Sensor Fusion using Extended Kalman Filter

EE615 Control and Computational Laboratory

SAURAV KUMAR | 21D070063

SIDDHANT DONGARE | 21D070070

Contents

1 Introduction

1.1 Aim

The purpose of this experiment is to understand and implement sensor fusion of accelerometer and gyrometer sensors using Extended Kalman Filter (EKF).

1.2 Background

Orientation estimation is an integral part of the attitude control of aerial vehicles. As the sensor values can be noisy and inaccurate. Appropriate estimation techniques are required to separate the values from noise. Extended Kalman Filter is one of the most popular techniques used for these vehicles to fuse the data from both sensors as explained in [3]. This approach works for Inertial Measurement Units (IMUs) as well.

This is not an original idea, most of the text and equations described below are studied from [2] and [3].

1.3 Capturing Orientation as a Quaternion

Any orientation in a three-dimensional euclidean space of a frame A with respect to a frame B can be represented by a unit quaternion , $q \in H^4$, in Hamiltonian space defined as:

$$ {}^{B}_{A}\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ e_x \sin\frac{\alpha}{2} \\ e_y \sin\frac{\alpha}{2} \\ e_z \sin\frac{\alpha}{2} \end{bmatrix} \tag{1} $$

where α is the rotation angle and e is the unit vector representing the rotation axis. The conjugate quaternion is used to represent the orientation of frame B relative to frame A:

$$ {}^{B}_{A}\mathbf{q}^* = {}^{A}_{B}\mathbf{q} = \begin{bmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{bmatrix} \tag{2} $$

2 Methodology

Input and Output We are using the accelerometer and gyrometer data provided by matlab in the file `rpy_9axis.mat` as input and the output is the orientation plot with samples. This is done to easily compare with the in-built MATLAB function.

Later on, we also interface an actual sensor (mpu6050 sensor) with an arduino to get more data and plots.

2.1 Base Code

```
ld = load('rpy_9axis.mat');
accelR = ld.sensorData.Acceleration;
gyroR = ld.sensorData.AngularVelocity;
magR = ld.sensorData.MagneticField;
Fs  = ld.Fs;  % Hz

fuse = complementaryFilter('SampleRate', Fs, 'HasMagnetometer', false);
orientationMatlab = fuse(accelR, gyroR);
orientationEulerMatlab = eulerd(orientationMatlab,'ZYX','frame');
plot(orientationEulerMatlab);
title('Orientation Estimate');
legend('Z-rotation', 'Y-rotation', 'X-rotation');
xlabel('Sample')
ylabel('Degrees');
```

3 Quaternion-Based Extended Kalman Filter

We implement the Quaternion EKF where attitude quaternion is obtained with gyroscope and accelerometer measurements, via the extended Kalman filter using the approach from [2]. The process is divided into three stages, initialization, prediction and correction.

3.1 Initialization

We use the EKF to estimate an orientation represented as a quaternion $\mathbf{q}$. With the quaternion $\mathbf{q}$ as the state vector, and the angular velocity $\boldsymbol{\omega}$, in rad/s, as the control vector.

$$\begin{aligned} \mathbf{x} &\triangleq \mathbf{q} = \begin{bmatrix} q_w & \mathbf{q}_v \end{bmatrix}^T = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T \\ \mathbf{u} &\triangleq \boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T \end{aligned} \tag{3}$$

Hence, the transition models are

$$\dot{\mathbf{q}}_t = \mathbf{A}\mathbf{q}_{t-1} + \mathbf{B}\boldsymbol{\omega}_t + \mathbf{w}_\mathbf{q} \qquad \text{where } \mathbf{w}_\mathbf{q} \text{ is the process noise} \tag{4}$$

$$\hat{\mathbf{q}}_t = \mathbf{F}\mathbf{q}_{t-1} = e^{\mathbf{A}\Delta t}\mathbf{q}_{t-1} \qquad \text{where } \Delta t \text{ is the inverse of sampling frequency} \tag{5}$$

Initial state is taken as $\mathbf{q}_0$ and the initial estimated covariance of the state $\mathbf{P}_0 = \mathbf{I}_4$

$$\mathbf{q}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T \tag{6}$$

Another suggested initial value is using the normalized initial accelerometer readings

$$\mathbf{C} = \begin{bmatrix} c_{ij} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{a}_0 \end{bmatrix} \quad \Rightarrow \quad \mathbf{q}_0 = \begin{bmatrix} \frac{1}{2}\sqrt{c_{11}+c_{22}+c_{33}+1} \\ \frac{1}{2}\text{sgn}(c_{32}-c_{23})\sqrt{c_{11}-c_{22}-c_{33}+1} \\ \frac{1}{2}\text{sgn}(c_{13}-c_{31})\sqrt{c_{22}-c_{33}-c_{11}+1} \\ \frac{1}{2}\text{sgn}(c_{21}-c_{12})\sqrt{c_{33}-c_{11}-c_{22}+1} \end{bmatrix} \tag{7}$$

