SENSOR FUSION USING KALMAN FILTER IN MATLAB

V ADARSH (21307R001)
PRADEEP CHUDASAMA (213070111)

1 Introduction

In this experiment sensor fusion was achieved in MATLAB using Kalman filter. An inertial measurement unit(IMU) was used, which was connected to Arduino for interfacing with PC. IMU's accelerometer and gyroscope readings were fused to get an accurate reading of pitch and roll angles to estimate orientation of IMU. The pitch action is a rotation of IMU about it's y-axis and roll action is rotation of IMU about it's x-axis.

2 Kalman Filter

Kalman Filter is an optimal state estimation technique. We are interested in knowing the internal states of a system, but in most practical cases states are not directly measurable , and only the output that comes out of the system can be measured directly by some sensors. One can develop the state space model of a system and find equations according to which state evolves but it requires that we know the EXACT value of initial state of a system because if initial state is known then the states at all future time can be accurately determined using the system model. But in reality, one cannot accurately determine initial state of a system which in turn means the states we calculate using system model are also not accurate and in fact are only an estimate of true state. To ensure these estimates are accurate one can examine how the output is affected by states and incorporate this effect in state estimation equation and correct the estimates. For this the system needs to be observable. This is what an observer does. It estimates the state using system dynamics and output. Consider a discrete system :

$x_k = Ax_{k-1} + Bu_k$

$z_k = Cx_k$

An observer equation looks like:

$\hat{x}_k = A\hat{x}_{k-1} + Bu_k + K(z_k - \hat{z}_k) \qquad (1)$

Here $\hat{x}$ is the state estimate. Observer estimates the state by using the system and model and incorporation of the effect of states on system output (the 3rd term in eq(1)). Kalman filter is a commonly used observer that estimates the optimal value of 'K' in equation (1).

So, for kalman filter we need to develop a system model according to which state evolves. Using these equations we make a "prediction" of state at time state 'k'. Next we measure the system output z_k and calculate the estimate of output $\hat{xz_k}$ using the predicted state. The term $z_k - \hat{z}_k$ is refereed to as Innovation in kalman filter literature. This innovation gives an idea of how accurate our state prediction is. Then we update the state estimate by incorporating this innovation using Kalman Gain 'K'.

Kalman filter estimates 'K' my minimizing the error covariance matrix P. P is a covariance matrix whose elements are co-variances of the error vector which is the difference between true state and estimated state.

define $\tilde{x}_k = x_k - \hat{x}_k$ as error in state estimate.

Then at k^{th} time step the error covariance matrix is

$P_k = E[\tilde{x}_k \tilde{x}_k^T] \qquad (2)$

Clearly the diagonal of P_k contains the variances of the error. To minimize the error means minimizing the deviation of error $\tilde{x}$ from its mean which in this case is 0 (since $E(\tilde{x}) = E(x) - E(\hat{x}) = 0$ since expected value of $\hat{x}$ is true value x). Hence kalman gain is calculated by minimizing the trace of P_k, meaning we are minimizing the deviation of estimated state from true state.

In kalman filter 2 models are needed, the process model which determines how states evolve and measurement model which determines how states affect output. From here onwards, for any variable s we denote $\hat{s}$ as estimate of s, $\hat{s}^-$ as prior estimate of s and $\hat{s}^+$ as updated estimate of s.

The process and measurement models are assumed to include gaussian noise with zero mean in them. These noises account for stochastic nature or uncertainties in our model which may arise due to improper state modelling or some disturbances entering the system or some uncertainties present in sensor outputs.

Denote process noise by w_k and measurement noise by v_k, then $Q = E(w_k w_k^T)$ is the process noise covariance matrix and $R = E(v_k v_k^T)$ is the measurement noise covariance matrix.

