Data Fusion with 9 Degrees of Freedom Inertial Measurement Unit To Determine Object's Orientation

By

Long Tran

Senior Project

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo

June 2017

Abstract

Data fusion is a technique to integrate different types of data to a single unit to provide a more reliable representation of tracking measurement. Today, data fusion can be found in many applications such as tracking and surveillance system as well as on mobile applications. In this project, 9 Degrees of Freedom (DOF) Inertial Measurement Unit (IMU) composed of 3-axis Magnetometer, 3-axis Accelerometer, and 3-axis Gyroscope are processed to yield the object's rotations in 3 dimensions. While the magnetometer (compass) is used to determine the heading angle, accelerometer is used to determine tilt axis, gyroscope can be process to calculate the angular velocity. Each sensor has its own advantages under different static or dynamic scenarios. By analyzing each sensor separately, the angle output computed from each sensor is then fused with angles from other sensors using filter algorithms. For data integration, Complimentary Filter and Extended Kalman Filter Algorithms are used in this project. To achieve a low cost solution to determine the object's orientation, the whole experiment is conducted using an Arduino Uno board integrating with ROHM- sensor shield for Arduino (SENSORSHLD1-EVK-101) that includes 9 degrees of freedom.

Table of Contents

Table of Figures

I. Introduction

Before doing this project, I always think why people need to find multiple solutions to solve a single problem, why not use those extra time to solve other problems rather than going around in the same circle. My mind has changed after this data fusion project. Understanding the advantages and disadvantages between different techniques, I fall in love with this project not only because it extends my technical knowledge, but also teaches me a new strategy to live. Data fusion has been used for many years in a lot of applications. It has been applied widely in robotic projects such as in dynamic control system and environment modelling using appropriate models and sensors [1], or to combine GPS and Inertial Navigation Systems for Pedestrians in Urban Area [2] . In the world today where details matter, data reliability has more and more impacts on many technologies. A rocket with the inaccuracy of a few mili-degree may not hit the target, a gunshot arming with a small inclined angles may injure allies instead of enemies. Therefore, combining data is a very challenging job to provide reliable, trustful measurements for experiments that require high accuracy. This complicated study used to involve many professional engineers and scientists, but in this paper I will help to deliver this fusion knowledge in such an easy and full of details explanation. Throughout this report, I will start with each sensor specification, and how to determine heading and tilt angles from three set of sensors. After that, I will walk you through fusion algorithms including Complimentary and Kalman Filter, and finally is my experiment and conclusion.

Experiencing the hardship when doing researches, I usually find it consume a lot of time to repeat the research paper experiment due to the lack of detailed instructions. People used to explain the project in a high level block without spending more time organize their works to give their peers a full detail guideline of how to build to project ground up. To avoid the struggles that

I have overcome, I will not stop at the technique to fuse data, I will also walk you through each step to rebuild the whole experiment interacting with Arduino C++ code, Matlab image representation, as well as the communication to those chip sensors. In the end, I hope you will enjoy my paper and withdraw some good information to expand this project to be bigger and maybe apply to more useful projects.

II. Objective

The purpose for this project is to design a low cost Inertial Measurement Unit via Sensor Fusion Algorithms in Arduino Environment to utilize sensor data to and optimize the solution of determining the object's orientation. Three types of sensors are used: Accelerometer, Gyroscope and Magnetometer to form a 9 Degrees of Freedom are fed to two different filter systems including Complimentary and Kalman Filter to reduce the sum of square error and be able to react on both dynamic and static system. Without any initial setup information, the system is able to determine the shape and angles in full 3D – 360 degrees on each axis.

III. Axis and rotation's definition

Before going further to details, let's define the reference for three sensors and their rotation axis. Based on the right hand rule, and its representation of axis and positive rotation definition:

The pointing direction of your index finger is in the direction of X.

Now curl your middle finger to point toward the Y's direction.

Lastly raise your thumb up, that is Z's direction

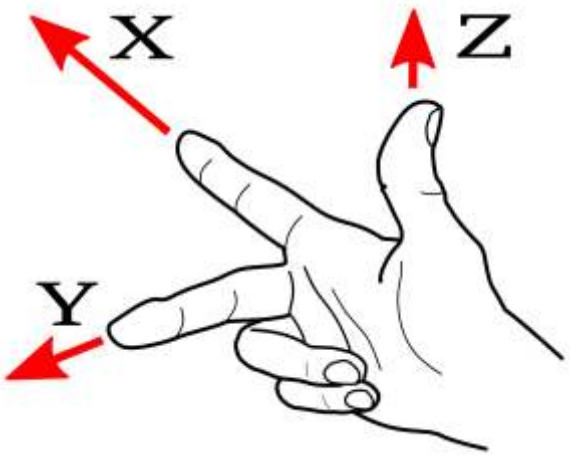

Figure 1: Right hand rule

Now, the positive rotation is defined as when the object rotates clockwise around each axis look from the origin toward the positive direction.

Rotate clockwise around x-axis, that is roll (+).

Rotate clockwise around y-axis, that is pitch (+).

Rotate clockwise around z-axis, that is yaw (+).

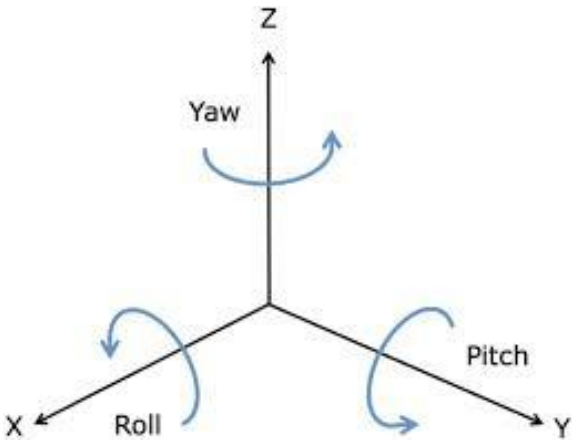

Figure 2: Positive Angle Definition

IV. Type of sensors and their features

Three type of sensors: Accelerometer, Gyroscope, and Magnetometer will be covered in this section. Each come with their advantages and disadvantages in the method of how to calculate the roll (θ)– pitch (ϕ) – yaw (ψ) angles using each sensor. For the simplicity of the project, I bought the ROHM- sensor shield for Arduino (SENSORSHLD1-EVK-101) that includes 9 degrees of freedom (DOF). The module consists of an accelerometer (KX122), a gyroscope (KXG03), a magnetometer (BM1422), an analog temperature sensor (BDE0600G), a digital barometric pressure sensor (BM1383GLV), a hall switch sensor (BU52014HFV), an analog UV sensor (ML8511), a digital color sensor (BH1745), an optical proximity sensors and ambient light sensor (RPR-0521). For the scope of this project, we only use the accelerometer, gyroscope and magnetometer sensor, all three sensors are mounted onto the sensor shield, so we don't have to worry too much about the alignment errors among the 9 axes. Note that, if you buy the three sensors separately, you need to aligns your sensors in such a way that each three axes (XYZ) of each sensor matches with other XYZ axis from other sensors. The specification of SENSORSHLD1-EVK-101 unit is showed in Table 1.

Table 1: SENSORSHLD1-EVK-101 specification

Power Requirement	
Input Voltage	3.3 – 5 VDC
Power	1.2 W max
Measurement	
Accelerometer (KX122)	
Range	± 2g, ± 4g, ± 8g
Resolution	16 bits (65536)
Non – Linearity	0.6% of FS
Noise	0.75 mg RMS at 50 Hz
Output Data Rate	Typical 50 Hz
Gyroscope (KXG03)	
Range	± 256, ± 512, ± 1024, ± 2048 deg/s
Resolution	16 bits (65536)
Non - Linearity	0.5% of FS
Noise Density	0.03 deg/sec/√Hz
Output Data Rate	Up to 51200 Hz
Magnetometer (BM1422)	
Range	± 1200 µT
Resolution	16 bits (65536)
Output Data Rate	100 Hz

1. *Accelerometer*

According to [3], the accelerometer measures all forces that apply to the object. We could think of a ball is floating inside a cubic box. When there is no force applied, the ball will be floating in the center of the box.

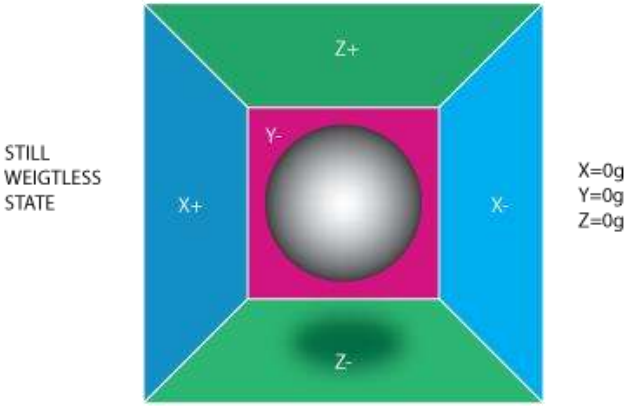

Figure 3: Accelerometer ax = ay = az = 0g

In the above picture, I also assign the sign to each of the wall in 3 X-Y-Z directions. If we place this box standstill on Earth, the only force that applied to the box now is the gravitational force. We know force is equal to mass multiply with acceleration or F = ma.

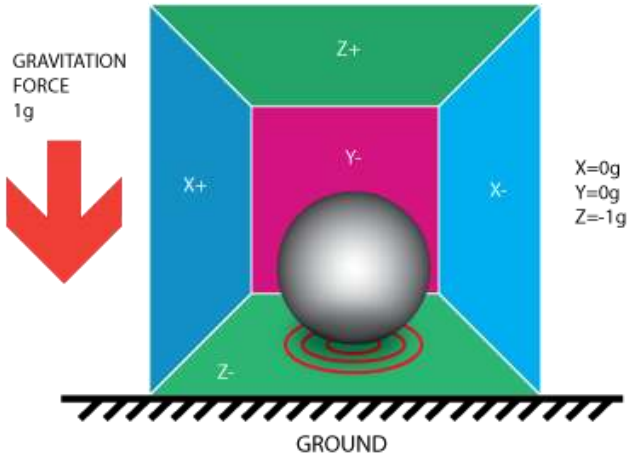

Figure 4: Accelerometer ax = ay = 0g, az = -1g

This example shows the box get -1g (1 g = 9.81 m/s^2) on the Z direction. Similarly, it will get 1g for each of the direction it faces to the ground.

