Sensor Fusion using Complementary filter

EE615 Control and Computational Laboratory

Rathour Param Jitendrakumar | 190070049

Tejas Sanjaykumar Pagare | 190070067

Spring Semester 2022-23

Contents

1 Introduction

1.1 Aim

The purpose of this experiment is to understand and implement sensor fusion of accelerometer and gyrometer sensors using complementary filter.

1.2 Background

Orientation estimation is an integral part of the attitude control of aerial vehicles. Sensor values can be noisy and inaccurate. Complimentary filter is a simplistic alternative for these vehicles fuses the data from both sensors as explained in [6]. This approach works for Inertial Measurement Units (IMUs) as well.

This is not an original idea, most of the texts and equations described below are studied from [2], [3] and [6] sources.

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:

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

Given two quaternions $\mathbf{p}$ and $\mathbf{q}$, the cosine of the angle Ω subtended by the arc between them is equal to the dot product of the two quaternions.

$$ \cos\Omega = \mathbf{p} \cdot \mathbf{q} = p_w q_w + p_x q_x + p_y q_y + p_z q_z \tag{3} $$

It is easy to see that the dot product of any $\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}$ and the identity quaternion $\mathbf{q}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ is equal to the q_w component:

$$ q \cdot q_I = q_w \tag{4} $$

The simple Linear intERPolation (LERP) between two quaternions $\mathbf{p}$ and $\mathbf{q}$ is obtained as:

$$ \overline{\mathbf{r}} = (1-\alpha)\mathbf{p} + \alpha\mathbf{q} \tag{5} $$

where $\alpha \in [0, 1]$. But this does not keep the unit norm, so we must normalize the resulting interpolation:

$$ \widehat{\mathbf{r}} = \frac{\overline{\mathbf{r}}}{\|\overline{\mathbf{r}}\|} \tag{6} $$

The Spherical Linear intERPolation (SLERP) gives a correct evaluation of the weighted average of two points lying on a curve. In the case of quaternions, the points lie on the surface of the 4D sphere (hypersphere).

$$ \widehat{\mathbf{r}} = \frac{\sin([1-\alpha]\Omega)}{\sin\Omega}\mathbf{p} + \frac{\sin(\alpha\Omega)}{\sin\Omega}\mathbf{q} \tag{7} $$

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.

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);
orientation = fuse(accelR, gyroR);

