EE 615
Experiment No. 4

Sensor Fusion Extended Kalman Filter

Lab report submitted by

Group 12
Drishtant Jain: 24M1085
Gagan G: 24M1090

April 15, 2025

Under the guidance of

Prof. Debasattam Pal
Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Contents

1 Introduction

Orientation estimation is a fundamental task in modern control and robotics systems, especially for aerial and mobile platforms that rely on accurate pose information for navigation and control. Raw sensor data from gyroscopes and accelerometers are typically noisy and subject to drift, making it unsuitable for direct use in critical decision-making or control tasks.

To address these limitations, sensor fusion techniques are employed to combine information from multiple sensors in a statistically optimal manner. One of the most widely adopted methods for real-time sensor fusion is the **Extended Kalman Filter (EKF)**. EKF is a recursive Bayesian estimator used for nonlinear systems, making it suitable for processing rotational motion described via quaternions.

This experiment explores the implementation of quaternion-based EKF for estimating orientation using gyroscope and accelerometer measurements. The performance of the filter is evaluated against MATLAB's inbuilt complementary filter using real sensor data.

2 Aim

The aim of this experiment is to implement and evaluate a **Quaternion-based Extended Kalman Filter (EKF)** for fusing accelerometer and gyroscope sensor data to accurately estimate the orientation of a system in 3D space.

3 Background

Precise orientation tracking is essential for aerial vehicles, mobile robots, and wearable devices. Sensors like accelerometers and gyroscopes are commonly used for this purpose:

- **Gyroscopes** provide angular velocity with high temporal resolution but suffer from drift over time due to bias integration.
- **Accelerometers** offer a stable reference aligned with gravity but are sensitive to vibrations and external forces.

To overcome the limitations of individual sensors, the Extended Kalman Filter algorithm is applied to fuse their data. EKF recursively estimates the system's state by alternating between **prediction** (using gyroscope data) and **correction** (using accelerometer data), even when the underlying models are nonlinear. The quaternion representation is chosen for orientation due to its computational efficiency and immunity to singularities (such as gimbal lock).

This sensor fusion pipeline enables robust, real-time estimation of 3D orientation that is more accurate and stable than using either sensor alone.

4 Capturing Orientation as a Quaternion

In three-dimensional space, orientation of one frame with respect to another can be effectively represented using a unit quaternion. A quaternion is a four-dimensional vector

defined as:

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ e_x \sin\left(\frac{\alpha}{2}\right) \\ e_y \sin\left(\frac{\alpha}{2}\right) \\ e_z \sin\left(\frac{\alpha}{2}\right) \end{bmatrix}$$

where:

- α is the rotation angle.
- $\mathbf{e} = [e_x, e_y, e_z]^T$ is the unit vector along the axis of rotation.
- q_w is the scalar part and q_x, q_y, q_z are the vector parts.

Quaternion Conjugate and Inverse

The conjugate of a quaternion $\mathbf{q}$ is given by:

$$\mathbf{q}^* = \begin{bmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{bmatrix}$$

If $\mathbf{q}$ is a unit quaternion, then its conjugate is also its inverse:

$$\mathbf{q}^{-1} = \mathbf{q}^*$$

Quaternion Multiplication

Given two quaternions $\mathbf{p}$ and $\mathbf{q}$, their product $\mathbf{p} \otimes \mathbf{q}$ is given by:

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \end{bmatrix}$$

This operation is non-commutative, i.e., $\mathbf{p} \otimes \mathbf{q} \neq \mathbf{q} \otimes \mathbf{p}$.