So far, we have analyzed the force measured by the accelerometer on a single axis. In fact, this sensor can measure on 3 degrees of freedom. Let's take another example where the X and Z corners are facing down to the ground, and the angle between x and z is 45^o degree.

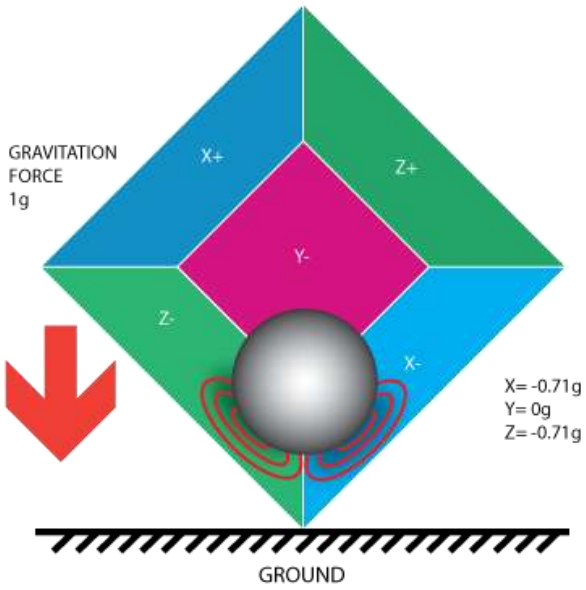

Figure 5: ax = az = -0.71g, ay = 0g

The total acceleration for a standstill object on Earth is 1g, or $a_x^2 + a_y^2 + a_z^2 = 1$ (g). X and Z is 45 degrees between each other. ➔ $a_x = a_z$ and $a_{y.} = 0$ ➔$2\,a_x^2 = 1$ ➔ $a_x = a_z = 0.71g$ which is equivalent Pythagoreans theory [4].

According to the document AN3461 from Freescale Semiconductor [5]: with the three sets of rotation matrices

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{pmatrix} \quad R_y(\phi) = \begin{pmatrix} cos\phi & 0 & -sin\phi \\ 0 & 1 & 0 \\ sin\phi & 0 & cos\phi \end{pmatrix} \quad R_z(\psi) = \begin{pmatrix} \cos\psi & sin\,\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Solving the $R_{xyz} = R_xR_yR_z$ for the pitch and roll angles, we get

$\tan(\theta) = \frac{a_y}{a_z}$ $\tan(\phi) = -\frac{a_x}{\sqrt{(a_y^2+a_z^2)}}$

or in Arduino C code:

```
float roll = (float) (atan2(KX122_Accel_Y_OUT, KX122_Accel_Z_OUT)) * RAD_TO_DEG;
float pitch = (float) (atan2(-KX122_Accel_X_OUT, sqrt(KX122_Accel_Y_OUT* KX122_Accel_Y_OUT + KX122_Accel_Z_OUT*KX122_Accel_Z_OUT)))  * RAD_TO_DEG;
```

Using the accelerometer alone could not help you to determine the yaw angle. This is due to the fact that when the z axis is facing down to the ground, no matter what yaw angle you rotate around the z-axis, it always measures $a_z = 1g$ and $a_x = a_y = 0$. Therefore, we need another sensor to calculate the yaw angle.

Accelerometer is very accurate in long term, and the roll and pitch angles can be calculated precisely when the object is standstill on Earth. However, when the sensor is moving, the moving acceleration will impact the rotation calculation. However, if we apply a low pass filter to the output of the accelerometer, most of the noise and the external force acceleration will be removed, the filtered value is then used to integrate with other sensor which is covered in section [V].

2. *Magnetometer*

Magnetometer (compass) is an instrument that measure the magnetic field's strength in a particular position. People have used compass for thousands years to determine the heading (yaw) angle. We know that the Earth is like a giant magnet with the two North and South poles on two sides when the Earth spin around itself. When magnetometer is on a flat surface, with no other electro-magnetic field surrounding, the yaw angle is calculated as:

$$\psi = \tan^{-1}\left(\frac{m_y}{m_x}\right)$$

```
float yaw = atan2(My,Mx) * RAD_TO_DEG;
```

Note that, this formula is incorrected when the sensor is tilted. I will cover the tilt compensation later on in the fusion algorithm part [V].

When rotate the magnetometer clockwise, the yaw angle should increase, and decreases for the opposite direction. If you find the opposite, this can be done by changing the sign of your y-direction value. This happens due to the magnetometer is facing upside down.

Figure 6: Earth's magnetic field

3. *Gyroscope*

Gyroscope detects angular velocity based on the microelectromechanical systems (MEMS). A small resonating mass attached to the MEMS is shifted as the angular velocity changes. Follow Newton's law of conservation of momentum, the output movement is converted to a small electronic signal, which is then amplified and read by a controller [6]. In short, each gyroscope channel measure angular velocity (rotation speed) around one of the axis. For this project, we use a full 3 DOF to capture the changes in all three X-Y-Z directions.

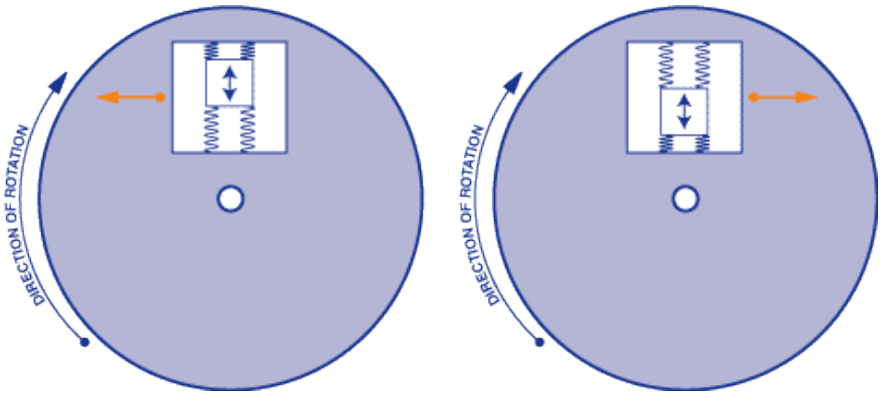

Figure 7: Internal operational view of a MEMS gyroscope sensor

To go from angular velocity to the rotation angle, the Math can be easily known as:

Angle = angular velocity * time
Or: $\theta = \omega_\theta * \Delta t$ $\phi = \omega_\phi * \Delta t$ $\psi = \omega_\psi * \Delta t$
Where ω_θ, ω_ϕ, and ω_ψ are angular velocity of roll (θ), pitch (ϕ), and yaw (ψ) angle respectively

The only problem with the gyroscope is its calculation drift overtime because of the integration. A small error measuring at one time will carry on by the integral. In addition to that, due to the inertia, the gyro – rates won't come back to zero when object is standstill. For example, if the raw data is measured at an ω value, your object was moved and then put back to the standstill state, the value measured will be different from ω. However, gyroscope is very accurate in short term. Therefore, people usually apply a high pass filter to the gyroscope measurement after they obtain the angular rate from the sensor [7].

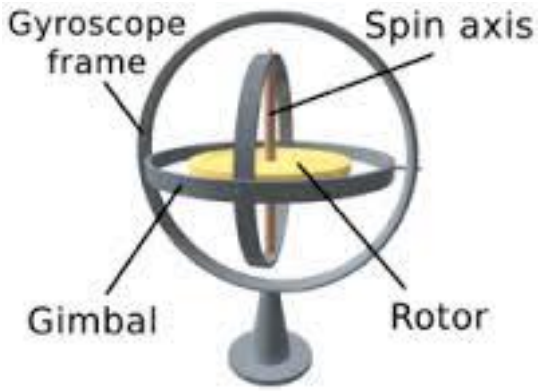

Figure 8: Gyroscope sensor

V. Fusion method

If you read up to this point, you probably already have a basic knowledge of how to calculate the three roll, pitch, and yaw angle from the accelerometer, gyroscope, and magnetometer.

The first step is to combine the accelerometer and magnetometer to calculate the tilt compensation for the yaw (heading) angle. According to Jay A. Farrell [8], we will use the roll and pitch values from the calculation of accelerometer in the formula to calculate the heading with compensation.

$$M_x = m_x \cdot \cos(\phi) + m_z \cdot \sin(\phi)$$

$$M_y = m_x \cdot \sin(\theta) \cdot \sin(\phi) + m_y \cdot \cos(\theta) - m_z \cdot \sin(\theta) \cdot \cos(\phi)$$

$$\psi = \tan^{-1}\frac{M_y}{M_x}$$

```
float Mx = mx * cos(pitchA) + mz * sin(pitchA);
float My = mx * sin(rollA) * sin(pitchA) + my * cos(rollA) - mz * sin(rollA) * cos(pitchA);
float yaw = atan2(-My,Mx) * RAD_TO_DEG;
```

After that, the three angles will be integrated with the three independent angular calculation from the gyroscope to form a more accurate result. Below is the whole procedure block diagram:

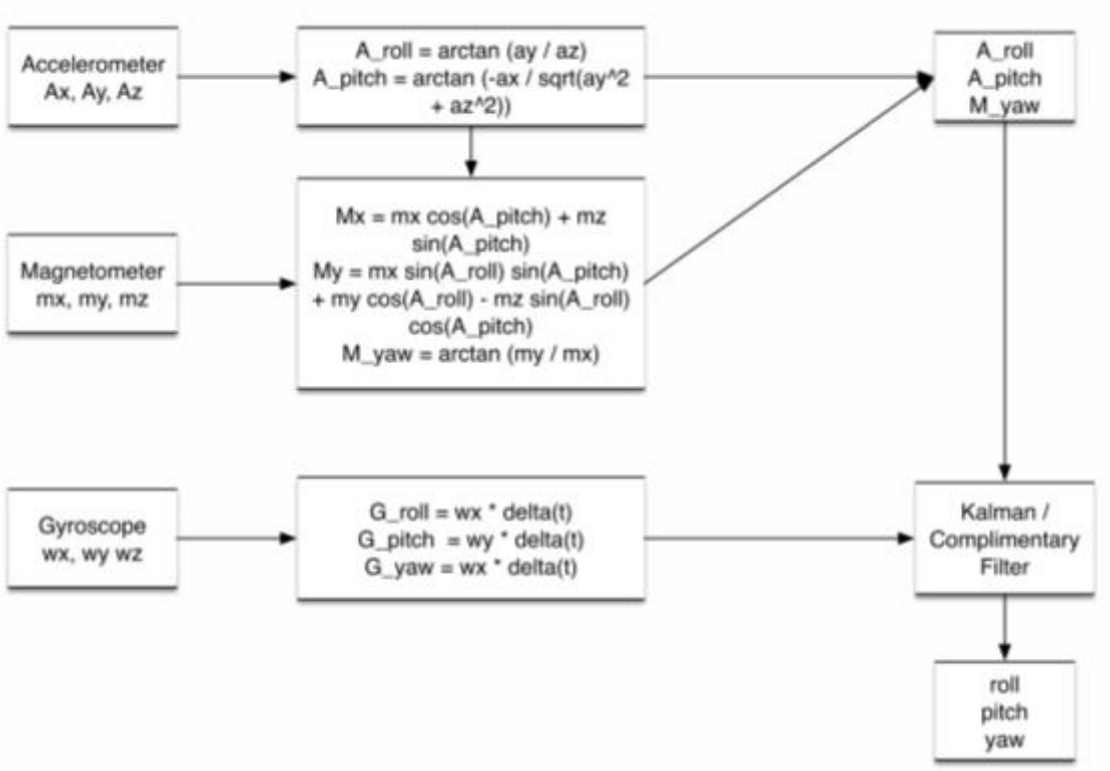

Figure 9: Procedure's block diagram

Everything will be the same except for the filter part. In this project, I will use Complimentary filter and Kalman filter to optimize the output angle calculated from each individual sensor.

1. *Complimentary Filter*

The name of this filter probably explain for its calculation. It basically takes advantage of each sensor and compensate for the disadvantages the other sensors have. In this case, as I already explained above, accelerometer performs the best with low frequency while gyroscope performs the best with high frequency. Complimentary filter is simple and easy to use, it

contains a fixed value for low pass and high pass's weights. In short term, we use data from the gyroscope, and in long term, we use data from accelerometer. The filter has the following form:

$$CF\_angle = HP\_weight * (CF\_angle + gyro\_data * \Delta t) + LP\_weight * acc\_data$$

Where the two constant HP_weight (high-pass weight) and LP_weight (low-pass weight) have to add up to one. From my experiment, HP_weight value should be very close to one, and LP_weight should be very close to zero. Several experiments are performed with different set of high pass and low pass weights - Appendix B, I visually look at the graph generated from each measurement and compare with accelerometer and gyroscope output in slow motion and high motion respectively. As a result, Complimentary Filter works the best when HP_weight = 0.98 and LP_weight = 0.02. The expression becomes:

$$CF\_angle = 0.98 * (CF\_angle + gyro\_data * \Delta t) + 0.02 * acc\_data$$

The expression of course is put in an infinite loop with a fix Δt time constant for simplicity. Every cycle, the new filtered value is equal 98% of the previous filter-angle value adding with the angle calculated by gyroscope data, the whole thing adds with 2% of angle calculated from the accelerometer/magnetometer data. Note that, different weights are used and shown in Error! Reference source not found..

Because the two low pass and high pass weights are fixed at the beginning, overtime, this complimentary filter will result in a more drift result, and yield a small inaccuracy in the measurement. We need a better filter; it is called Kalman filter.

2. *Kalman filter*

Unlike the complimentary filter where it uses a fixed number for low pass and high pass weight. Kalman filter is an algorithm that takes into account a series of measurements overtime,

in this case, they are the measurement from all three sensors used in this experiment, with the model noise and the measurement noise, the filter is able to dynamically calculate and update the weight – or Kalman gain value. This system is heavy in math, but provide a more accurate result than those based on single measurement alone.

Kalman filter consist two main steps: measurement update, and time update

In this project, the three roll, pitch and yaw angle are independent; therefore, each angle could be tracked separately as following:

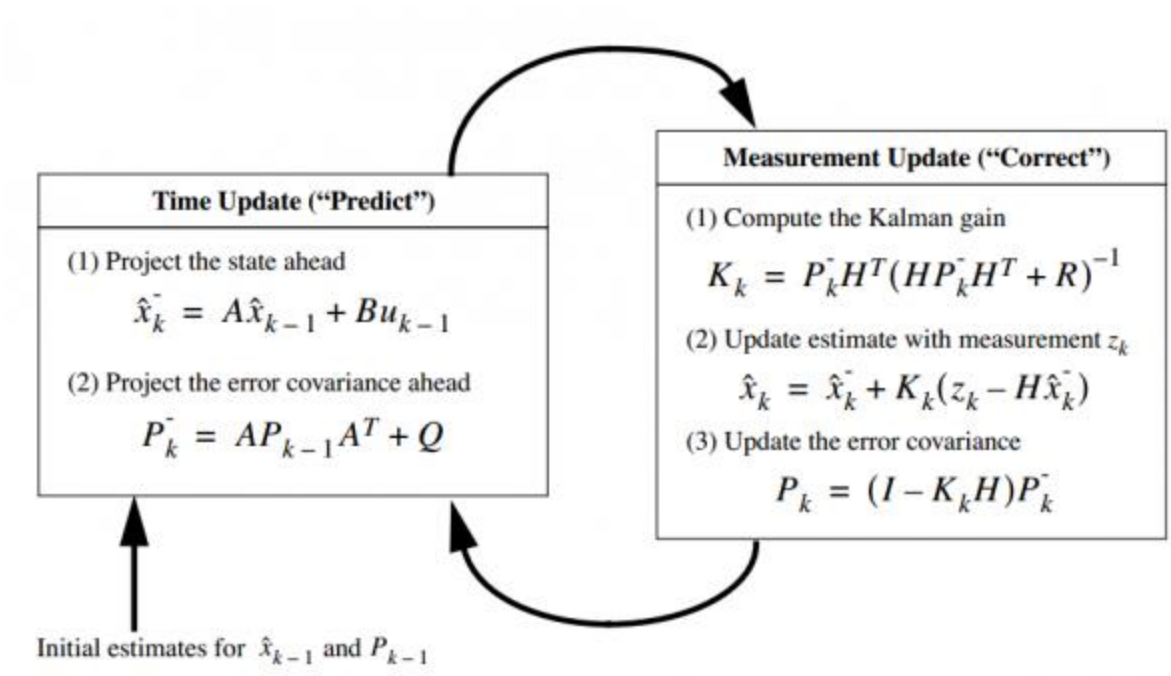

Figure 10: Extended Kalman Filter Process

According to Kristian Sloth Lauszus [9], let's define:

$$x_k = \begin{bmatrix} \theta \\ \theta_b' \end{bmatrix}, \quad A = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \quad u_{k-1} = \theta', \quad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q'_{\theta b} \end{bmatrix} \Delta t, \quad H = [1 \quad 0], \quad P = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}, \text{ and } \quad K = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

where x_k is the state variable consisting two elements: angle and bias angular velocity

A is the state transition model which is applied to the previous state $X_{k|k-1}$

B is the control input model which is applied to the control vector u_{k-1}

H is the observation model which maps the true state space into the observed space

Q is the covariance of the process noise

R is the covariance of the observation noise

P is the estimate covariance matrix

K is the Kalman gain

Time Update (Predict) step

Step 1: $X_{k|k-1} = A\, x_{k-1|k-1} + B\theta'$

$$\begin{bmatrix} \theta \\ \theta_b' \end{bmatrix}_{k \mid k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \theta_b' \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \theta'$$

$$\begin{bmatrix} \theta \\ \theta_b' \end{bmatrix}_{k|k-1} = \begin{bmatrix} \theta + \Delta t\, (\theta' - \theta_b') \\ \theta_b' \end{bmatrix}_{k-1|k-1}$$

As you can see, the new angle θ is equal to the previous old angle plus the difference of angle rate multiply by Δt

```
rate = newRate - bias;

angle += dt * rate;
```

Step 2: $P_{k|k-1} = A\, P_{k-1|k-1}\, A^T + Q_k$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q'_{\theta b} \end{bmatrix} \Delta t$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} = \begin{bmatrix} P_{00} + \Delta t(\Delta t P_{11} - P_{01} - P_{10} + Q_\theta) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q'_{\theta b}\Delta t \end{bmatrix}$$

The equation can be translated to the Arduino code as following:

```
P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);

P[0][1] -= dt * P[1][1];

P[1][0] -= dt * P[1][1];

P[1][1] += Q_bias * dt;
```

Measurement Update (Predict) step

Step 1:

Define $S = H P_{k|k-1} H^T + R$

$$S = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + R = P_{00\,k|k-1} + R$$

```
S = P[0][0] + R_measure;
```

$K_k = P_{k|k-1} H^T S^{-1}$

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} * \frac{1}{S} = \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{S}$$

```
K[0] = P[0][0] / S;

K[1] = P[1][0] / S;
```

Step 2:

Define $y_k = z_k - H\, x_{k|k-1}$

$$y_k = z_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \theta_b \end{bmatrix}_{k|k-1} = z_k - \theta_{k|k-1}$$

$$x_{k|k} = x_{k|k-1} + K_k\, y_k$$

$$\begin{bmatrix} \theta \\ \theta_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \theta_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0\, y \\ K_1\, y \end{bmatrix}_k$$

```
y = newAngle - angle

angle += K[0] * y;

bias += K[1] * y;
```

Step 3:

$P_{k|k-1} = (I - K_k\, H)\, P_{k|k-1}$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{00} & K_1 P_{01} \end{bmatrix}$$

```
P[0][0] -= K[0] * P[0][0];

P[0][1] -= K[0] * P[0][1];

P[1][0] -= K[1] * P[0][0];
```

```
P[1][1] -= K[1] * P[0][1];
```

The covariance matrix of process noise (Q) in this case is the accelerometer noise and the estimated bias rate. This information is found from the manual of accelerometer and gyroscope then round up to the close mili-unit. In addition, the covariance of measurement noise (R) is referred to the variance of the measurement, if it is too high, the filter will react slowly as the new measurement has more uncertainty, on the other hands, if R is too small, the output result becomes noisy as we trust the new measurement of accelerometer too much. Based on the experiment, the following values are picked for the process and the measurement noise

```
Q_angle = 0.001;

Q_bias = 0.003;

R_measure = 0.03;
```

VI. Data and Analysis

To run the experiment with the 9 Degrees of Freedom chips integrated with Arduino, we need to make sure so stay away at least two feet from electronic devices, magnets, or irons to eliminate the electromagnetic field affecting the magnetometer. First, let the chip standstill then move it around for a few seconds then keep repeating combination of static and dynamic process until having around one minute of data. It is very important to let the chip standstill between dynamic movements because we will use static moments as our good reference since the external force is equal to zero and the accelerometer data can be used to calculate the accurate angles. All output angles from raw data and two filters are logged and plotted .vs time in the three directions to compare the result. Figure 11 and shows the roll angles of the system using raw data and both Complimentary and Kalman Filtered data where the blue curve represents the angle calculated from the accelerometer, the orange curve represents the angle calculated from the gyroscope, and the red and black curves are the result from Kalman filter and Complimentary filter respectively. The IMU was keeping static in the first 4 seconds, from 10s to 14s, from 15s to 17s, from 18s to 20.5s, from 21s to 23s, from 24s to 25s, and was moving otherwise.