Also, the standard deviation of the measurements from the gyrometer sensor and the accelerometer sensor are taken as $\sigma_{\boldsymbol{\omega}} = 0.3$ rad/s and $\sigma_{\mathbf{a}} = 0.5$ rad/s respectively.

3.2 Prediction

3.2.1 Linearization

The next state $\hat{\mathbf{q}}_t$ is calculated by applying $\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)$ to the previous state.

$$\hat{\mathbf{q}}_t = \begin{bmatrix} \hat{q_w} \\ \hat{q_x} \\ \hat{q_y} \\ \hat{q_z} \end{bmatrix} = \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) = \underbrace{\left(\mathbf{I}_4 + \frac{\Delta t}{2}\boldsymbol{\Omega}_t\right)}_{\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)} \mathbf{q}_{t-1} \quad \text{where } \boldsymbol{\Omega}_t = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & \lfloor \boldsymbol{\omega} \rfloor_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \tag{8}$$

3.2.2 Process Noise Covariance

The Process Noise Covariance matrix $\mathbf{Q}_t$ is calculated as

$$\mathbf{Q}_t = \sigma_{\boldsymbol{\omega}}^2 \mathbf{W}_t \mathbf{W}_t^T \quad \text{where } \mathbf{W}_t = \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \tag{9}$$

We use this to calculate $\hat{\mathbf{P}}_t$, the Predicted Covariance of the state *before seeing the measurements* as

$$\hat{\mathbf{P}}_t = \mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)\mathbf{P}_{t-1}\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)^T + \mathbf{Q}_t \tag{10}$$

3.3 Correction

First, we calculate the Measurement $\mathbf{z}_t$ by normalizing the accelerometer readings

$$\mathbf{z}_t = \mathbf{a} = \frac{1}{\sqrt{a_x^2 + a_y^2 + a_z^2}} \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T \tag{11}$$

3.3.1 Measurement Model

The Measurement Model $\mathbf{h}(\hat{\mathbf{q}}_t)$ and its Jacobian $\mathbf{H}(\hat{\mathbf{q}}_t)$ which is called the Observation Model linking the predicted state and the measurements are defined as follows assuming the gravity is

$$\mathbf{g} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \quad \text{In East-North-Up (ENU) system.} \tag{12}$$

$$\mathbf{h}(\hat{\mathbf{q}}_t) = \hat{\mathbf{a}} = \mathbf{C}(\hat{\mathbf{q}})^T \mathbf{g} = \begin{bmatrix} 1 - 2(\hat{q}_y^2 + \hat{q}_z^2) & 2(\hat{q}_x\hat{q}_y - \hat{q}_w\hat{q}_z) & 2(\hat{q}_x\hat{q}_z + \hat{q}_w\hat{q}_y) \\ 2(\hat{q}_x\hat{q}_y + \hat{q}_w\hat{q}_z) & 1 - 2(\hat{q}_x^2 + \hat{q}_z^2) & 2(\hat{q}_y\hat{q}_z - \hat{q}_w\hat{q}_x) \\ 2(\hat{q}_x\hat{q}_z - \hat{q}_w\hat{q}_y) & 2(\hat{q}_w\hat{q}_x + \hat{q}_y\hat{q}_z) & 1 - 2(\hat{q}_x^2 + \hat{q}_y^2) \end{bmatrix}^T \mathbf{g} \tag{13}$$

$$\mathbf{H}(\hat{\mathbf{q}}_t) = 2 \begin{bmatrix} g_y q_z - g_z q_y & g_y q_y + g_z q_z & -2g_x q_y + g_y q_x - g_z q_w & -2g_x q_z + g_y q_w + g_z q_x \\ -g_x q_z + g_z q_x & g_x q_y - 2g_y q_x + g_z q_w & g_x q_x + g_z q_z & -g_x q_w - 2g_y q_z + g_z q_y \\ g_x q_y - g_y q_x & g_x q_z - g_y q_w - 2g_z q_x & g_x q_w + g_y q_z - 2g_z q_y & g_x q_x + g_y q_y \end{bmatrix} \tag{14}$$