plot(eulerd( orientation, 'ZYX', 'frame'));
title('Orientation Estimate');
legend('Z-rotation', 'Y-rotation', 'X-rotation');
xlabel('Sample')
ylabel('Degrees');
```

3 Complementary Filter without correction

Attitude quaternion obtained with gyroscope and accelerometer-magnetometer measurements, via complementary filter using the approach from [3].

First, the current orientation is estimated at time t, from a previous orientation at time $t-1$, and a given angular velocity, ω, in rad/s. The intial orientation $\mathbf{q}_0$ is taken as $[1\ 0\ 0\ 0]^T$

This orientation is computed by numerically integrating the angular velocity and adding it to the previous orientation, which is known as an attitude propagation.

$$\begin{aligned}
\mathbf{q}_\omega &= \left(\mathbf{I}_4 + \tfrac{\Delta t}{2}\mathbf{\Omega}_t\right)\mathbf{q}_{t-1} \\
&= \begin{bmatrix} 1 & -\frac{\Delta t}{2}\omega_x & -\frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z \\ \frac{\Delta t}{2}\omega_x & 1 & \frac{\Delta t}{2}\omega_z & -\frac{\Delta t}{2}\omega_y \\ \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z & 1 & \frac{\Delta t}{2}\omega_x \\ \frac{\Delta t}{2}\omega_z & \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_x & 1 \end{bmatrix} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \\
&= \begin{bmatrix} q_w - \frac{\Delta t}{2}\omega_x q_x - \frac{\Delta t}{2}\omega_y q_y - \frac{\Delta t}{2}\omega_z q_z \\ q_x + \frac{\Delta t}{2}\omega_x q_w - \frac{\Delta t}{2}\omega_y q_z + \frac{\Delta t}{2}\omega_z q_y \\ q_y + \frac{\Delta t}{2}\omega_x q_z + \frac{\Delta t}{2}\omega_y q_w - \frac{\Delta t}{2}\omega_z q_x \\ q_z - \frac{\Delta t}{2}\omega_x q_y + \frac{\Delta t}{2}\omega_y q_x + \frac{\Delta t}{2}\omega_z q_w \end{bmatrix}
\end{aligned} \tag{8}$$

Secondly, the tilt is computed from the accelerometer measurements as:

$$\begin{aligned}
\phi &= \arctan2(a_y, a_z) \\
\theta &= \arctan2(-a_x, \sqrt{a_y^2 + a_z^2})
\end{aligned} \tag{9}$$

Only the pitch, θ , and roll, ϕ , angles are computed, leaving the yaw angle, ψ equal to zero.

We transform the roll-pitch-yaw angles to a quaternion representation:

$$\mathbf{q}_{am} = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \end{pmatrix} \tag{10}$$

Finally, after each orientation is estimated independently, they are fused with the complementary filter.

$$\mathbf{q_t} = (1-\alpha)\mathbf{q}_\omega + \alpha\mathbf{q}_{am} \tag{11}$$

where $\mathbf{q}_\omega$ is the attitude estimated from the gyroscope, $\mathbf{q}_{am}$ is the attitude estimated from the accelerometer and the magnetometer, and α is the gain of the filter.

The filter gain is a floating value within the range [0,1]. When α=1, the attitude is estimated entirely with the accelerometer and When α=0, it is estimated entirely with the gyroscope. These values decide how much of each estimation is "mixed" into the quaternion. We select $\alpha = 0.02$, i.e, focussing more on the gyroscope readings as they are mostly accurate in smaller durations.

This is a simple LERP implementation commonly used to linearly interpolate quaternions with small differences between them.

3.1 Implementation

```
q = quaternion(1,0,0,0); % q_init
orientation = [q];
M = size(accelR);
i = 1;
delta_t = 1/Fs;
alpha = 0.02;
while i<=M(1)
    accelReadings = accelR(i,:);
    gyroReadings = gyroR(i,:);
    wx = gyroReadings(1);
    wy = gyroReadings(2);
    wz = gyroReadings(3);
    w = [wx wy wz]';

    ax = accelReadings(1);
    ay = accelReadings(2);
    az = accelReadings(3);
    a = [ax ay az]';

    % Integration
    w_mat = [0 -wx -wy -wz;
             wx 0 wz -wy;
             wy -wz 0 wx;
             wz wy -wx 0];
    [q0, q1, q2, q3] = parts(q);
    q_omega = quaternion(((eye(4)+delta_t/2*w_mat)*[q0, q1, q2, q3]')');

    % Calculation of pitch, roll, yaw
    phi = atan2(ay, az);
    theta = atan2(-ax, sqrt(ay*ay+az*az));
    psi = 0;

    % Transform the pitch, roll, yaw angles to quaternion orientation
    q_accelerometer = quaternion([
```

```
    cos(phi/2)*cos(theta/2)*cos(psi/2)+sin(phi/2)*sin(theta/2)*sin(psi/2),
    sin(phi/2)*cos(theta/2)*cos(psi/2)-cos(phi/2)*sin(theta/2)*sin(psi/2),
    cos(phi/2)*sin(theta/2)*cos(psi/2)+sin(phi/2)*cos(theta/2)*sin(psi/2),
    cos(phi/2)*cos(theta/2)*sin(psi/2)-sin(phi/2)*sin(theta/2)*cos(psi/2)]);