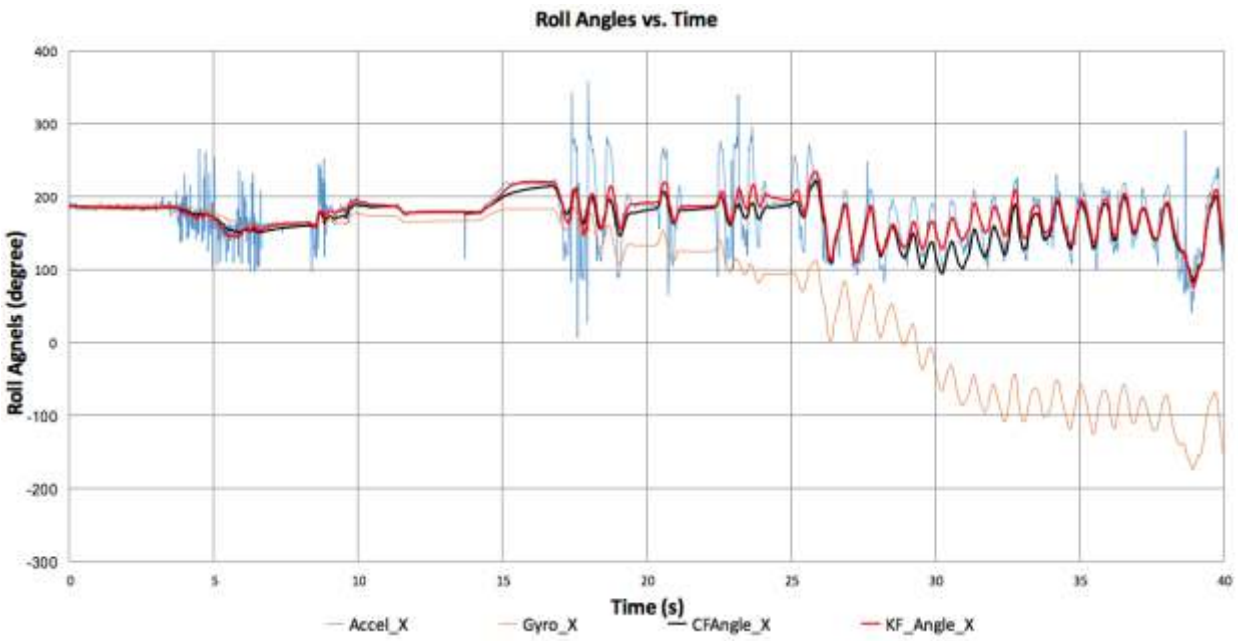

Figure 11: Roll Angles vs. Time

The two problems with gyroscope and accelerometer can easily be observed on this graph. The blue curve (accelerometer) is quite noisy – with a lot of oscillations when the sensor is moving, but accurate in long term – or when it stays flat. The gyroscope is more accurate in short term, but has a drift problem in long term. As you can see, the orange curve diverges and keeps going down instead of going back to the same level as before.

By fusing the two accelerometer and gyroscope sensors using the Complimentary Filter (black), and the Kalman Filter (red), these two black and red curves show the two filters reduce the oscillation of the acceleration noises, and follow the shape of the gyroscope angles in highly dynamic system. Both filters do a pretty good job, but Kalman filter surely has more impacts on the result. As I explain in IV-1, angles calculated from accelerometer sensor can be used as a reference when the object is not moving; for example, in the first 4 seconds, and from 10s to 14s, both filters are in line with the accelerometer angle. At 14s, the IMU was moving for one second, and holding static for the next two seconds, from what being shown in Figure 11, only the

Kalman Filter angle align on top of the accelerometer angle, the Complimentary Filtered angle takes two seconds to reach the reference. The same things happen from 18s to 20.5s, from 21s to 23s.

Let's look at the zoom-in version in Figure 12

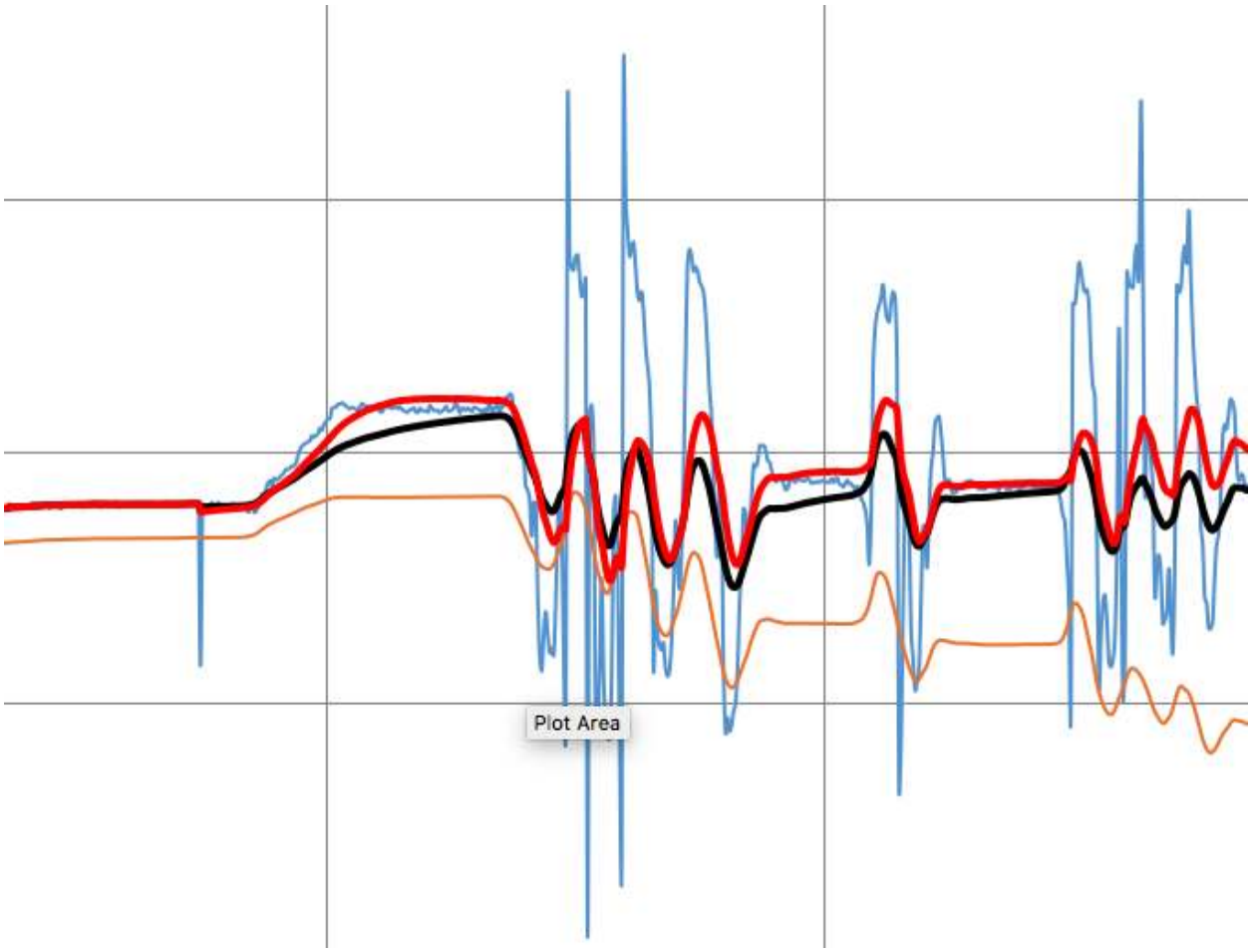

Figure 12: Filtered data (roll angles) - zoom in version

The pitch and yaw angles are plotted in Figure 15 and Figure 16.