We subsequently calculate the intermediate matrices which lead us to the estimated next state.

$$\mathbf{R} = \boldsymbol{\sigma}_{\mathbf{a}}^2 \mathbf{I}_3 \qquad \text{Measurement Noise Covariance Matrix} \tag{15}$$

$$\mathbf{v}_t = \mathbf{z}_t - \mathbf{h}(\hat{\mathbf{q}}_t) \qquad \text{Measurement Residual} \tag{16}$$

$$\mathbf{S}_t = \mathbf{H}(\hat{\mathbf{q}}_t)\hat{\mathbf{P}}_t\mathbf{H}^T(\hat{\mathbf{q}}_t) + \mathbf{R} \qquad \text{Measurement Prediction Covariance} \tag{17}$$

$$\mathbf{K}_t = \hat{\mathbf{P}}_t\mathbf{H}^T(\hat{\mathbf{q}}_t)\mathbf{S}_t^{-1} \qquad \text{Kalman Gain} \tag{18}$$

With this we calculate the estimated covariance of the state *after seeing the measurements* $\mathbf{P}_t$ and normalize the corrected quaternion $\mathbf{q}_t$ to avoid its divergence.

$$\begin{aligned} \mathbf{q}_t &= \hat{\mathbf{q}}_t + \mathbf{K}_t\mathbf{v}_t \quad \Rightarrow \quad \mathbf{q}_t \leftarrow \frac{1}{\|\mathbf{q}_t\|}\mathbf{q}_t \\ \mathbf{P}_t &= \big(\mathbf{I}_4 - \mathbf{K}_t\mathbf{H}(\hat{\mathbf{q}}_t)\big)\hat{\mathbf{P}}_t \end{aligned} \tag{19}$$

The prediction and the correction steps are to be repeated for each sample of data.

3.4 Implementation

```
countSamples = size(accelR, 1);
delta_t = 1/fs;
orientation = quaternion.zeros(countSamples, 1);
[current_attitude, P_mat, variance_gyro, variance_accel] = initialize(accelR(1,:));
for idx = 1:countSamples
    accelReadings = accelR(idx,:);
    gyroReadings = gyroR(idx,:);
    [current_attitude,P_mat] = prediction(current_attitude, gyroReadings, P_mat, variance_gyro, delta_t);
    [current_attitude,P_mat] = correction(current_attitude, accelReadings, P_mat, variance_accel);
    orientation(idx, :) = current_attitude;
end
% 3D figure or Sensor
for j = numel(orientation)
   viewer(orientation(j));
end
orientationEuler = eulerd(orientation,'ZYX','frame');

% Supporting Functions
function [q_0, P_0, variance_gyro, variance_accel] = initialize(accel)
```

```
    q_0 = quaternion(1,0,0,0);
    % Uncomment below to initialize using accelerator readings
    % accel_normalized = (accel/norm(accel))';
    % C = [[0;0;0] [0;0;0] accel_normalized];
    % q_0 = quaternion(sqrt(C(1,1)+C(2,2)+C(3,3)+1), sign(C(3,2)-C(2,3))*sqrt(C(1,1)-C(2,2)-C(3,3)+1),
↪   sign(C(1,3)-C(3,1))*sqrt(C(2,2)-C(1,1)-C(3,3)+1), sign(C(2,1)-C(1,2))*sqrt(C(3,3)-C(2,2)-C(1,1)+1));
    % q_0 = normalize(q_0);
    P_0 = eye(4);
    variance_gyro = 0.3^2;
    variance_accel = 0.5^2;
end
function [new_quat,new_P_mat] = prediction(quat, w, P_mat, variance_gyro, delta_t)
    [qw, qx, qy, qz] = parts(quat);
    wx = w(1);
    wy = w(2);
    wz = w(3);
    Omega_matrix = [0 -wx -wy -wz;
                    wx 0 wz  -wy;
                    wy -wz 0 wx;
                    wz wy -wx 0];

    F_mat = eye(4)+(delta_t/2)*Omega_matrix;
    new_quat = quaternion((F_mat*[qw, qx, qy, qz]')');
    W_mat = (delta_t/2)*[-qx -qy -qz;
                qw -qz qy;
                qz qw -qx;
                -qy qx qw];
    Q_mat = variance_gyro*(W_mat*W_mat');
    new_P_mat = F_mat*P_mat*F_mat' + Q_mat;
end
function [q_t,P_t] = correction(quat, accel, P_mat, variance_accel)
    [qw, qx, qy, qz] = parts(quat);
    accel_normalized = accel/norm(accel);
    z_t = accel_normalized';
    C_mat = [1-2*(qy^2+qz^2)     2*(qx*qy-qw*qz)      2*(qx*qz+qw*qy);
            2*(qx*qy+qw*qz)     1-2*(qx^2+qz^2)     2*(qy*qz-qw*qx);
            2*(qx*qz-qw*qy)     2*(qw*qx+qy*qz)     1-2*(qx^2+qy^2)];
    g = [0 0 1]';
    gz = g(3);
    a_hat = C_mat'*g;
    h_vec = a_hat;
    H_mat = 2*[-gz*qy     gz*qz     -gz*qw     gz*qx;
                gz*qx     gz*qw      gz*qz     gz*qy;
                0         -2*gz*qx   -2*gz*qy     0];
    R_mat = variance_accel*eye(3);
    v_t = z_t - h_vec;
    S_t = H_mat*P_mat*H_mat' + R_mat;
    K_t = (P_mat*H_mat')*inv(S_t);
    q_t = quat+quaternion((K_t*v_t)');
    P_t = (eye(4)-K_t*H_mat)*P_mat;
    q_t = sign(parts(q_t))*q_t;
    q_t = normalize(q_t);
end
```