Rotating a Vector using a Quaternion

Let $\mathbf{v} = [v_x, v_y, v_z]^T$ be a 3D vector. Its representation as a *pure quaternion* is:

$$\mathbf{v}_q = \begin{bmatrix} 0 \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

The rotation of $\mathbf{v}$ using quaternion $\mathbf{q}$ is computed as:

$$\mathbf{v}_{\text{rot}} = \mathbf{q} \otimes \mathbf{v}_q \otimes \mathbf{q}^*$$

Conversion to Rotation Matrix (DCM)

The quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ can also be converted to a rotation matrix $R(\mathbf{q}) \in SO(3)$ as:

$$R(\mathbf{q}) = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\ 2(q_x q_y - q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z + q_w q_x) \\ 2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

This matrix can be used to rotate 3D vectors and transform between coordinate frames.

5 Sensor Models

In this experiment, we work with two types of sensors commonly found in Inertial Measurement Units (IMUs): the **gyroscope** and the **accelerometer**. These sensors provide raw data that must be filtered and fused due to their inherent imperfections such as drift, noise, and bias.

5.1 Gyroscope Sensor Model

A gyroscope measures the angular velocity $\boldsymbol{\omega}_k$ of the body frame with respect to the inertial frame. However, the measured signal $y_{G,k}$ is affected by sensor bias b_k and additive noise $v_{G,k}$, and is modeled as:

$$y_{G,k} = \boldsymbol{\omega}_k + b_k + v_{G,k} \tag{1}$$

Here:

- b_k is the zero-rate bias that typically evolves slowly over time.
- $v_{G,k}$ is zero-mean Gaussian white noise.

To capture the evolving nature of the bias, we model it as a first-order random walk process:

$$b_k = b_{k-1} + w_{b,k} \tag{2}$$

where $w_{b,k}$ is Gaussian noise representing the drift in bias. This makes the bias part of the state vector to be estimated by the filter.

The true angular velocity can then be estimated from the measurements as:

$$\hat{\boldsymbol{\omega}}_k = y_{G,k} - \hat{b}_k \tag{3}$$

5.2 Accelerometer Sensor Model

The accelerometer measures the net specific force experienced by the device, which includes both linear acceleration and the gravity vector. The accelerometer output is modeled as:

$$y_{A,k} = \mathbf{a}_k - \mathbf{g}_k + v_{A,k} \tag{4}$$

where:

- $\mathbf{a}_k$ is the actual linear acceleration of the body.
- $\mathbf{g}_k$ is the gravitational acceleration vector.
- $v_{A,k}$ is zero-mean Gaussian noise from the sensor.

In static or quasi-static conditions, $\mathbf{a}_k \approx 0$, and thus:

$$y_{A,k} \approx -\mathbf{g}_k \tag{5}$$

This makes the accelerometer useful for correcting the orientation drift caused by the gyroscope over time. However, during rapid motion, the accelerometer readings may deviate significantly due to inertial forces.

Sensor Fusion Motivation

Since the gyroscope suffers from long-term drift and the accelerometer is noisy during dynamic motion, neither sensor alone is sufficient for robust orientation tracking. By fusing these two sensor streams using an Extended Kalman Filter, we obtain an accurate and drift-free orientation estimate.

6 Quaternion-Based Extended Kalman Filter

The Extended Kalman Filter (EKF) is a nonlinear extension of the classical Kalman Filter that linearizes the system dynamics and observation model around the current state estimate. In this experiment, we use the EKF to estimate the orientation of a body in 3D space using quaternion representation.

The state vector $\mathbf{x}_t$ at time t consists of a unit quaternion $\mathbf{q}_t$ representing orientation:

$$\mathbf{x}_t = \mathbf{q}_t = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T \tag{6}$$

The control input to the system is the angular velocity vector $\boldsymbol{\omega}_t = [\omega_x, \omega_y, \omega_z]^T$, obtained from the gyroscope.

6.1 Initialization

The filter is initialized with:

- Initial orientation: $\mathbf{q}_0 = [1, 0, 0, 0]^T$
- Initial state covariance: $P_0 = I_4$
- Process noise variance: σ_ω^2
- Measurement noise variance: σ_a^2

Optionally, $\mathbf{q}_0$ may also be initialized based on the initial accelerometer readings, by converting them to a gravity-based orientation.