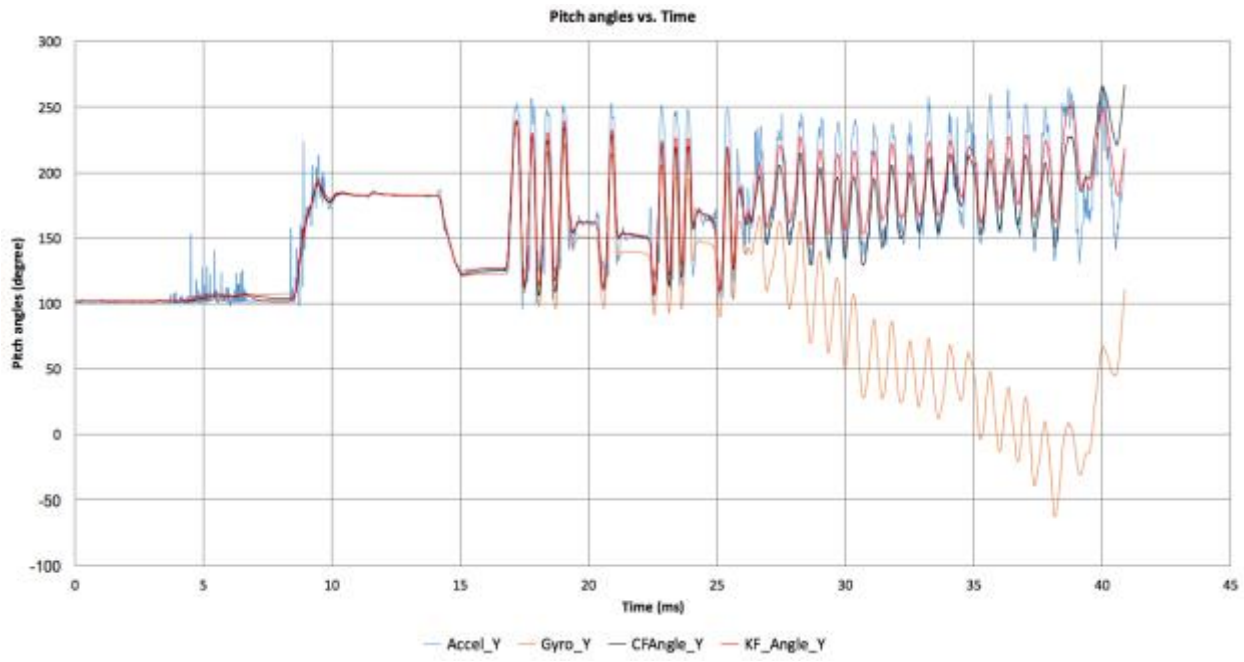

Figure 13: Pitch Angles vs. Time

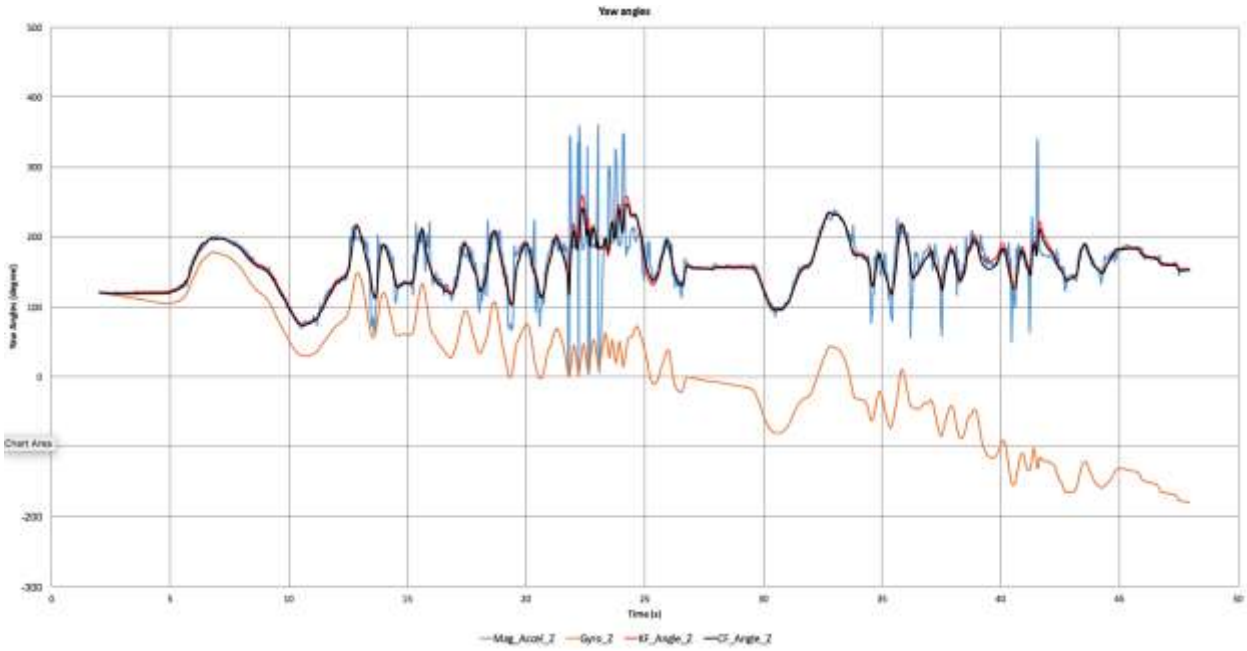

Figure 14: Yaw Angles vs. Time

Reading the graph can help us analyze the results, but does not help to observe the data in real time. In fact, I also wrote a small program in Matlab that communicate with the Arduino board and grab data and display with an airplane model in every 100ms. Figure 15 and Figure 16 show some examples when the IMU board tilt at certain angles on three turning directions.

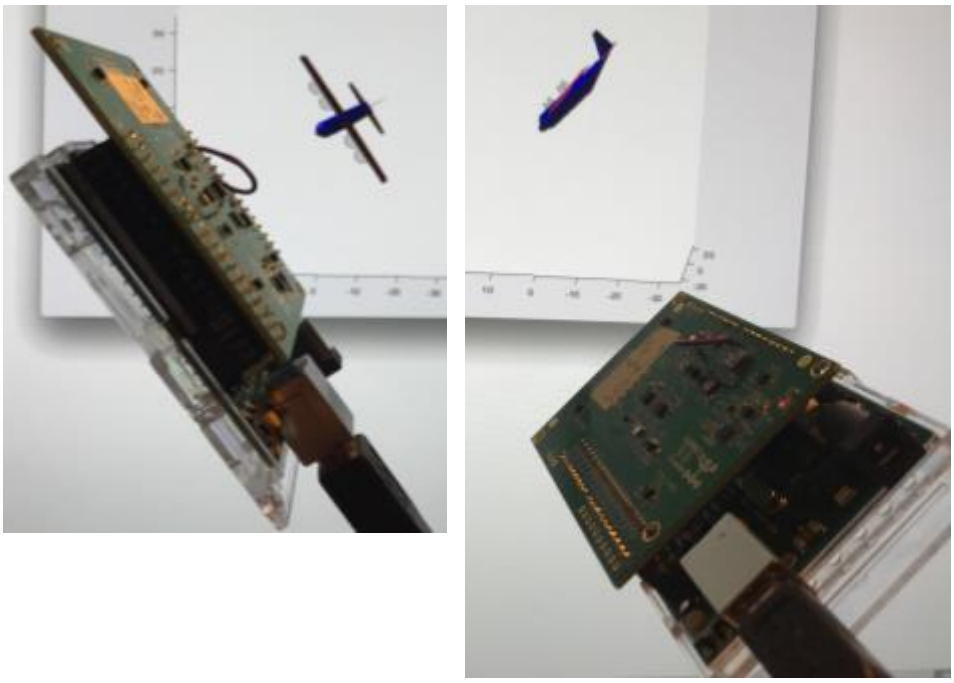

Figure 15: Display angles with airplane model in real time with Matlab (1)

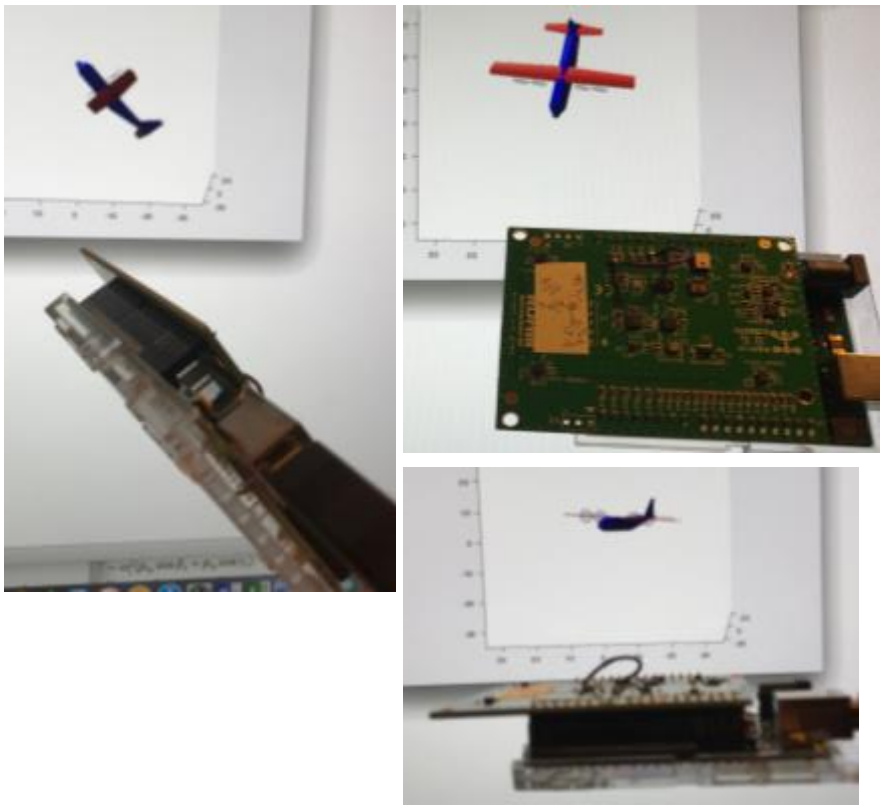

Figure 16: Display angles with airplane model in real time with Matlab (2)

In all examples above, the airplane always matches the pose of the Inertial Measurement Unit board. Of course, there is a maximum delay of 100 ms due to the slow update rate of Matlab application (10 Hz). However, if the board is rotated slowly, one can observe no difference in the pose between the airplane model and the Arduino board. This can clearly show that our experiment is successful. You can check out my demo at: https://youtu.be/Qqu_J5i-fMk

VII. Conclusion

To summarize, data fusion is used in every day of life to help to achieve the best accurate measurement. This project is a good way to show how collaboration will affect the overall result. In the experiment that we involve throughout the report, the gyroscope or accelerator or magnetometer themselves can determine the rotation angles. However, by combining the full

IMU with three sensors together, we can actually get a more promising result. I also use the two different type of sensors to differentiate the accuracy between the twos. The complimentary use a constant weight for both low pass and high pass filter and does not react to the change of the overall system. On the other hand, the Kalman filter takes into account the historical/statistical value to analyze for the current state; therefore, it can react faster and better. This implies a good strategy for those stubborn people who never want to change, they will end up similar to the complimentary filter, easy and simple, but not accurate enough. If you already read the introduction, you now can guess why I fall in love with this project and what strategies I withdraw.