4 Simulation and Modeling

4.1 Simulation Environment and Model structure

MATLAB version 2020A was used.

4.2 Interfacing the sensor with arduino

We can also use mpu6050 sensor interfaced with arduino mega 2560 using I2C communication to get estimated readings.

```
% Interfacing IMU6050 sensor with Arduino Mega250 -------
clc;
clear all;
```

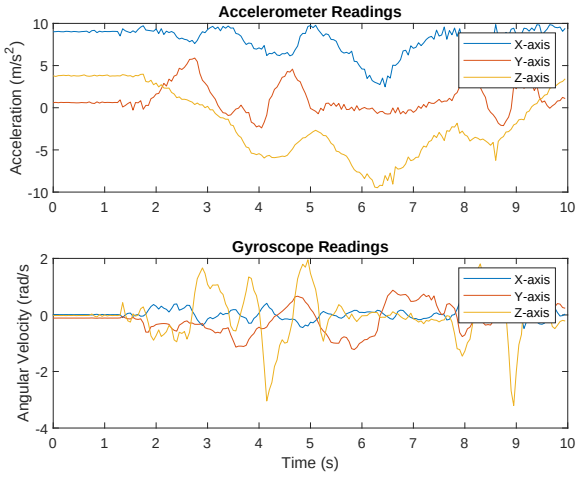
```
% a = arduino(); %Update the name of communication port
a = arduino('COM5', 'Mega2560', 'Libraries', 'I2C');
fs = 20; % Sample Rate in Hz
imu = mpu6050(a,'SampleRate',fs,'OutputFormat','matrix');
%%
% Reading the realtime data from IMU6050 sensor -------
decim = 1;
duration = 1; % seconds
fs = 20;         % Hz
N = duration*fs;
```

4.3 Plotting

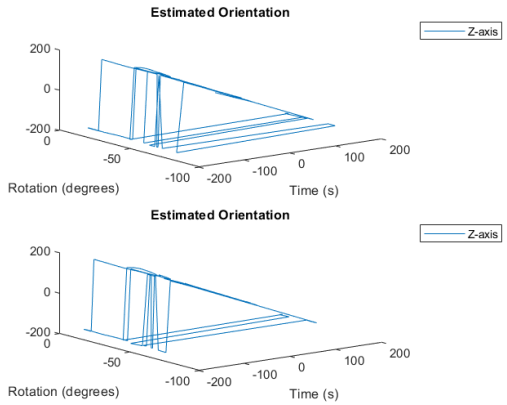
```
% Acceleratometer and Gyrometer readings
N=N*10;
timeVector = (0:(N-1))/fs;
figure
subplot(2,1,1)
plot(timeVector,accelR)
legend('X-axis','Y-axis','Z-axis')
ylabel('Acceleration (m/s^2)')
title('Accelerometer Readings')

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')

% Orientation Estimate with all angles combined
figure
subplot(2,1,1)
plot(eulerd( orientation, 'ZYX', 'frame'));
title('Orientation Estimate');
legend('Z-rotation', 'Y-rotation', 'X-rotation');
xlabel('Sample')
ylabel('Degrees');

subplot(2,1,2)
plot(eulerd( orientationMatlab, 'ZYX', 'frame'));
title('Orientation Estimate');
legend('Z-rotation', 'Y-rotation', 'X-rotation');
xlabel('Sample')
ylabel('Degrees');

% Orientation Estimate with angles separated
figure
subplot(2,3,1)
plot(orientationEuler(:,1))
ylabel('Degrees')
xlabel('Sample')
subplot(2,3,2)
plot(orientationEuler(:,2))
ylabel('Degrees')
xlabel('Sample')
subplot(2,3,3)
plot(orientationEuler(:,3))
ylabel('Degrees')
xlabel('Sample')
subplot(2,3,4)
plot(orientationEulerMatlab(:,1))
ylabel('Degrees')
xlabel('Sample')
subplot(2,3,5)
plot(orientationEulerMatlab(:,2))
ylabel('Degrees')
xlabel('Sample')
subplot(2,3,6)
plot(orientationEulerMatlab(:,3))
```

```
ylabel('Degrees')
xlabel('Sample')

% Orientation Estimate, 3D curve
figure
subplot(2,1,1)
plot3(orientationEuler(:,1),orientationEuler(:,2),orientationEuler(:,3))
legend('Z-axis','Y-axis','X-axis')
xlabel('Time (s)')
ylabel('Rotation (degrees)')
title('Estimated Orientation')

subplot(2,1,2)
plot3(orientationEulerMatlab(:,1),orientationEulerMatlab(:,2),orientationEulerMatlab(:,3))
legend('Z-axis','Y-axis','X-axis')
xlabel('Time (s)')
ylabel('Rotation (degrees)')
title('Estimated Orientation')
```

