Extended kalman filter for Sensor fusion

Contents

1 Introduction **1**
1.1 Implementation 2

2 Metrics conversion **2**

3 6-DOF angle estimation **3**
3.1 Calculations 3
3.2 Problems 3

4 9-DOF configuration for position **3**

5 Code **4**
5.1 Main Function 4
5.2 Data Extraction 5
5.3 Angles 5
5.3.1 Orientation 5
5.3.2 Kalman filter 5
5.4 Acceleration 7
5.5 Velocity 7

1 Introduction

An extended kalman filter is used to estimate the state of a discrete time controlled process. The posterior state is computed by the prior state and the weighted difference. There are two steps that are taken the predictive step and the update step.
Equation (1) shows the predictive step

$$\boldsymbol{x}_{k|k-1} = f(\boldsymbol{x}_{k-1|k-1}, \boldsymbol{u}_{k-1}), \tag{1a}$$
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_{k-1}. \tag{1b}$$

The f is a function of all the state variables. The predictive state estimate is done by Equation(1a) and the predictive covariance estimate is done by Equation(1b) where $\boldsymbol{P}_{k-1|k-1}$ is the predictive estimate at the previous time step. The Jacobian of the state transition matrix $\boldsymbol{F}_k$ is used to linearize the non linear equations using partial differentiation with respect to the state variables. The covariance matrix $\boldsymbol{Q}_{k-1}$ of the process noises which remains the same throughout the simulation and can be adjusted to give a more continuous and smooth function.

Equation (2) Shows the update state

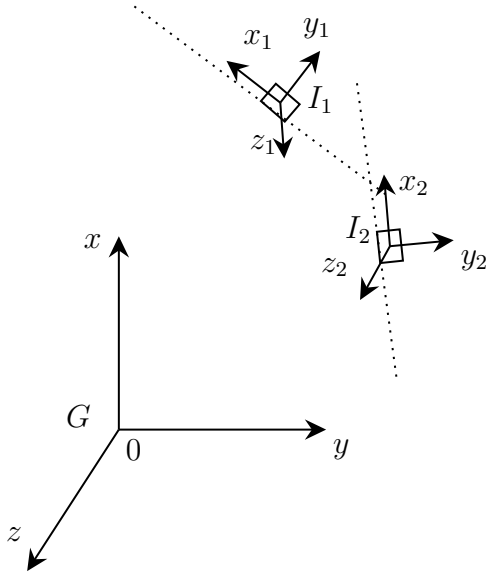
$$\boldsymbol{y}_k = \boldsymbol{z}_k - h(\boldsymbol{x}_{k|k-1}), \tag{2a}$$
$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k, \tag{2b}$$
$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1}, \tag{2c}$$
$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k, \tag{2d}$$
$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1}. \tag{2e}$$

Here we have the equations for the update step. The error $\boldsymbol{y}_k$ in eq (2a) is calculated by finding the difference between the actual measured data at time k with the predicted data from the predictive step. $h(x)$ is a function of all the measurements that are taken from the sensors. Hence there are two separate functions f which is for the state variables that are calculated and $h(x)$ which is function containing all the state that are taken from the sensors. Two functions are used as the states that are provided by the sensors can be different from the states that are required for the predictive step. The Jacobian $\boldsymbol{H}_k$ is the Jacobian $h(x)$ function with respect to the state variables that are measured. The process noises for the measurement $\boldsymbol{R}_k$ in eq (2b), accounts for uncertainty in the sensor or measured readings. The near optimal predictive gain $\boldsymbol{K}_k$ in eq (2c) is calculated using the predictive covariance estimate. This is done by calculating $\boldsymbol{S}_k$ in eq (2b) which captures the total uncertainty between the actual measurement z_k and the predicted measurement and is the covariance matrix at time step k. In equations (2d) and (2e) the prediction is updated with respect to the same time step k and not the previous time step $k-1$ and $\boldsymbol{x}_{k|k}$ is updated incorporating the new measurements at time k. The uncertainty in the

state after incorporating the measurements is represented by $\boldsymbol{P}_{k|k}$. The values predicted for time step k with respect to the time step $k-1$ is represented by $k|k-1$. This does not incorporate the measurement data. The outcome at time step k with respect to the predicted estimate of time step k and incorporating the measurement data is represented by $k|k$.
The prediction and update processes operate in a continuous loop as long as the model is running. Each updated step serves as the foundation for making predictions in the subsequent step, ensuring a seamless and iterative estimation process.