VIII. Future Works

1. *Compensate misalignment error among 9 DOF axes*

   The project assumes perfect alignment among 9 degrees of freedom axes. However, in reality, there is always a small misalignment that decreases the accuracy of the output angles. My future work includes finding a way to calibrate the chip to compensate for the misalignment error.

2. *Implement different filter models*

   Complimentary and Extended Kalman filter are well known filters that has been used for several decades. Besides that, there are also other filter such as fuzzy adaptive federated Kalman Filter (FAFKF) or Extended Information Filter (EIF) that may yield a more accurate results.

Reference

[1] B. Siciliano, "Multisensor Data Fusion," *Springer Handbook of Robotics,* pp. 19-20, 2008.

[2] K. B. &. J. Demkowicz, "Data Integration from GPS and Inertial Navigation Systems for Pedestrians in Urban Area," *TransNav,* vol. 7, 2013.

[3] G. Gangster, "Accelerometer & Gyro Tutorial," [Online]. Available: http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/. [Accessed 2017].

[4] T. P. Site, "The Pythagorean Theorem," [Online]. Available: https://www.theproblemsite.com/reference/mathematics/geometry/the-pythagorean-theorem/. [Accessed 2017].

[5] MarkPedley, "Tilt Sensing Using a Three-Axis Accelerometer," vol. AN3461, pp. 5-10, 03 2013.

[6] Ronzo, "How a Gyro Works," [Online]. Available: https://learn.sparkfun.com/tutorials/gyroscope/how-a-gyro-works. [Accessed 2017].

[7] J. Esfandyari, "Sensor Technology," [Online]. Available: http://www.mouser.com/applications/sensor_solutions_mems/. [Accessed 2017].

[8] J. A. Farrell, Aided Navigation, McGraw Hill, 2008, pp. 356-357.

[9] K. S. Lauszus, "A practical approach to Kalman filter and how to implement it," 10 9 2012. [Online]. Available: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/. [Accessed 2017].

Appendices

Appendix A: Cost Breakdown

Product	Desc	Quantity	Price
	DEV-11224 Arduino Uno	1	$24.95
	SENSORSHLD1-EVK-101 SHIELD BOARD MULTI-SENSOR	1	$108.75
	RTL – 10423 ROHS USB Cable – A to B – 6 foot retail	1	$4.95

Appendix B: Complimentary with different weights

As explained in IV-1 above, the accelerometer data is not reliable when the IMU is dynamically changing. I use the same data as shown in Figure 11, and generate several pairs of high pass and low pass filter weights. Figure 17 and Figure 18 shows the roll angle calculated from accelerometer (blue), gyroscope (orange), Kalman Filter (red), and Complimentary Filter with high pass weights: 98% (black), 90% (brown), 80% (yellow), 70% (green), and 50% (violet) respectively. From the experiment, the lower the high pass weight, the more it follows the accelerometer angle. However, for every ripple – or when the IMU is moving, we don't want the Complimentary Filter follows the blue curve too much. Therefore, 98% for high pass is a reasonable value for this project.

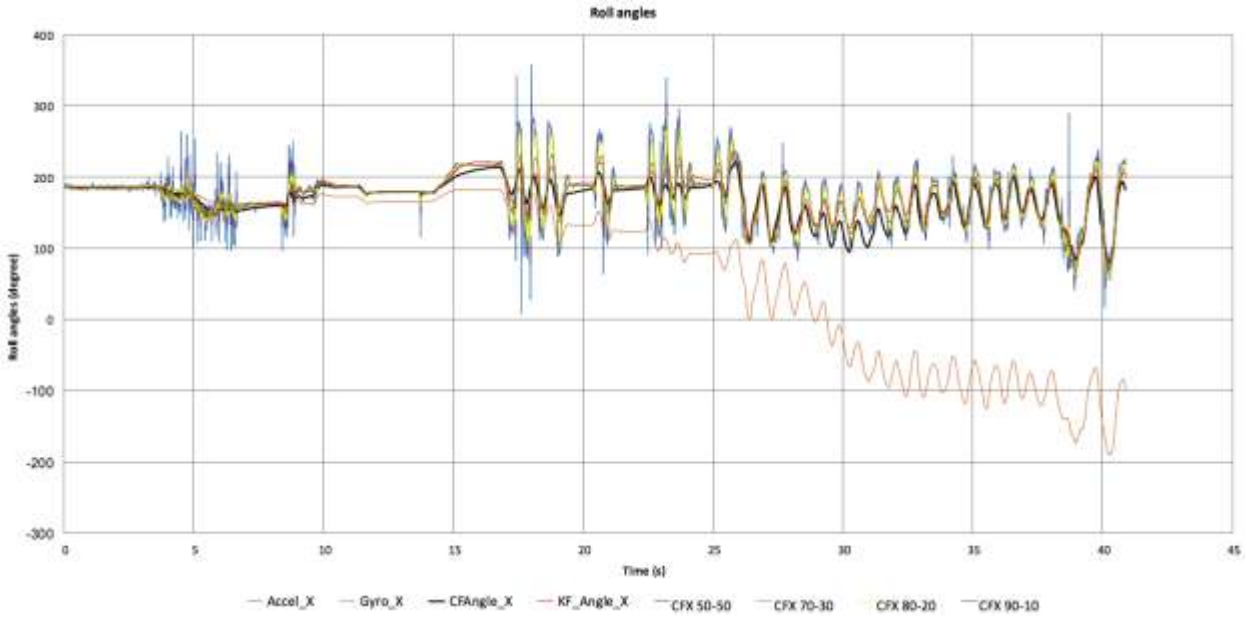

Figure 17: Experiment with multiple set of Complimentary Filters for roll angle

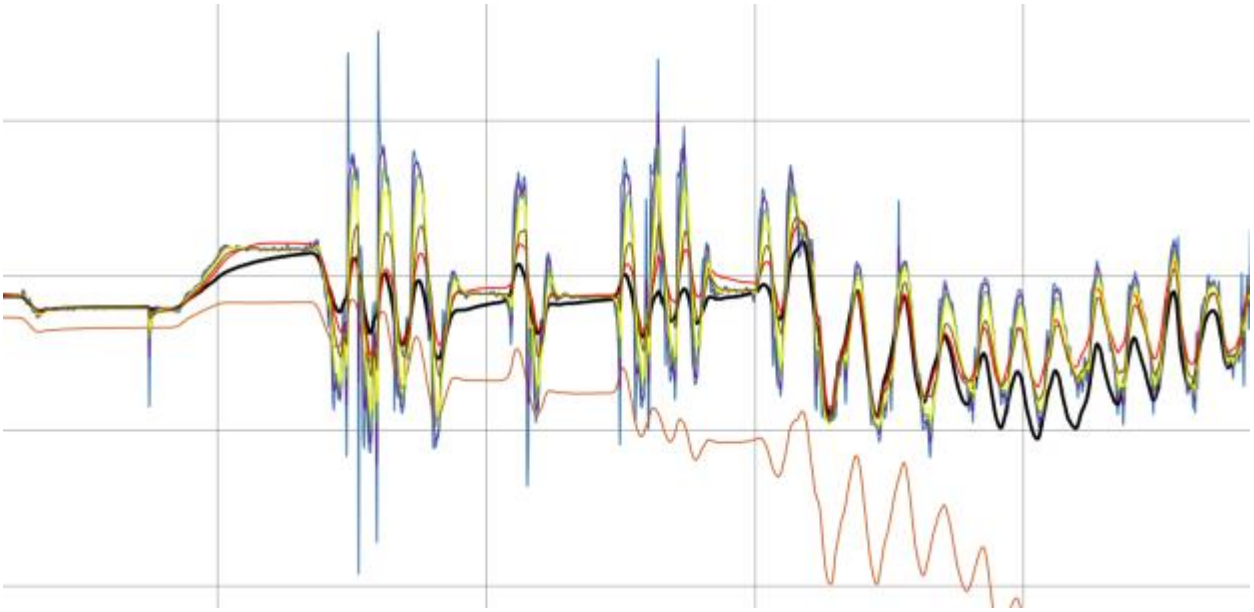

Figure 18: Experiment with multiple set of Complimentary Filters for roll angle (zoom-in)

Appendix C: Interfacing hardware

The ROHM- sensor shield (SENSORSHLD1-EVK-101) is designed to work with Arduino, so we only need to attach both boards together and then hook up the USB A-B cable to the computer as follow:

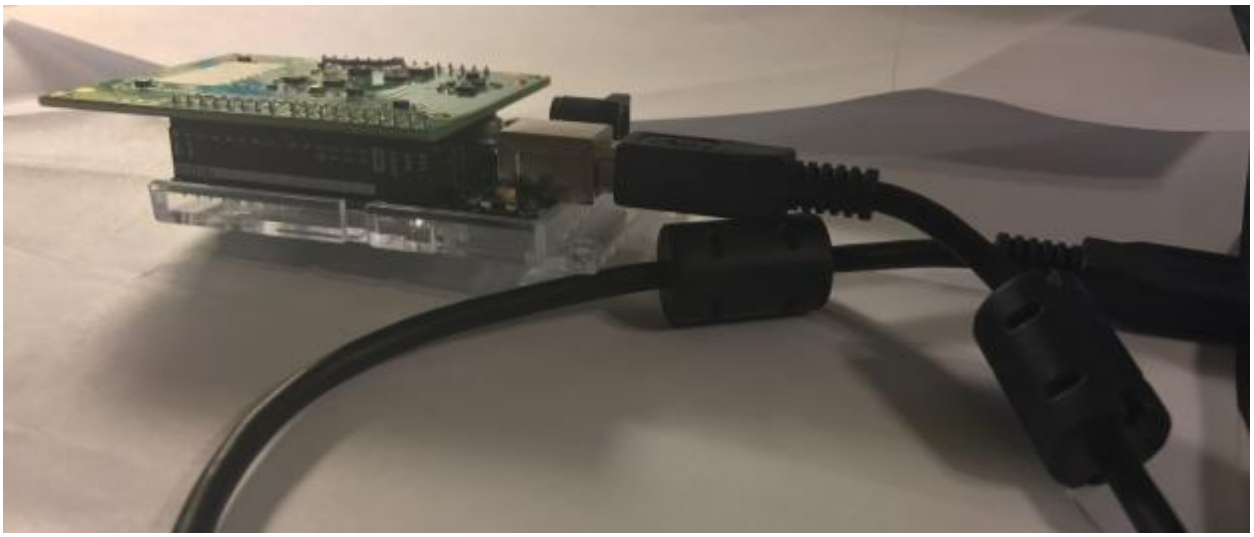

Figure 19: Interfacing SENSORSHLD1-EVK-101 with Arduino

We will use i2C interface to obtain data from the accelerometer (KX122), the gyroscope (KXG03), and the magnetometer (BM1422). Please refer to the datasheet for more information.