    % Simple LERP
    q = (1-alpha)*q_omega + alpha*q_accelerometer;

    orientation = [orientation;q];
        i=i+1;
end
```

4 Algebraic Quaternion Algorithm

Attitude quaternion obtained with gyroscope and accelerometer-magnetometer measurements and LERP/SLERP corrections, via algebraic quaternion algorithm using the approach from [2].

4.1 Quaternion-Based Complementary Filter

A complementary filter fuses attitude estimation in quaternion form from gyroscope data with accelerometer and magnetometer data in the form of a delta quaternion.

If only IMU data is provided (gyroscopes and accelerometers), it corrects only roll and pitch of the attitude. If magnetometer data is also provided a second step is added to the algorithm where a magnetic delta quaternion is derived to correct the heading of the previous estimation by aligning the current frame with the magnetic field. In the experiment we only pursue orientation estimation with only gyroscope and magnetometer data.

In the Prediction step the measured angular velocity is used to compute a first estimation of the orientation in quaternion form.

The majority of literature that calculates the quaternion derivative from an angular rate measurement typically focuses on the representation of the orientation of the local frame (L) relative to the global frame (G).

$$ {}^{G}_{L}\dot{\mathbf{q}}_{\omega,t_k} = \frac{1}{2}\, {}^{G}_{L}\mathbf{q}_{t_{k-1}}\, {}^{L}\omega_{q,t_k} \tag{12} $$

However, [6] uses the inverse orientation, so the quaternion derivative is computed using the inverse unit quaternion, which is simply the conjugate:

$$ {}^{L}_{G}\dot{\mathbf{q}}_{\omega,t_k} = {}^{G}_{L}\dot{\mathbf{q}}^{*}_{\omega,t_k} = -\frac{1}{2}\, {}^{L}\omega_{q,t_k}\, {}^{L}_{G}\mathbf{q}_{t_{k-1}} \tag{13} $$

where ${}^{L}\omega_{q,t_k} = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is the measured angular velocity, in radians per second, arranged as a pure quaternion at time t_k , and ${}^{L}_{G}\mathbf{q}_{t_{k-1}}$ is the previous estimate of the orientation.

The orientation of the global frame relative to local frame at time t_k can be finally computed by numerically integrating the quaternion derivative using the sampling period $\Delta t = t_k - t_{k-1}$. (Discrete integration step)

$$ {}^{L}_{G}\mathbf{q}_{\omega,t_k} = {}^{L}_{G}\mathbf{q}_{t_{k-1}} + {}^{L}_{G}\dot{\mathbf{q}}_{\omega,t_k}\Delta t \tag{14} $$

The correction step is based on a multiplicative approach, where the predicted quaternion ${}^{L}_{G}\mathbf{q}_{\omega}$ is corrected by means of two delta quaternions:

$$ {}^{L}_{G}\mathbf{q} = {}^{L}_{G}\mathbf{q}_{\omega}\, \Delta\mathbf{q}_{\text{acc}} \tag{15} $$

The delta quaternions are computed and filtered independently by the high-frequency noise. This correction process consists of two steps: firstly, the correction of the predicted quaternion's roll and pitch, and secondly, the correction of the yaw angle if measurements of the magnetic field are available.

4.2 Accelerometer-Based Correction

The inverse predicted quaternion ${}^{G}_{L}\mathbf{q}_{\omega}$ is used to rotate the normalized body frame gravity vector ${}^{L}\mathbf{a}$, measured by the accelerometer, into the global frame:

$$\mathbf{R}({}^{G}_{L}\mathbf{q}_{\omega})\,{}^{L}\mathbf{a} = {}^{G}\mathbf{g}_{p} \tag{16}$$

where ${}^{G}\mathbf{g}_{p} = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^{T}$ is the predicted gravity, which always has a small deviation from the real gravity vector ${}^{G}\mathbf{g} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. We compute the delta quaternion $\Delta\mathbf{q}_{\text{acc}}$ to rotate ${}^{G}\mathbf{g}$ into ${}^{G}\mathbf{g}_{p}$:

$$\mathbf{R}(\Delta\mathbf{q}_{\text{acc}})\,{}^{G}\mathbf{g} = {}^{G}\mathbf{g}_{p} \tag{17}$$

Similar to the auxiliary quaternions, we find a closed-form solution:

$$\Delta\mathbf{q}_{\text{acc}} = \begin{bmatrix} \sqrt{\frac{g_z+1}{2}} & -\frac{g_y}{\sqrt{2(g_z+1)}} & \frac{g_x}{\sqrt{2(g_z+1)}} & 0 \end{bmatrix}^{T} \tag{18}$$

This has a singularity at $g_z = -1$, but it can be ignored, because the value of g_z will always be closer to 1.

The delta quaternion is affected by the accelerometer's high frequency noise, so we scale it down by using an interpolation with the identity quaternion $\mathbf{q}_I$. As demonstrated above, the dot product with $\mathbf{q}_I$ is equal to the $\Delta q_{w\text{acc}}$ component of $\Delta\mathbf{q}_{\text{acc}}$.

If $\Delta q_{w\text{acc}} > \epsilon$, where ϵ is a threshold value (default is ϵ=0.9), a simple LERP else SLERP is used:

$$\widehat{\Delta\mathbf{q}}_{\text{acc}} = \frac{\overline{\Delta\mathbf{q}}_{\text{acc}}}{\|\overline{\Delta\mathbf{q}}_{\text{acc}}\|} \tag{19}$$

The predicted quaternion from gyroscopes is multiplied with the delta quaternion to correct the roll and pitch components:

$${}^{L}_{G}\mathbf{q}' = {}^{L}_{G}\mathbf{q}_{\omega}\,\widehat{\Delta\mathbf{q}}_{\text{acc}} \tag{20}$$

4.3 Adaptive Gain

If the vehicle moves with high acceleration then the magnitude and direction of the total measured acceleration vector are different from gravity, and so the attitude is calculated using a false reference.

As, linear acceleration doesn't affect gyrometer readings, thus we can still used them to estimate a somewhat accurate attitude.

Hence, we need a adaptive gain algorithm instead of a constant gain fusion algorithm to overcome the above issue when the optimal gain has been evaluated for static conditions.

We first define a magnitude error e_m as follows

$$e_m = \frac{|\|\,{}^{L}\tilde{a}\| - g|}{g}. \tag{21}$$

Here $g = 9.81ms^{-}2$ and $\|\,{}^{L}\hat{a}\|$ is the norm of the local frame acceleration vector we measure before normalization.

Using the LERP and SLERP definitions, the filtering gain α is defined such that it is dependent on the magnitude error e_m through the gain factor f as follows

$$\alpha = \overline{\alpha} f(e_m) \tag{22}$$

where the constant $\overline{\alpha}$ that gives the best filtering result in static conditions. We call $f(e_m)$ as the gain factor, which is a piecewise continuous function of the magnitude error.

This gain factor is equal to 1 when the magnitude of the non-gravitational acceleration is not high enough to overcome the acceleration gravity and the value of the error magnitude does not reach the first threshold t_1 . If the nongravitational acceleration rises and the error magnitude exceeds that first threshold, the gain factor decreases linearly with the increase of the magnitude error until reaching zero for error magnitude equal to the second threshold t2 and over.

$$f(e_m) = \begin{cases} 1 & \text{if } e_m \leq t_1 \\ \frac{t_2 - e_m}{t_1} & \text{if } t_1 < e_m < t_2 \\ 0 & \text{if } e_m \geq t_2 \end{cases} \tag{23}$$

Empirically, the threshold values giving the best results are 0.1 and 0.2 .

4.4 Bias Estimation

A gyrometer's bias reading is a slow-varying signal, so we can consider it as a low frequency noise. By low-pass filtering, we can separate the bias from the actual angular velocity. Note, this can be only done when the sensor is in a steady-state condition.

Now, when the sensor is in the steady-state, the bias is updated else it is assumed to be equal to the previous step value. The estimated bias is then subtracted from the gyroscope reading obtaining a bias-free angular velocity measurement.

This procedure is done initially to better our readings.

4.5 Implementation