6.2 Prediction Step

In the prediction step, we propagate the quaternion forward in time using the angular velocity measurements. The quaternion update is computed using the first-order approximation:

$$\hat{\mathbf{q}}_t = \left(I_4 + \frac{\Delta t}{2} \Omega(\boldsymbol{\omega}_t) \right) \mathbf{q}_{t-1} \tag{7}$$

where Δt is the sampling interval, and $\Omega(\boldsymbol{\omega})$ is the quaternion multiplication matrix defined as:

$$\Omega(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

The process noise covariance is given by:

$$Q_t = \sigma_\omega^2 \, W_t W_t^T \tag{8}$$

where W_t is defined as:

$$W_t = \frac{\Delta t}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix}$$

The state covariance is propagated using the linearized state transition matrix F_t:

$$P_t^- = F_t P_{t-1} F_t^T + Q_t \tag{9}$$

where $F_t = I_4 + \frac{\Delta t}{2} \Omega(\boldsymbol{\omega}_t)$.

6.3 Correction Step

In the correction step, we use the accelerometer to correct for drift in the quaternion estimate. First, we normalize the accelerometer measurement:

$$\mathbf{z}_t = \frac{\mathbf{a}_t}{\|\mathbf{a}_t\|} \tag{10}$$

Assuming gravity in the ENU frame is $\mathbf{g} = [0, 0, 1]^T$, the expected measurement is computed by rotating gravity into the body frame using the estimated quaternion:

$$\hat{\mathbf{z}}_t = C(\hat{\mathbf{q}}_t)^T \mathbf{g} \tag{11}$$

Here $C(\mathbf{q})$ is the Direction Cosine Matrix (DCM) derived from the quaternion.
The measurement residual is:

$$\mathbf{v}_t = \mathbf{z}_t - \hat{\mathbf{z}}_t \tag{12}$$

The observation model Jacobian H_t is derived by linearizing the rotation matrix $C(\mathbf{q})$ with respect to the quaternion components.

The Kalman gain is:

$$K_t = P_t^- H_t^T \left(H_t P_t^- H_t^T + R \right)^{-1} \tag{13}$$

where $R = \sigma_a^2 I_3$ is the measurement noise covariance.
The state estimate and covariance are updated as:

$$\mathbf{q}_t = \hat{\mathbf{q}}_t + K_t \mathbf{v}_t, \quad \mathbf{q}_t \leftarrow \frac{\mathbf{q}_t}{\|\mathbf{q}_t\|} \tag{14}$$

$$P_t = (I_4 - K_t H_t) P_t^- \tag{15}$$

This concludes one full EKF iteration for quaternion-based orientation tracking.

7 Implementation

This section describes the complete MATLAB implementation of the Quaternion-based Extended Kalman Filter (EKF) for orientation estimation using data from an MPU6050 IMU interfaced via an Arduino Mega.

1. Interfacing the IMU with Arduino Mega

The MPU6050 sensor is connected using I2C protocol. A sampling rate of 20 Hz is set:

Listing 1: Sensor Initialization and Arduino Setup

```
clc;
clear all;
a = arduino('COM8', 'Mega2560', 'Libraries', 'I2C');
fs = 20; % Sample Rate in Hz
imu = mpu6050(a,'SampleRate',fs,'OutputFormat','matrix');
```

2. Real-Time Data Collection and Fusion Loop

Accelerometer and gyroscope readings are recorded in real time, and both MATLAB's 'imufilter' and our custom EKF are used for orientation estimation. Visualization is shown live using a 3D viewer.