Appendix D: Code

```
#include <stdio.h>
#include "Kalman.h"
// ----- Debugging Definitions -----

#define KX122       //KX122
#define KXG03       //KXG03
#define MagField    //BM1422

// ----- Demo Mode Definitions -----
//#define CSVOutput
#define SensorSamplePeriod  500 //in ms

// ----- Included Files -----
//#include <Wire.h>         //Default I2C Library

#define SCL_PIN 5  //A5       //Note that if you are using the I2C based sensors, you will need to download and
#define SCL_PORT PORTC    //install the "SoftI2CMaster" as "Wire" does not support repeated start...
#define SDA_PIN 4  //A4         //References:
#define SDA_PORT PORTC    //  http://playground.arduino.cc/Main/SoftwareI2CLibrary

#include "SoftI2CMaster.h"  //  https://github.com/felias-fogg/SoftI2CMaster
#define I2C_TIMEOUT 1000  // Sets Clock Stretching up to 1sec
#define I2C_FASTMODE 1    // Sets 400kHz operating speed

// ----- Globals -----
```

```
int sensorValue = 0;
float sensorConvert = 0;
unsigned int j;
unsigned int lineClear = 0;
double intervalTime = 0;

//Digital Input Globals - Hall Sensor
#ifdef HallSen
int Hall_Out0 = 0;
int Hall_Out1 = 1;
#endif

//I2C globals (using SoftI2CMaster libary) - MEMs Kionix Sensor
int I2C_check = 0;



int matlabData = -1;
#define ACCEL 1
#define MAGN  2
#define GYRO  3



#ifdef KX122
int KX122_DeviceAddress = 0x3C;  //this is the 8bit address, 7bit address = 0x1E
int KX122_Accel_X_LB = 0;
int KX122_Accel_X_HB = 0;
int KX122_Accel_Y_LB = 0;
int KX122_Accel_Y_HB = 0;
int KX122_Accel_Z_LB = 0;
int KX122_Accel_Z_HB = 0;
```

```
int KX122_Accel_X_RawOUT = 0;
int KX122_Accel_Y_RawOUT = 0;
int KX122_Accel_Z_RawOUT = 0;
float KX122_Accel_X_OUT = 0;
float KX122_Accel_Y_OUT = 0;
float KX122_Accel_Z_OUT = 0;
#endif



#ifdef KXG03
int         i = 11;
int         t = 1;
short int   aveX = 0;
short int   aveX2 = 0;
short int   aveX3 = 0;
short int   aveY = 0;
short int   aveY2 = 0;
short int   aveY3 = 0;
short int   aveZ = 0;
short int   aveZ2 = 0;
short int   aveZ3 = 0;
int KXG03_DeviceAddress = 0x9E;  //this is the 8bit address, 7bit address = 0x4F
int KXG03_Gyro_X_LB = 0;
int KXG03_Gyro_X_HB = 0;
int KXG03_Gyro_Y_LB = 0;
int KXG03_Gyro_Y_HB = 0;
int KXG03_Gyro_Z_LB = 0;
int KXG03_Gyro_Z_HB = 0;
float KXG03_Gyro_X = 0;
float KXG03_Gyro_Y = 0;
```

```
float KXG03_Gyro_Z = 0;
short int KXG03_Gyro_X_RawOUT = 0;
short int KXG03_Gyro_Y_RawOUT = 0;
short int KXG03_Gyro_Z_RawOUT = 0;
short int KXG03_Gyro_X_RawOUT2 = 0;
short int KXG03_Gyro_Y_RawOUT2 = 0;
short int KXG03_Gyro_Z_RawOUT2 = 0;
int KXG03_Accel_X_LB = 0;
int KXG03_Accel_X_HB = 0;
int KXG03_Accel_Y_LB = 0;
int KXG03_Accel_Y_HB = 0;
int KXG03_Accel_Z_LB = 0;
int KXG03_Accel_Z_HB = 0;
float KXG03_Accel_X = 0;
float KXG03_Accel_Y = 0;
float KXG03_Accel_Z = 0;
short int KXG03_Accel_X_RawOUT = 0;
short int KXG03_Accel_Y_RawOUT = 0;
short int KXG03_Accel_Z_RawOUT = 0;
#endif

#ifdef MagField
unsigned int BM1422_DeviceAddress = 0x1E;
short int BM1422_Mag_X_LB = 0;
short int BM1422_Mag_X_HB = 0;
short int BM1422_Mag_Y_LB = 0;
short int BM1422_Mag_Y_HB = 0;
short int BM1422_Mag_Z_LB = 0;
short int BM1422_Mag_Z_HB = 0;
short int BM1422_Mag_X_RawOUT = 0;
```

```
short int BM1422_Mag_Y_RawOUT = 0;
short int BM1422_Mag_Z_RawOUT = 0;
float BM1422_Mag_X = 0;
float BM1422_Mag_Y = 0;
float BM1422_Mag_Z = 0;
#endif



#define RAD_TO_DEG 57.29578
#define M_PI 3.14159265358979323846
Kalman KFilter[3];
float KFAngle[3];
float CFAngle[3];
#define RawDataOutput 1

float mag_off[3];
unsigned long tStart, tStop, tElapse;
unsigned long eTime;
float yaw_gyro;
void setup()
{

  //Wire.begin();        // start I2C functionality
  Serial.begin(115200);  // start serial port at 9600 bps
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  I2C_check = i2c_init();
```

```
  if(I2C_check == false){
    while(1){
      Serial.write("I2C Init Failed (SDA or SCL may not be pulled up!");
      Serial.write(0x0A); //Print Line Feed
      Serial.write(0x0D); //Print Carrage Return
      delay(500);
    }
  }

  pinMode(13, OUTPUT);   //Setup for the LED on Board

  pinMode(2, INPUT_PULLUP);
  pinMode(3, INPUT_PULLUP);
  //pinMode(13, INPUT_PULLUP);

  //----- Start Initialization for KX122 Accel Sensor -----
#ifdef KX122
  //1. CNTL1 (0x18) loaded with 0x40 (Set high resolution bit to 1)
  //2. CNTL1 (0x18) loaded with 0xC0 (Enable bit on)

  i2c_start(KX122_DeviceAddress);  //This needs the 8 bit address (7bit Device Address + RW
bit... Read = 1, Write = 0)
  i2c_write(0x18);
  i2c_write(0x40);
  i2c_stop();

  i2c_start(KX122_DeviceAddress);  //This needs the 8 bit address (7bit Device Address + RW
bit... Read = 1, Write = 0)
  i2c_write(0x18);
  i2c_write(0xC0);
```

```
  i2c_stop();

#endif
  //----- END Initialization for KX122 Accel Sensor -----

  //----- Start Initialization for KXG03 Gyro Sensor -----
#ifdef KXG03
  //1. STBY REG (0x43) loaded with 0xEF

  i2c_start(KXG03_DeviceAddress);  //This needs the 8 bit address (7bit Device Address +
RW bit... Read = 1, Write = 0)
  i2c_write(0x43);
  i2c_write(0x00);
  i2c_stop();

#endif
  //----- END Initialization for KXG03 Gyro Sensor -----

#ifdef MagField
  i2c_start(BM1422_DeviceAddress);  //This needs the 8 bit address (7bit Device Address +
RW bit... Read = 1, Write = 0)
  i2c_write(0x1B);
  i2c_write(0xC0);
  i2c_stop();

  i2c_start(BM1422_DeviceAddress);  //This needs the 8 bit address (7bit Device Address +
RW bit... Read = 1, Write = 0)
  i2c_write(0x5C);
  i2c_write(0x00);
  i2c_stop();
```

```
  i2c_start(BM1422_DeviceAddress);  //This needs the 8 bit address (7bit Device Address +
RW bit... Read = 1, Write = 0)
  i2c_write(0x5D);
  i2c_write(0x00);
  i2c_stop();

  i2c_start(BM1422_DeviceAddress);  //This needs the 8 bit address (7bit Device Address +
RW bit... Read = 1, Write = 0)
  i2c_write(0x1D);
  i2c_write(0x40);
  i2c_stop();
#endif



    mag_off[0] = -28.5;
    mag_off[1] = -22.73;
    mag_off[2] = -21.73;

    delay(2000);
    GetAccelData ();
    GetMagnData ();
    GetGyroData ();

    KFilter[0].setAngle((float) (atan2(KX122_Accel_Y_OUT, KX122_Accel_Z_OUT) + PI) *
RAD_TO_DEG);
    KFilter[1].setAngle((float) (atan2(-KX122_Accel_X_OUT, sqrt(KX122_Accel_Y_OUT*
KX122_Accel_Y_OUT + KX122_Accel_Z_OUT*KX122_Accel_Z_OUT)))  * RAD_TO_DEG);

    float norm = Norm(KX122_Accel_X_OUT, KX122_Accel_Y_OUT, KX122_Accel_Z_OUT);
```

```
  float pitchA = -asin(KX122_Accel_X_OUT / norm);// * RAD_TO_DEG;
  float rollA = asin(KX122_Accel_Y_OUT / cos(pitchA) / norm);// * RAD_TO_DEG;

    float magX = BM1422_Mag_X - mag_off[0];
    float magY = BM1422_Mag_Y - mag_off[1];
    float magZ = BM1422_Mag_Z - mag_off[2];

    norm = sqrt(magX*magX + magY*magY + magZ*magZ);

    float mx = magX/ norm;
    float my = -1 * magY/ norm;
    float mz = magZ/ norm;

    float Mx = mx * cos(pitchA) + mz * sin(pitchA);