The process model is given as :

$x_k = A_{k-1}x_{k-1} + w_k \;;\quad w_k \sim N(0, Q)$

measurement model is:

$z_k = H_k x_k + v_k \;\; ; v_k \sim N(0, R)$

The kalman filter algorithm is:

Algorithm 1: KALMAN FILTER

```
1 Initialize $\hat{x}_0^+, P_0^+$ for $k = 0$
2 $k = k + 1$
3 Prior estimate:
    $\hat{x}_k^- = A_{k-1}\hat{x}_{k-1}^+$
    $P_k^- = A_{k-1}P_{k-1}^+A_{k-1}^T + Q$
4 Measurement of output:
    $z_k$
5 Innovation:
    $y_k = z_k - H_k\hat{x}_k^-$
6 Kalman gain:
    $K_k = P_k^- H_k^T(H_k P_k^- H_k^T + R)^{-1}$
7 Updated estimate:
    $\hat{x}_k^+ = \hat{x}_k^- + K_k y_k$
    $P_k^+ = (I - K_k H_k)P_k^-$
8 go to step 2
```

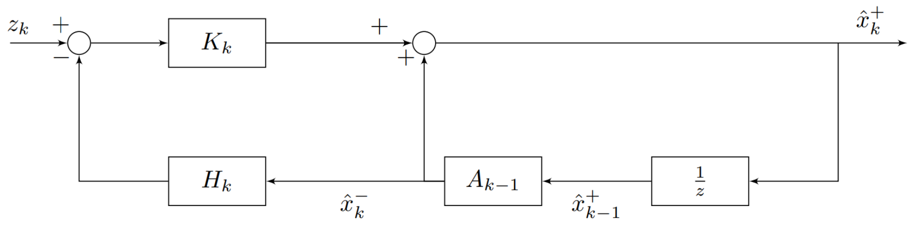

Figure 1: Block diagram of Kalman filter

3 Implementation Details

We tried implementing 2 different methods. The 1st method did not give proper results, hence we finally implemented the second one. We briefly describe the 2 methods in the following subsections.

3.1 Method-1

Here the state vector comprised of yaw,pitch and roll angles. The process model involves making a prior estimate using gyroscope data and then measurement model updates the estimate using accelerometer readings. Hence

$$x_k = \begin{bmatrix} \psi_k \\ \theta_k \\ \phi_k \end{bmatrix}$$

If $w_k = [w_x, w_y, w_z]^T$ is the angular velocity obtained from gyroscope, we define the input u_{k-1} as:

$$u_{k-1} = \begin{bmatrix} w_x \Delta t \\ w_y \Delta t \\ w_z \Delta t \end{bmatrix}$$

The process model can be defined as:

$$x_k = x_{k-1} + u_{k-1} + w_k$$

Here $A_{k-1} = I_3$

The next step is to develop measurement model. Here measurement is readings from accelerometer.

$$z_k = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}.$$

After estimating states using process model we can use those yaw,pitch and roll angles to find the rotation matrix,

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

Then we can relate the states to the output by:

$$\hat{z}_k = R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

$$\Rightarrow \hat{z}_k = \begin{bmatrix} -gsin\theta \\ gsin\phi cos\theta \\ gcos\phi cos\theta \end{bmatrix}$$

This is a nonlinear equation of states. We hence linearize this to find the measurement matrix H_k as:

$$H_k = \begin{bmatrix} 0 & -gcos\theta & 0 \\ 0 & -gsin\phi sin\theta & gcos\phi cos\theta \\ 0 & -gcos\phi sin\theta & -gsin\phi cos\theta \end{bmatrix}$$

So the measurement model is:

$$z_k = H_k x_k + v_k$$

As states earlier, this implementation method did not give proper results. The problem we faced was the estimated yaw,pitch and roll angles were very noisy and further the values of yaw, pitch and roll were underestimated, as in the estimated angles were 30-40 degrees off by the actual values. Section-5 presents the plots of yaw,pitch and roll angles with respect to time, where this problem can be easily seen.

The noisy behaviour can be due to the fact that accelerometer not only measures acceleration due to gravity but also linear acceleration which is caused due to motion of sensor and other vibrations. This can be modelled as high frequency noise which caused the noisy behaviour of estimates. The underestimation is mainly due to the gyro zero rate offset. Basically gyroscope when no motion is present outputs some small value which is not exactly 0, and this offset gets accumulated over time which causes the high drift in estimated angles.