Listing 2: Sensor Data Collection and EKF Loop

```
while i < N
    [accelReadings, gyroReadings] = read(imu);
    accelR = [accelR; accelReadings];
    gyroR = [gyroR; gyroReadings];

    % MATLAB's filter
    fuse = imufilter('SampleRate', fs);
    orientationMatlab = fuse(accelR, gyroR);

    % EKF Initialization
    countSamples = size(accelR, 1);
    [current_attitude, P_mat, variance_gyro, variance_accel] =
       ↪ initialize(accelR(1,:));
    orientation = quaternion.zeros(countSamples, 1);

    for idx = 1:countSamples
        [current_attitude,P_mat] = prediction(current_attitude,
           ↪ gyroR(idx,:), P_mat, variance_gyro, delta_t);
        [current_attitude,P_mat] = correction(current_attitude,
           ↪ accelR(idx,:), P_mat, variance_accel);
        orientation(idx, :) = current_attitude;
    end

    viewer(orientation(end)); % Live 3D orientation viewer
end
```

3. EKF Initialization Function

Listing 3: Initialization Function

```
function [q_0, P_0, variance_gyro, variance_accel] =
    ↪ initialize(accel)
    q_0 = quaternion(1,0,0,0);  % Identity quaternion
    P_0 = eye(4);               % Initial state covariance
    variance_gyro = 0.3^2;
    variance_accel = 0.5^2;
end
```

4. Prediction Step

Listing 4: Prediction Step

```
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
```

5. Correction Step

Listing 5: Correction Step

```
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]';
    a_hat = C_mat' * g;

    H_mat = 2 * [-g(3)*qy, g(3)*qz, -g(3)*qw, g(3)*qx;
                  g(3)*qx, g(3)*qw, g(3)*qz, g(3)*qy;
                  0, -2*g(3)*qx, -2*g(3)*qy, 0];

    R_mat = variance_accel * eye(3);
    v_t = z_t - a_hat;
    S_t = H_mat * P_mat * H_mat' + R_mat;
    K_t = P_mat * H_mat' / S_t;

    q_t = quat + quaternion((K_t * v_t)');
    q_t = normalize(sign(parts(q_t)) * q_t); % Prevent sign flips
    P_t = (eye(4) - K_t * H_mat) * P_mat;
end
```