    float My = mx * sin(rollA) * sin(pitchA) + my * cos(rollA) - mz * sin(rollA) * cos(pitchA);
    float yaw = atan2(-My,Mx) * RAD_TO_DEG;

  if (yaw > 360) {
      yaw -= 360;
  }
  else if (yaw < 0) {
      yaw += 360;
  }
  KFilter[2].setAngle(yaw);
  yaw_gyro = yaw;
  CFAngle[2] = yaw;

  tStart = millis();
}
```

```
void loop()
{
   GetAccelData ();
   GetMagnData ();
   GetGyroData ();
   float roll = (float) (atan2(KX122_Accel_Y_OUT, KX122_Accel_Z_OUT)) * RAD_TO_DEG;
   float pitch = (float) (atan2(-KX122_Accel_X_OUT, sqrt(KX122_Accel_Y_OUT*
KX122_Accel_Y_OUT + KX122_Accel_Z_OUT*KX122_Accel_Z_OUT)))  * RAD_TO_DEG;

   float norm = Norm(KX122_Accel_X_OUT, KX122_Accel_Y_OUT, KX122_Accel_Z_OUT);
   float pitchA = -asin(KX122_Accel_X_OUT / norm);// * RAD_TO_DEG;
   float rollA = asin(KX122_Accel_Y_OUT / cos(pitchA) / norm);// * RAD_TO_DEG;

     float magX = BM1422_Mag_X - mag_off[0];
     float magY = BM1422_Mag_Y - mag_off[1];
     float magZ = BM1422_Mag_Z - mag_off[2];

     norm = sqrt(magX*magX + magY*magY + magZ*magZ);

     float mx = magX/ norm;
     float my = -1 * magY/ norm;
     float mz = magZ/ norm;

     float Mx = mx * cos(pitchA) + mz * sin(pitchA);
     float My = mx * sin(rollA) * sin(pitchA) + my * cos(rollA) - mz * sin(rollA) * cos(pitchA);
     float yaw = atan2(-My,Mx) * RAD_TO_DEG;

   if (yaw > 360) {
      yaw -= 360;
   }
```

```
  else if (yaw < 0) {
     yaw += 360;
  }
  tStop = millis();
  tElapse = tStop - tStart;
  tStart = millis();
  float temp = tElapse / 1000.0;
  KFAngle[0] = KFilter[0].getAngle(roll, KXG03_Gyro_X, temp);
  KFAngle[1] = KFilter[1].getAngle(pitch, KXG03_Gyro_Y, temp);

  KFAngle[2] = KFilter[2].getAngle(yaw, KXG03_Gyro_Z, temp);

  float gyro_newangle = KXG03_Gyro_Z * temp;
  yaw_gyro += gyro_newangle;
  CFAngle[2] = 0.98 * (CFAngle[2] + gyro_newangle) + 0.02 * yaw;



  char t;
 if (Serial.available () > 0)
 {
    while(Serial.available() > 0) {
       t= Serial.read();
    }
    Serial.flush();
    Serial.print(KFAngle[0], 4);
    Serial.println();

    Serial.print(KFAngle[1], 4);
    Serial.println();
```

```
    Serial.print(KFAngle[2], 4);
    Serial.println();
  }

}


void GetAccelData () {
   //----- START Code for Reading KX122 Accel Sensor -----
#ifdef KX122
  i2c_start(KX122_DeviceAddress);
  i2c_write(0x06);
  i2c_rep_start(KX122_DeviceAddress | 1);  // Or-ed with "1" for read bit
  KX122_Accel_X_LB = i2c_read(false);
  KX122_Accel_X_HB = i2c_read(false);
  KX122_Accel_Y_LB = i2c_read(false);
  KX122_Accel_Y_HB = i2c_read(false);
  KX122_Accel_Z_LB = i2c_read(false);
  KX122_Accel_Z_HB = i2c_read(true);
  i2c_stop();

  KX122_Accel_X_RawOUT = (KX122_Accel_X_HB<<8) | (KX122_Accel_X_LB);
  KX122_Accel_Y_RawOUT = (KX122_Accel_Y_HB<<8) | (KX122_Accel_Y_LB);
  KX122_Accel_Z_RawOUT = (KX122_Accel_Z_HB<<8) | (KX122_Accel_Z_LB);

  KX122_Accel_X_OUT = (float)KX122_Accel_X_RawOUT / 16384;
  KX122_Accel_Y_OUT = (float) -1 * KX122_Accel_Y_RawOUT / 16384;
  KX122_Accel_Z_OUT = (float) KX122_Accel_Z_RawOUT / 16384;
#endif
```

```
#ifdef RawDataOutput1
Serial.write("KX122 (X) = ");
Serial.print(KX122_Accel_X_OUT);
Serial.write(" g");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("KX122 (Y) = ");
Serial.print(KX122_Accel_Y_OUT);
Serial.write(" g");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("KX122 (Z) = ");
Serial.print(KX122_Accel_Z_OUT);
Serial.write(" g");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
#endif

  //----- END Code for Reading KX122 Accel Sensor -----
}

void GetGyroData () {
  //----- START Code for Reading KXG03 Gyro Sensor -----
#ifdef KXG03

  i2c_start(KXG03_DeviceAddress);
  i2c_write(0x02);
  i2c_rep_start(KXG03_DeviceAddress | 1);
  KXG03_Gyro_X_LB = i2c_read(false);
```

```
  KXG03_Gyro_X_HB = i2c_read(false);
  KXG03_Gyro_Y_LB = i2c_read(false);
  KXG03_Gyro_Y_HB = i2c_read(false);
  KXG03_Gyro_Z_LB = i2c_read(false);
  KXG03_Gyro_Z_HB = i2c_read(true);
  i2c_stop();

  KXG03_Gyro_X_RawOUT = (KXG03_Gyro_X_HB<<8) | (KXG03_Gyro_X_LB);
  KXG03_Gyro_Y_RawOUT = (KXG03_Gyro_Y_HB<<8) | (KXG03_Gyro_Y_LB);
  KXG03_Gyro_Z_RawOUT = (KXG03_Gyro_Z_HB<<8) | (KXG03_Gyro_Z_LB);

  //Scale Data
  KXG03_Gyro_X = (float) -1 * (KXG03_Gyro_X_RawOUT * 0.007813 + 0.000004);
  KXG03_Gyro_Y = (float)KXG03_Gyro_Y_RawOUT * 0.007813 + 0.000004;
  KXG03_Gyro_Z = (float) -1 * (KXG03_Gyro_Z_RawOUT * 0.007813 + 0.000004);
#endif
  //----- END Code for Reading KXG03 Gyro Sensor -----

#ifdef RawDataOutput1
Serial.write("KXG03 Gyro (X) = ");
Serial.print(KXG03_Gyro_X);
Serial.write(" deg/sec");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("KXG03 Gyro (Y) = ");
Serial.print(KXG03_Gyro_Y);
Serial.write(" deg/sec");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("KXG03 Gyro (Z) = ");
```

```
Serial.print(KXG03_Gyro_Z);
Serial.write(" deg/sec");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
#endif
}

void GetMagnData () {

#ifdef MagField
  i2c_start(BM1422_DeviceAddress);
  i2c_write(0x10);
  i2c_rep_start(BM1422_DeviceAddress | 1);
  BM1422_Mag_X_LB = i2c_read(false);
  BM1422_Mag_X_HB = i2c_read(false);
  BM1422_Mag_Y_LB = i2c_read(false);
  BM1422_Mag_Y_HB = i2c_read(false);
  BM1422_Mag_Z_LB = i2c_read(false);
  BM1422_Mag_Z_HB = i2c_read(true);
  i2c_stop();

  BM1422_Mag_X_RawOUT = (BM1422_Mag_X_HB<<8) | (BM1422_Mag_X_LB);
  BM1422_Mag_Y_RawOUT = (BM1422_Mag_Y_HB<<8) | (BM1422_Mag_Y_LB);
  BM1422_Mag_Z_RawOUT = (BM1422_Mag_Z_HB<<8) | (BM1422_Mag_Z_LB);

  BM1422_Mag_X = BM1422_Mag_X_RawOUT * 0.042;
  BM1422_Mag_Y = BM1422_Mag_Y_RawOUT * 0.042;
  BM1422_Mag_Z = BM1422_Mag_Z_RawOUT * 0.042;
#endif
```

```
#ifdef RawDataOutput1
Serial.write("BM1422 Mag (X) = ");
Serial.print(BM1422_Mag_X);
Serial.write("uT");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("BM1422 Mag (Y) = ");
Serial.print(BM1422_Mag_Y);
Serial.write("uT");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
Serial.write("BM1422 Mag (Z) = ");
Serial.print(BM1422_Mag_Z);
Serial.write("uT");
Serial.write(0x0A);  //Print Line Feed
Serial.write(0x0D);  //Print Carrage Return
#endif

}

float Norm(float a, float b, float c) {
  return sqrt(a*a + b*b + c*c);
}
```

```
%Matlab
arduino = serial('/dev/cu.usbmodem1411','BaudRate', 115200)
set(arduino,'Timeout',1000);
```

```
fopen(arduino)
M = double(zeros(1,3));
eTime = uint64(0);
pause(10);

while (1)
    tic
    fprintf(arduino, 1);

    M(1) = fscanf(arduino, '%lf');
    M(2) = fscanf(arduino, '%lf');
    M(3) = fscanf(arduino, '%lf');
    M
    flushinput(arduino);
    figure (1)
    c130('color','blue',...
        'wing','red',...
        'tailwing','red',...
        'lines','none', ...
        'roll', M(1), ...
        'pitch', M(2),...
        'yaw', -M(3))
    view([-95 -5 20])
    axis ([-25 25 -35 35 -35 35])

    pause(0.05);
    toc
end

fclose(arduino);
```

```
delete(arduino);
clear ;
delete(instrfindall);
```