3.2 Method-2

The method implemented here is the same method implemented in [1]. For detailed explanation one can refer [1]. Here we provide a brief summary of the same.

To overcome method-1's problems , here we include the gyro offset and linear acceleration also along with the 3 angles to be estimated.

The process model models the error process:

$$x_{\varepsilon,k} = A_{k-1} x_{\varepsilon,k-1} + w_k$$

$$\Rightarrow x_{\varepsilon,k} = \begin{bmatrix} o_{\varepsilon,k} \\ b_{\varepsilon,k} \\ {}^{S}\mathrm{a}_{\varepsilon,k} \end{bmatrix} = A_{k-1} \begin{bmatrix} o_{\varepsilon,k-1} \\ b_{\varepsilon,k-1} \\ {}^{S}\mathrm{a}_{\varepsilon,k-1} \end{bmatrix} + w_k$$

where $x_{\varepsilon,k}$ is a 9x1 state vector comprising:

- $o_{\varepsilon,k}$: o is a 3x1 vector orientation vector comprising of roll,pitch and yaw angles. So $o_{\varepsilon,k}$ is the 3x1 orientation error vector at time step k which defines the angle by which estimated value exceeds true value.
- $b_{\varepsilon,k}$: it is a 3x1 vector representing gyro zero rate offset for x,y,z axes.
- ${}^S\mathrm{a}_{\varepsilon,k}$: it is a 3x1 vector that represents the error in estimation of linear acceleration in sensor frame

It is assumed that the error in state estimation is independent of error in previous time step. Hence $A_{k-1} = 0$. This means the prior estimate of error is taken as 0, i.e $x^-_{\varepsilon,k} = 0$

Next we develop measurement model. The measurement in this case is the difference of the gravity vector in sensor frame measured by gyroscope and accelerometer. Let ${}^S\mathrm{g}_{G,k}$ be the gravity vector in sensor frame calculated using gyroscope readings and ${}^S\mathrm{g}_{A,k}$ be gravity vector in sensor frame which are nothing but accelerometer readings. Then

$z_k = {}^S\mathrm{g}_{G,k} - {}^S\mathrm{g}_{A,k}$.

Let $w_k \in R^3$ be the gyroscope readings, and $b_k \in R^3$ be the gyroscope drift. To compensate for drift we need to subtract this from gyroscope readings. If o_k is the orientation in prevoius time step, then

$\Delta o_k = (w - b_k)dt$

The new orientation calculated is:

$o_k = o_k + \Delta o$

Using the new orientation one can find the rotation matrix R and use it to find:

$${}^S\mathrm{g}_{G,k} = R \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

Let ${}^S\mathrm{a}_k$ be the current linear acceleration, and $a = [a_x, a_y, a_z]$ be accelerometer readings then :

${}^S\mathrm{g}_{A,k} = a - {}^S\mathrm{a}_k$

The measurement matrix H_k relates the 9x1 state vector to z_k, and it's derivation can be found in [1]. Let ${}^S\mathrm{g}_{G,k} = [g_x, g_y, g_z]^T$, then

$$H = \begin{bmatrix} 0 & g_z & -g_y & 0 & -g_z\Delta t & g_y\Delta t & 1 & 0 & 0 \\ -g_z & 0 & g_x & g_z\Delta t & 0 & -g_x\Delta t & 0 & 1 & 0 \\ g_y & -g_x & 0 & -g_y\Delta t & g_x\Delta t & 0 & 0 & 0 & 1 \end{bmatrix}$$

Once the posterior estimate of state, $x^+_{\varepsilon,k}$ is found, we use this to update the orientation, o_k, gyro offset b_k and linear acceleration ${}^S\mathrm{a}_k$.

1. $o_k = o_{k-1} + x^+_{\varepsilon,k}(1:3)$
2. $b_k = b_{k-1} - x^+_{\varepsilon,k}(4:6)$
3. Linear acceleration is modelled as:

   ${}^S\mathrm{a}_k = c({}^S\mathrm{a}_{k-1})$, where $c \in [0,1]$ is decay factor. Once we find posterior error estimate we can update this as:

   ${}^S\mathrm{a}_k = c({}^S\mathrm{a}_{k-1}) - x^+_{\varepsilon,k}(7:9)$

   Since this method includes estimation of linear acceleration nd gyro offset, it yields much better results, which can be seen in Section-5.