```
delta_t = 1/Fs;
AccelerometerGain = 0.01;
threshold = 0.9;
omegaThreshold = 0.1;
accelerationThreshold = 0.1;
deltaomegaThreshold = 0.01;
gyrometerBias = [0 0 0];
gyrometerBiasAlpha = 0.01;
prevOmega = [0 0 0];
countSamples = size(accel, 1);
qI = quaternion(1,0,0,0);
q = quaternion.zeros(countSamples, 1);
omega = zeros(countSamples, 3);
current_attitude = qI;
g = 9.81;

for idx = 1:countSamples
    accelReadings = accelR(idx,:);
    gyroReadings = gyroR(idx,:);
    accelMagnitude = norm(accelReadings);

    w1 = gyroReadings(1);
    w2 = gyroReadings(2);
    w3 = gyroReadings(3);

    % Bias Estimation
    %% Steady State Check
    if (abs(accelMagnitude - g) <= accelerationThreshold)
    && (abs(w1-prevOmega(1)) <= deltaomegaThreshold)
    && (abs(w2-prevOmega(2)) <= deltaomegaThreshold)
    && (abs(w3-prevOmega(3)) <= deltaomegaThreshold)
```

```
    && (abs(w1 - gyrometerBias(1)) <= omegaThreshold)
    && (abs(w2 - gyrometerBias(2)) <= omegaThreshold)
    && (abs(w3 - gyrometerBias(3)) <= omegaThreshold)
        gyrometerBias = gyrometerBias + gyrometerBiasAlpha * (gyroReadings - gyrometerBias);
    end
    prevOmega = gyroReadings;
    % Subtraction of bias from given readings
    omega(idx,:) = gyroR(idx,:) - gyrometerBias;

    % Integration
    current_attitude = current_attitude *
    quaternion((gyroReadings-gyrometerBias)*delta_t, 'rotvec');

    accelNormalized = accelReadings/norm(accelReadings);
    predicted_gravity = rotatepoint(current_attitude, accelNormalized);

    % Finding Correction term using predicted gravity
    deltaAcceleration0 = sqrt((predicted_gravity(3) + 1) / 2);
    deltaAcceleration1 = predicted_gravity(2) / sqrt(2*(predicted_gravity(3)+1));
    deltaAcceleration2 = -predicted_gravity(1) / sqrt(2*(predicted_gravity(3)+1));
    deltaAcceleration3 = 0;
    deltaAcceleration = quaternion(deltaAcceleration0,
    deltaAcceleration1, deltaAcceleration2, deltaAcceleration3);

    % Adaptive Gain
    temp = abs(accelMagnitude - g);
    if temp <= 0.1
        accelGainFactor = 1;
    elseif temp < 0.2
        accelGainFactor = 2 - temp/0.1;
    else
        accelGainFactor = 0;
    end
    accelGain = AccelerometerGain * accelGainFactor;

    if deltaAcceleration0 > threshold % LERP
        corr_deltaAccel = (1-accelGain)*qI+accelGain*deltaAcceleration;
    else % SLERP
        Omega = acos(deltaAcceleration0);
        corr_deltaAccel = (sin((1-accelGain*Omega)/sin(Omega))*qI +
        (sin(accelGainFactor)*Omega)/sin(Omega))*deltaAcceleration;
    end
    corr_deltaAccel = normalize(corr_deltaAccel);

