation force of the Earth to measure the orientation of the sensor.

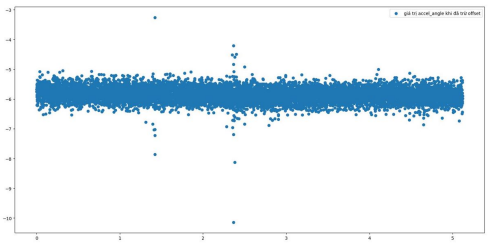

Fig 9: Accelerometer data, get from MPU-6050

We can only eliminate the bias but not the white Gaussian noise, that is why we need the Kalman Filter.

B.3 Magnetometer:

This is the most complex sensor to begin with, because its has two noise source: The hard iron, which is the magnetic field of the permenent nearby device (maybe the magnet of the speaker in an mobile phone, an iron casing,..etc

B.4 Earth magnetic field

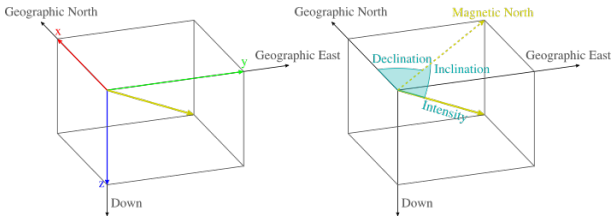

Fig 10: Earth magnetic field (Wikipedia)

At any location on Earth, the magnetic field can be locally represent by a 3 dimension factor: the magnetic vector (h_0). This vector can be decompose into 3 component: The Intensity vector (I) with is horizontally angled, the Declination angle, this is the angle of the North magnetic ole and North geography pole (D) and the Inclination vector with pointing up or down (L). Actually when we using the magnetometer, we have to look at the magnetic intensity map to know the D value to correct the measurement. [8]

The magnetometer has two noise source: Hard iron and Soft iron:

Hard iron will “slip” the magnetic coordinate by an linear amount, hard iron can be resolve by recenter the distribution of magnetic field to the root of the inner coordinates system

The soft iron: this is the “distortion” of the magnetic field. Normally, the magnetic field must has an nicely sphere shape which has the center is the root of coordinate system, but in case of soft iron, the magnetic field become an elliptic shape.

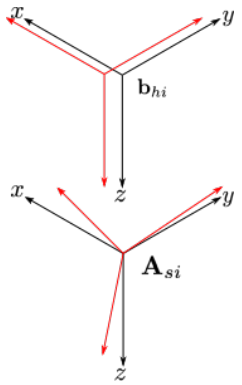

Fig 11: ”hard iron” and ”soft iron” effect.

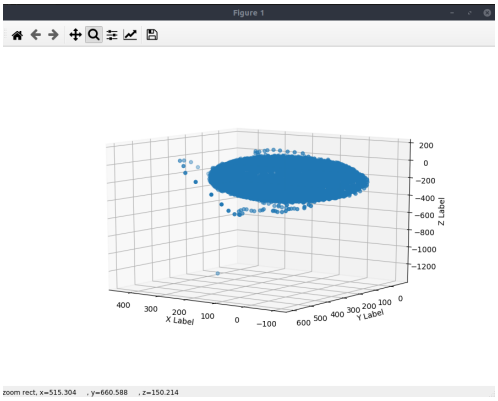

Fig 12: Magnetometer data suffered from both hard and soft iron, get from MPU-9250

The way to calibrate the magnetometer is based on [9] To solve hard iron, we calculate the offset by the equation for 3 axes:

$$offset_x = \frac{max(x) + min(x)}{2}$$

$$corrected_x = sensor - offset$$

$$offset_y = \frac{max(y) + min(y)}{2}$$

$$corrected_y = sensor - offset$$

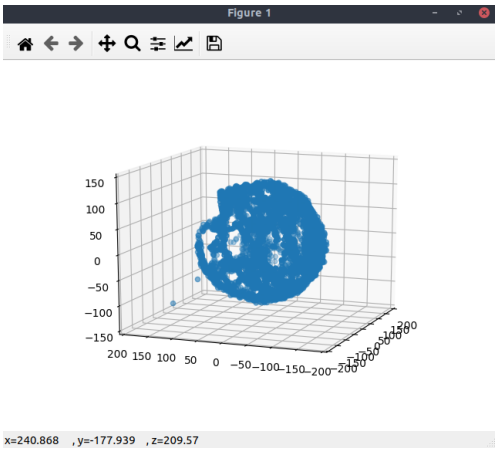

Fig 13: Magnetometer data calibrated.

$$offset_z = \frac{max(z) + min(z)}{2}$$
$$corrected_z = sensor - offset$$

This will turn the coordinates system to the (0,0,0) root.

To solve the soft iron, we using the following equation to turn the elliptic into the sphere:

$$\Delta_x = \frac{max(x) - min(x)}{2}$$
$$\Delta_y = \frac{max(y) - min(y)}{2}$$
$$\Delta_z = \frac{max(z) - min(z)}{2}$$
$$\Delta = \frac{\Delta_x + \Delta_y + \Delta_z}{3}$$

$$scale_x = \Delta/\Delta_x$$
$$scale_y = \Delta/\Delta_y$$
$$scale_z = \Delta/\Delta_z$$

$$corrected_x = (sensor_x - offset_x) * scale_x$$
$$corrected_y = (sensor_y - offset_y) * scale_y$$
$$corrected_z = (sensor_z - offset_z) * scale_z$$

This is the simple way to calibrate the magnetometer, the hard (and maybe better) way is using the elliptic equation fitting of the matlab. because the effect of hard and soft iron, the sensor data is elliptic, by finding the elliptic equation, we can find a transform equation for the data. This will be cover in the next version of the document.

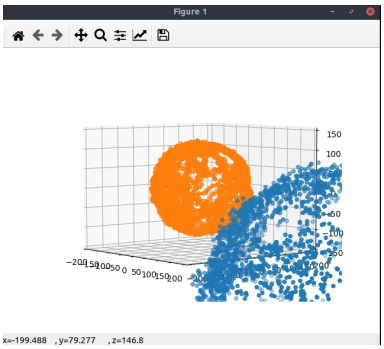

Fig 14: Magnetometer data after calibration, get from MPU-9250

References

[1] Dan Simon. *Optimal State Estimation*. John Wiley and Sons, Inc, 2006.

[2] Riki. Attitude determination with quaternion using ekf.

[3] Wikipedia. Quaternion.

[4] Wikipedia. Quaternions and spatial rotation.

[5] Joan Solà. Quaternion kinematics for the error-state kalman filter. *CoRR*, abs/1711.02508, 2017.

[6] Wikipedia. Conversion between quaternions and euler angles.

[7] Wikipedia. 3d rotation group.

[8] Testlabs. A way to calibrate a magnetometer.

[9] Mika Tuupola. How to calibrate a magnetometer?