4 Matlab Code

```
%——— section — 1: Interfacing IMU6050 sensor with
    Arduino Mega250 ———
clc;
clear all;
g=9.8;
% a = arduino(); %Update the name of communication port
a = arduino('COM5', 'Mega2560', 'Libraries', 'I2C');
fs = 20; % Sample Rate in Hz
imu = mpu6050(a,'SampleRate',fs,'OutputFormat','matrix');
%%
%——— section — 2: Reading the realtime data from
    IMU6050 sensor ———
decim = 1;
duration = 1; % seconds
fs = 20;          % Hz
N = duration*fs;
i1=0;
t=(0:((N*10)-1)).'/fs;
accelR=[];
gyroR=[];
```

```
openExample('shared_fusion_arduinoio/
    EstimateOrientationUsingInertialSensorFusionAndMPU9250Example
    ')
viewer_imuf = HelperOrientationViewer('Title',{'
    Visualization of imuf_Orientation'})
viewer_CF = HelperOrientationViewer('Title',{'
    Visualization of CF_Orientation'})

orientation_KF = zeros(N*10,1,'quaternion');
angles_prev=zeros(10,3);

%%%%%%%%%%%%%% Iniialization %%%%%%%%%
o=[0;0;0]; %orientation=o=[roll;pitch;yaw]
gyro_offset=[0;-0.09;0];
lin_accel=[0;0;0];

x_pre=zeros((N+1)*10,9);
x_up=zeros((N+1)*10,9);
x0_est=[0.01*ones(10,3),0.0*ones(10,3),0.0*ones(10,3)];
x_up(1:10,:)=x0_est;
P_up=diag([0.01^2;0.01^2;0.01^2;0^2;0^2;0^2;0^2;0^2;0^2])
    *100;
Q=1*diag
    ([0.000006092348396;0.000006092348396;0.000006092348396;

     0.000076154354947;0.000076154354947;0.000076154354947;

     0.009623610000000;0.009623610000000;0.009623610000000])
        ;
R=1*diag
    ([0.0099074550003;0.0099074550003;0.0099074550003]);
dt=1/fs;

while i1<N
     [accelReadings, gyroReadings] = read(imu);
     i1=i1+1;
     accelR = [accelR;accelReadings];
     gyroR = [gyroR;gyroReadings];

     q_mat=zeros(10,4);
     Aq=zeros(10,1);
     Bq=zeros(10,1);
     Cq=zeros(10,1);
     Dq=zeros(10,1);

    for r=1:10 % each sample consists of 10 readings, hence we perform the estimation 10 times
        w=gyroReadings(r,:)';
       %%%%%%%% prior estimation%%%%%%%%%%%
       phi=zeros(9,9);
       pred=phi*x_up(((i1-1)*10)+r,:)';
       x_pre((i1*10)+r,:)=pred';
       P_pre=phi*P_up*phi'+Q;


        %%% finiding z_k %%%%%%%%%%%%%%%%%%
        del_o=(w-gyro_offset)*dt; %gyro_offset is always subtracted to compensate for the drift
        o=o+del_o;
        R=eul2rotm([o(3,1),o(2,1),o(1,1)]);
        g_gyro=R*[0;0;g];
        g_accel=accelReadings(r,:)'-lin_accel;
        z=g_gyro-g_accel;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%  measurement matrix %%%%%%%%