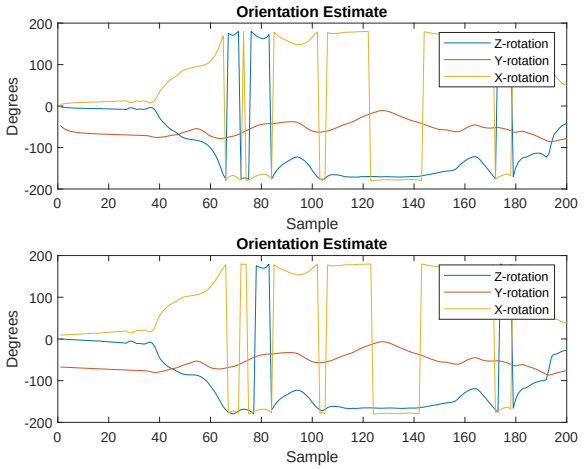
4.4 Simulation Results



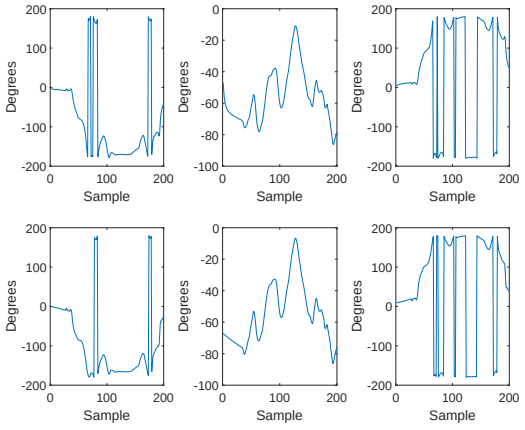
(a) Acceleratometer and Gyrometer readings



(b) Orientation Estimate



(c) Orientation Estimate with all angles combined



(d) Orientation Estimate with angles separated (Z, Y, X) axis respectively

Figure 1: Comparison of our implementation (above) and MATLAB's default implementation (below)

As can be seen in the figures, the estimation is highly accurate in all axes and the variation is similar with some differences possibly due to different constants used.

5 Conclusion

We enjoyed working on the task assigned and were able to complete it successfully.

References

[1] Quan Quan (auth.). *Introduction to Multicopter Design and Control*. Springer Singapore, 1 edition, 2017.

[2] AHRS Mario Garcia. Extended kalman filter. URL: `https://ahrs.readthedocs.io/en/latest/filters/ekf.html`.

[3] Angelo Maria Sabatini. Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation. *Sensors*, 11(10):9182–9206, 2011. URL: `https://www.mdpi.com/1424-8220/11/10/9182`.

[4] Inc. The MathWorks. Estimate orientation with a imu filter and imu data. URL: `https://in.mathworks.com/help/nav/ref/imufilter-system-object.html`.

[5] Inc. The MathWorks. Estimating orientation using inertial sensor fusion and mpu-9250. URL: `https://in.mathworks.com/help/fusion/ug/Estimating-Orientation-Using-Inertial-Sensor-Fusion-and-MPU-9250.html`.