8 Simulation Results

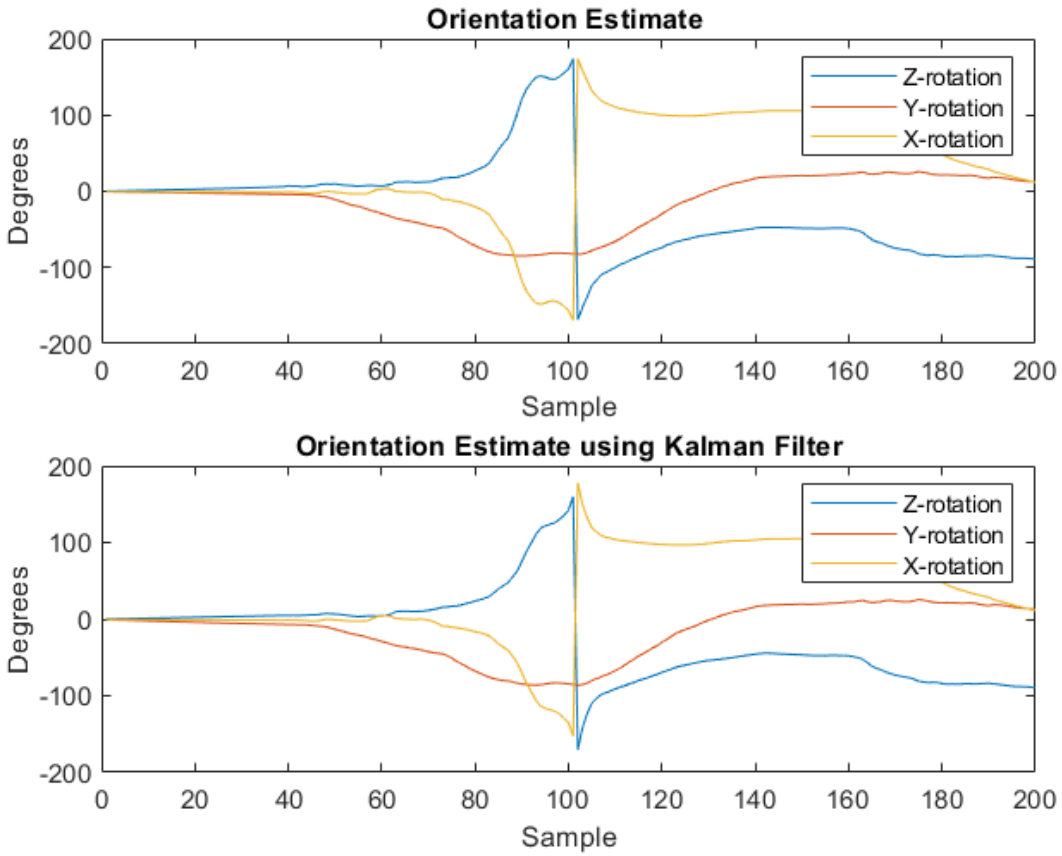

Figure 1: Orientation Estimate in Degrees

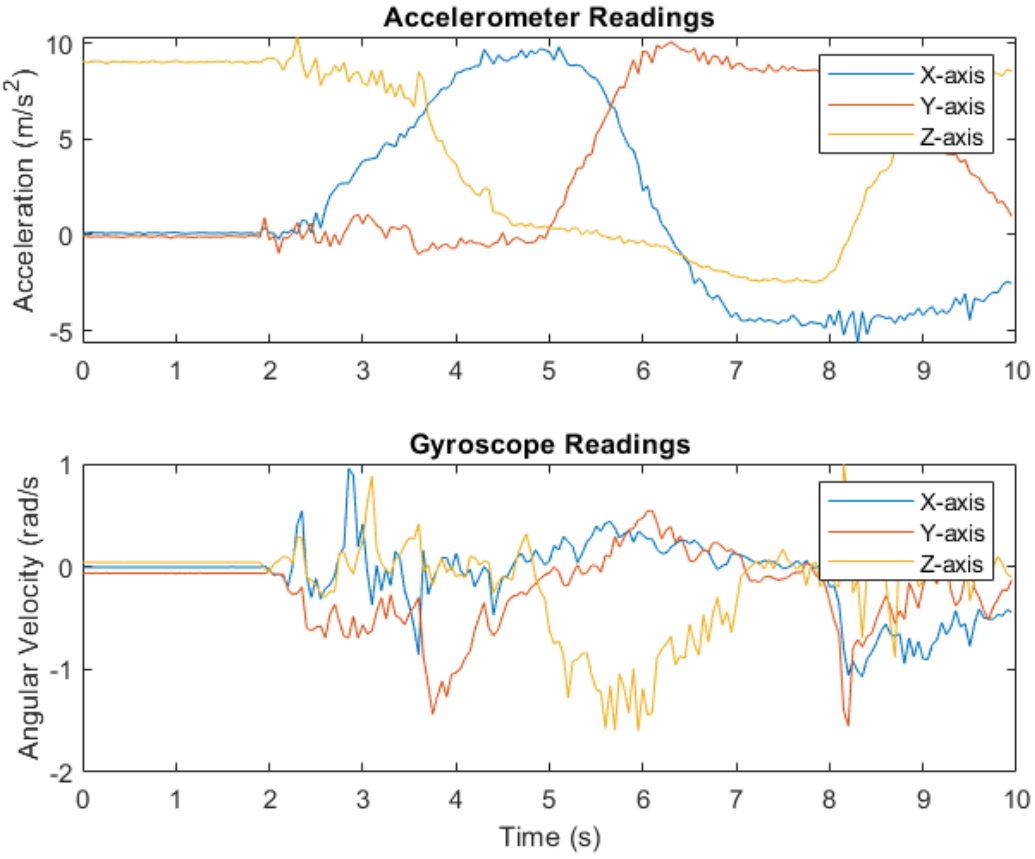

Figure 2: Accelerometer and Gyroscope Readings

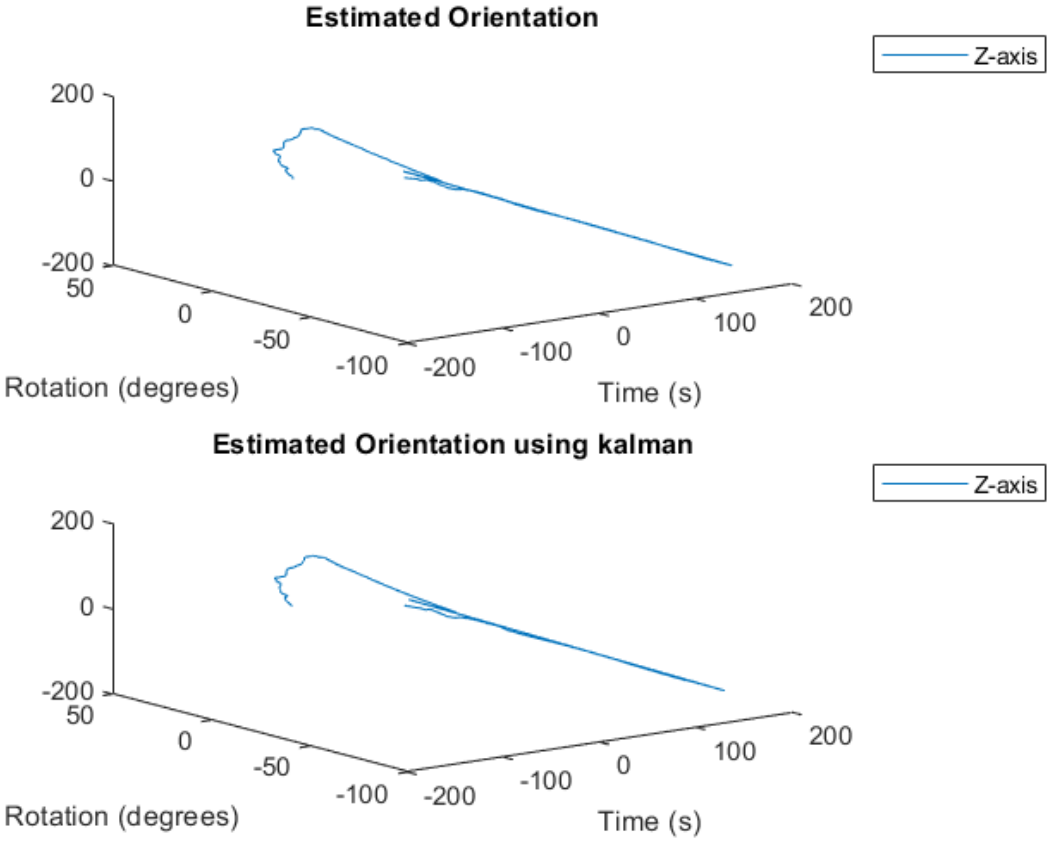

Figure 3: Orientation Estimate

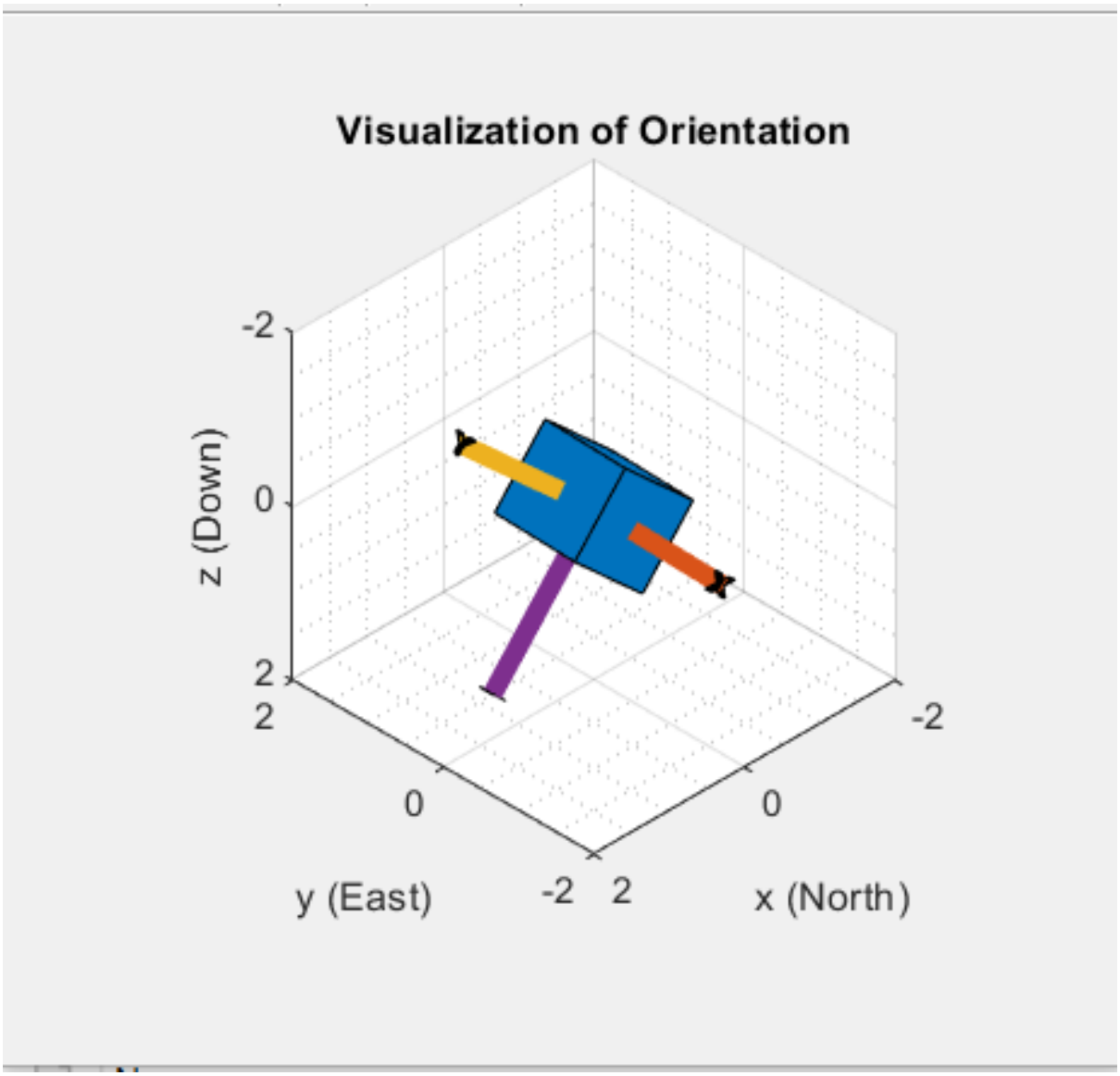

Figure 4: Visualisation of Orientation

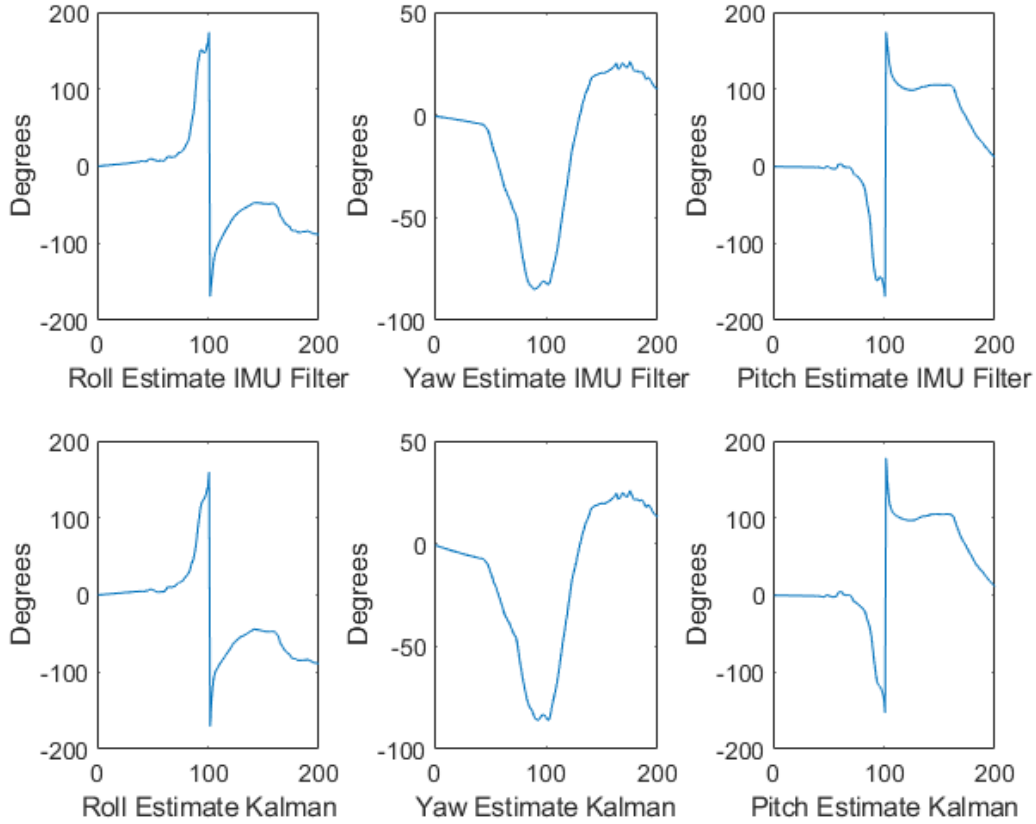

Figure 5: Orientation Estimate with Angles Separated

9 Challenges Faced and Solutions

During the course of implementing and testing the quaternion-based Extended Kalman Filter, we encountered a few technical and conceptual challenges. The following table summarizes the key issues and how we addressed them:

- **1. Initialization Drift in Orientation:**
  In some runs, the filter produced unstable estimates initially due to improper normalization of sensor values or poorly conditioned initial quaternions.

  Solution: We initialized the quaternion as the identity quaternion $[1, 0, 0, 0]$ and ensured all accelerometer inputs were normalized at every timestep.

- **2. Sensor Noise and Real-Time Delay:**
  The raw IMU data was noisy and caused sudden fluctuations in orientation estimation.

  Solution: We tuned the variance parameters for the gyroscope and accelerometer noise in the EKF and applied smoothing on input data. MATLAB's 'imufilter' was used as a benchmark.

- **3. Quaternion Sign Flipping:**
  Quaternion values can flip sign without affecting orientation, but it caused apparent discontinuities in plotting.

  Solution: We used the sign correction trick: $q \leftarrow \text{sign}(\text{parts}(q)) \cdot q$ and renormalized the quaternion after each update.

- **4. Live Plotting Latency:**
  Real-time visualization using the orientation viewer occasionally lagged or failed to update.

  Solution: We reduced the number of samples and limited the update to the latest quaternion, improving visualization speed.

10 Conclusion

This experiment successfully demonstrated the implementation of a quaternion-based Extended Kalman Filter for sensor fusion using accelerometer and gyroscope data. The EKF approach allowed for accurate, drift-compensated orientation tracking in real time.

By comparing the results with MATLAB's 'imufilter', we validated that the EKF achieved consistent and stable orientation estimates, with only minor deviations attributed to noise filtering differences. The use of quaternion representation ensured smooth interpolation and avoided gimbal lock, making it highly suitable for 3D motion tracking.

This exercise provided valuable insights into nonlinear state estimation, sensor modeling, and real-time filtering strategies, which are fundamental in control systems, robotics, and inertial navigation.