        %%%%%%%%  and kalman gain %%%%%%%%%%
        gx=g_gyro(1,1);
        gy=g_gyro(2,1);
        gz=g_gyro(3,1);
        H=[ 0  , gz, -gy, 0, -dt*gz, dt*gy, 1, 0, 0;
            -gz, 0, gx, dt*gz, 0, -dt*gx, 0, 1, 0;
            gy, -gx, 0, -dt*gy, dt*gx, 0, 0, 0, 1];
        z_hat=H*pred;
        y=z-z_hat;
        K=P_pre*(H')*inv(H*P_pre*H'+R);


        %%%%%%%%%% updates estimates %%%%%%%%%%%%%%
        update=pred+K*y;

        x_up((i1*10)+r,:)=(update');
        P_up=(eye(9)-K*H)*P_pre;


        %%%%%%% update orientation,%%%%%%%%%%
        %%%%%%% gyro offset & lin_accel%%%%%%
        o=o+update(1:3);
        decay=0.5;
        lin_accel=decay*lin_accel-update(7:9); % lin_accel model is ak=c*ak-1+w where w=noise c
```

```
           =decay
        gyro_offset=gyro_offset-update(4:6);


        %%%%%%% convert orientation to%%%%%%%%%
        %%%%%%% quaternion to visulaize %%%%%%%
        q1=[cos(0*o(3,1)/2),0,0,sin(0*o(3,1)/2)];
        q2=[cos(o(2,1)/2),0,sin(o(2,1)/2),0];
        q3=[cos(o(1,1)/2),sin(o(1,1)/2),0,0];
        qmul=quatmultiply(q1,quatmultiply(q2,q3));
        q_mat(r,:)=[qmul(1),qmul(2),qmul(3),qmul(4)];
         Aq(r,1)=qmul(1,1);
        Bq(r,1)=qmul(1,2);
        Cq(r,1)=qmul(1,3);
        Dq(r,1)=qmul(1,4);




    end

    orientation_KF(1+10*(i1-1):i1*10)=quaternion(Aq,Bq,Cq
        ,Dq);
    q_anim=quaternion(Aq,Bq,Cq,Dq);

    fuse = imufilter('SampleRate',fs,'DecimationFactor',
        decim);
    orientation = fuse(accelR,gyroR);

    % 3D figure or Sensor
    for j =1: 10

        viewer_CF(q_anim(j));
    end

    orientationEuler = eulerd(orientation,'ZYX','frame');
end
orientation_KF=eulerd(orientation_KF,'ZYX','frame');
disp((x_up*180/pi));
N=N*10;
timeVector = (0:(N-1))/fs;
figure(3)
plot(timeVector,orientationEuler(:,[2 3]));
xlabel('Time(s)')
ylabel('Rotation(deg)')
title('orientation ---IMUFILTER');
```

```
legend('Y-axis(PITCH)','X-axis(ROLL)');
timeVector = (0:(N-1))/fs;
figure(4)
plot(timeVector,orientation_KF(:,[2 3]));
xlabel('Time(s)')
ylabel('Rotation(deg)')
title('orientation --KALMAN filter');
legend('Y-axis(PITCH)','X-axis(ROLL)');
figure
subplot(2,1,1)
plot(timeVector,accelR)
legend('X-axis','Y-axis','Z-axis')
ylabel('Acceleration (m/s^2)')
title('Accelerometer Readings')

subplot(2,1,2)
plot(timeVector,gyroR)
legend('X-axis','Y-axis','Z-axis')
ylabel('Angular Velocity (rad/s')
xlabel('Time (s)')
title('Gyroscope Readings')
```