    % Final Upate
    current_attitude = sign(parts(current_attitude))*current_attitude;
    current_attitude = normalize(current_attitude);
    q(idx,:) = current_attitude;
end
```

5 Simulation and Modeling

5.1 Simulation Environment and Model structure

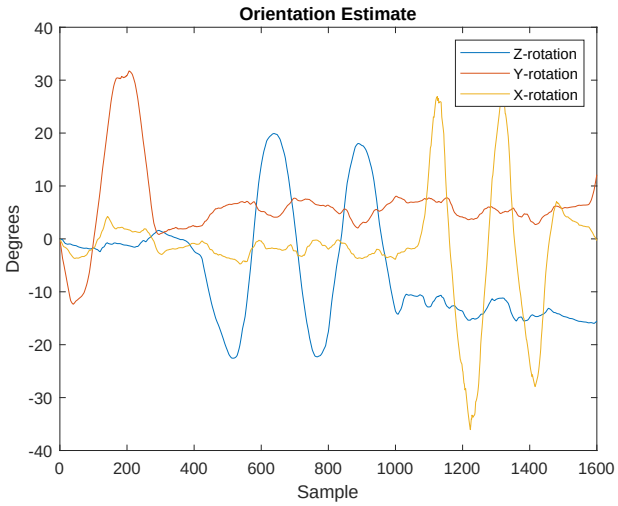
MATLAB version 2020A was used.

5.2 Interfacing the sensor with arduino

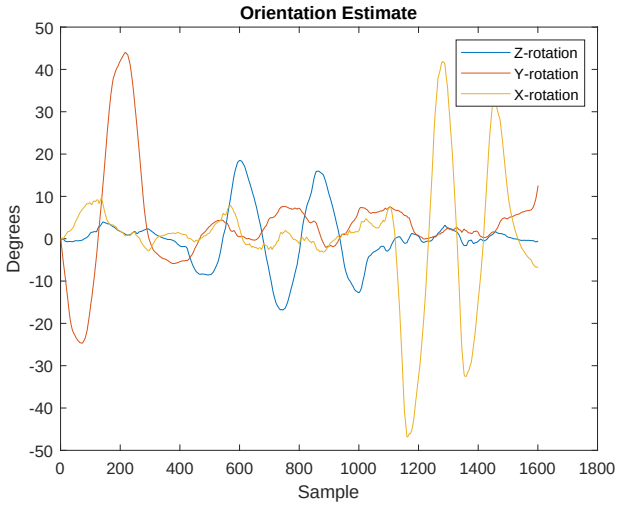
We can also use mpu6050 sensor interfaced with arduino mega 2560 using I2C communication to get estimated readings.

```
% 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;
```

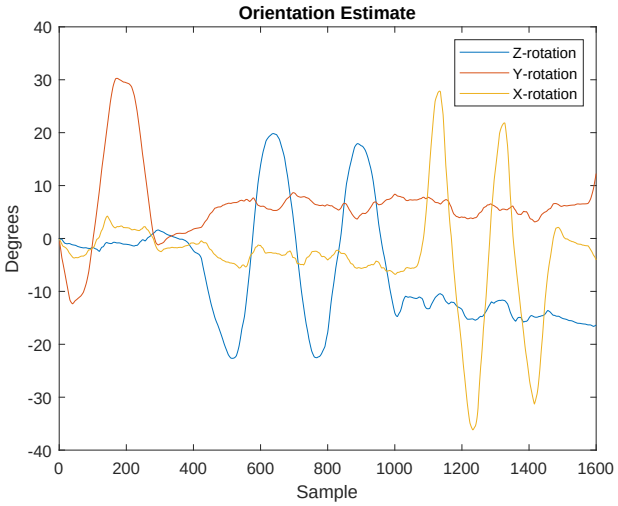
5.3 Simulation Results



(a) MATLAB's default complementary filter



(b) Complementary Filter without correction



(c) Complementary Filter with correction Algebraic Quaternion Algorithm

Figure 1: Implementation of various complementary filter

As can be seen in the figures, the estimation is somewhat accurate without correction but with correction the estimation is highly accurate with some differences possibly due to different constants used.

6 Conclusion

6.1 Summary

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. Algebraic quaternion algorithm. URL: `https://ahrs.readthedocs.io/en/latest/filters/aqua.html`.

[3] AHRS Mario Garcia. Complementary filter. URL: https://ahrs.readthedocs.io/en/latest/filters/complementary.html#ahrs.filters.complementary.Complementary.

[4] Inc. The MathWorks. Estimate orientation with a complementary filter and imu data. URL: https://in.mathworks.com/help/fusion/ug/estimate-orientation-with-a-complementary-filter-and-imu-data.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.

[6] Roberto G. Valenti, Ivan Dryanovski, and Jizhong Xiao. Keeping a good attitude: A quaternion-based orientation filter for imus and margs. *Sensors*, 15(8):19302–19330, 2015. URL: https://www.mdpi.com/1424-8220/15/8/19302.