5 Results

In this section, the plots of roll and pitch angles over a time of 10 seconds are presented. For comparison, we use the MATLAB's inbuilt IMUFILTER.

NOTE:
The Left plot in each figure corresponds to roll and pitch angles estimated by the Kalman filter implemented by us in MATLAB while the right plot corresponds to the same angles estimated by MATLAB's inbuilt IMUFILTER.

Figure-2,3 show the results obtained using Method-1. As stated in section-3, the results are not good, very noisy and a significant error exists in estimation. Further it can be seen that figure-3 is more noisy and has lesser error in estimation than figure-3. This is because, for figure-2 high values for Q,R were used which makes the system more noisy, but these high values indicate the actual uncertainties present in system model which helps the kalman filter properly minimize the error and hence better accuracy. Whereas figure-3 has low Q,R which makes results less noisy, but does not properly capture the actual uncertainties present in system which leads to lesser accuracy.

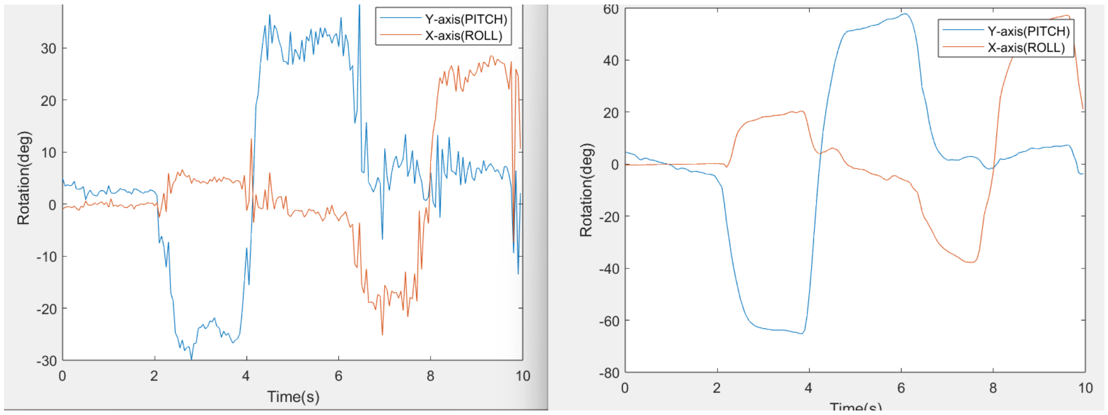

Figure 2:

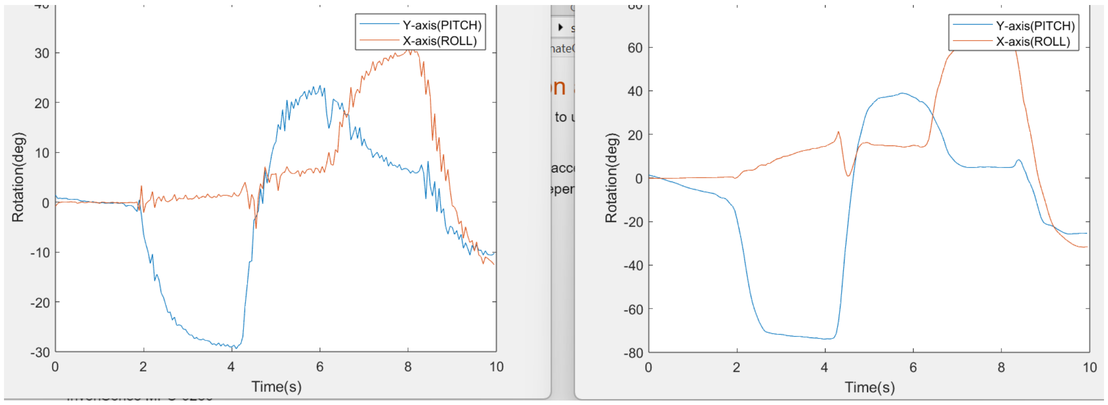

Figure 3:

Figure-4,5,6 present results obtained usng method-2. Cleaarly the results are much better than method-1.

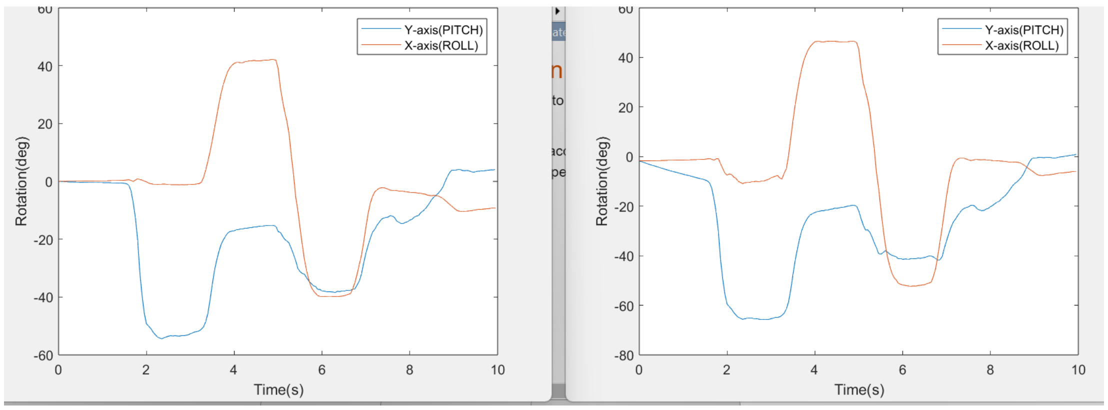

Figure 4:

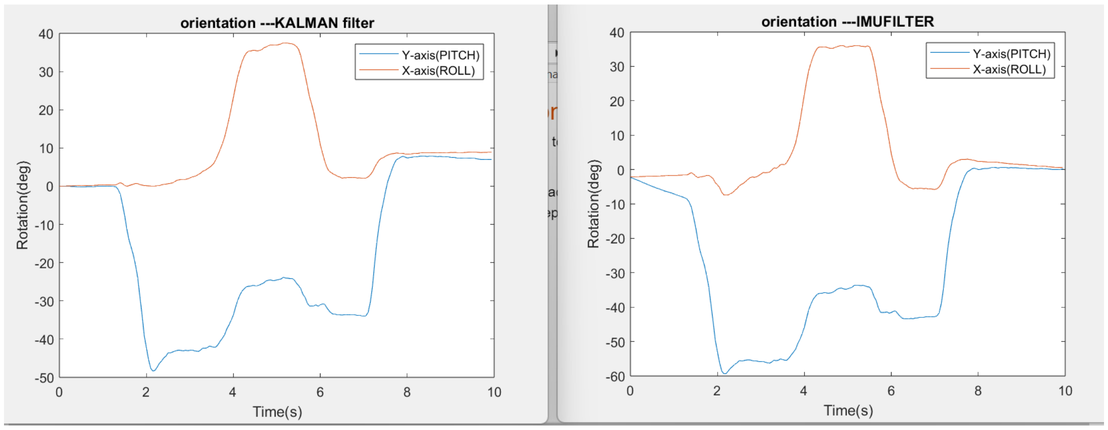

Figure 5:

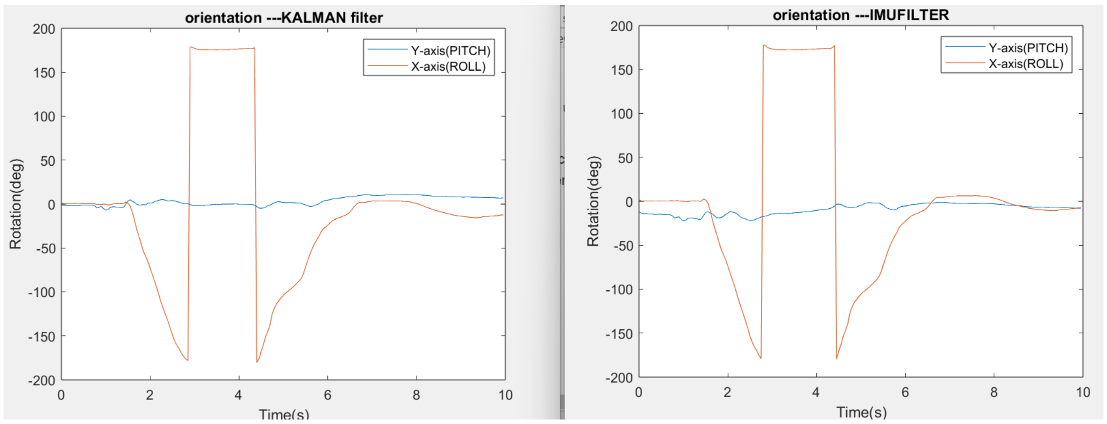

Figure 6:

6 References

1. Open Source Sensor Fusion. `https://github.com/memsindustrygroup/Open-Source-Sensor-Fusion/tree/master/docs`

2. Quan, Quan. Introduction to multicopter design and control. Singapore: Springer, 2017.