1.1 Implementation

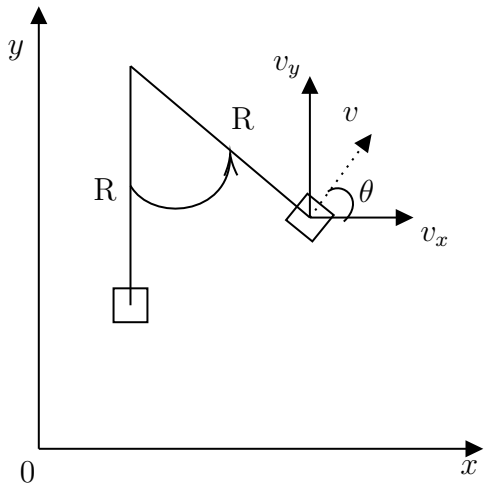
All Inertial Measurement Unit (IMU) sensors have their own initial frames (I_1, I_2), along with a common global frame G that serves as a reference for measuring acceleration and angular velocity. These initial frames are body-fixed frames that move with the IMU sensors, capturing motion relative to their respective local coordinate systems.



An Inertial Measurement Unit (IMU) provides two distinct sets of data: one from the accelerometer and another from the gyroscope. While the accelerometer does not accumulate measurement bias, unlike the gyroscope, it is highly susceptible to noise, particularly in the presence of mechanical vibrations. On the other hand, the gyroscope exhibits significantly lower noise levels over short durations but suffers from bias accumulation over time. By integrating gyroscope measurements to estimate the angle, a process known as dead reckoning, we can achieve accurate short-term orientation tracking. However, this approach leads to a gradual drift in the estimated angle due to the accumulation of bias.

To address this issue, an Extended Kalman Filter (EKF) can be used to integrate data from both sensors, providing a more accurate and stable estimate. Since the accelerometer data is inherently noisy but does not suffer from long-term drift, it can serve as the measurement update in the EKF. Meanwhile, the gyroscope, which provides smooth short-term estimates but accumulates bias over time, can be used as the predictive model. By combining these inputs, the EKF effectively reduces noise and mitigates drift, resulting in a smoother and more reliable estimate that can be used for further analysis.

2 Metrics conversion



The accelerometer provides data in meters per second squared (m/s^2), while the gyroscope measures angular velocity in radians per second (rad/s). To integrate both sensor readings effectively, the gyroscope data must be converted into linear accelerations. This conversion is achieved by transforming the angu-

lar accelerations across all planes into their corresponding linear accelerations. The resulting values are then differentiated with respect to the chosen time step in the measurement data to obtain consistent results.

By substituting R with the distance at which the sensor is placed on the hip, the linear velocity can be computed using the angular velocity from the gyroscope data. This process is applied across all reference frames, and the combined velocity is considered as the final lateral velocity. Differentiating this velocity with respect to time yields the linear accelerations.

3 6-DOF angle estimation

From a 6-DOF IMU sensor, data from the three-axis gyroscope and three-axis accelerometer are used to independently estimate orientation angles, assuming an initial angle of zero. However, gyroscope data is prone to drift over time due to integration errors, while accelerometer data is affected by noise and external forces. To improve accuracy, the individually computed angles are processed through a filtering technique, effectively reducing drift and noise, resulting in a more reliable orientation estimate.[1]

3.1 Calculations

The angle is determined from the gyroscope by integrating the angular velocity over time and adding it to the initial angle.

$$\theta_{\text{gyro}}(t) = \theta_{\text{gyro}}(t-1) + \omega_x \cdot \Delta t, \tag{3a}$$
$$\phi_{\text{gyro}}(t) = \phi_{\text{gyro}}(t-1) + \omega_y \cdot \Delta t, \tag{3b}$$

where $\theta_{\text{gyro}}(0) = 0$ and $\phi_{\text{gyro}}(0) = 0$. And

$$\omega_x = \frac{\text{gyro reading}_x}{\text{sensitivity}}, \tag{4a}$$
$$\omega_y = \frac{\text{gyro reading}_y}{\text{sensitivity}}. \tag{4b}$$

The angle derived from accelerometer data is calculated using gravity-based tilt estimation.

$$\theta_{\text{accel}} = \arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right), \tag{5a}$$
$$\phi_{\text{accel}} = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right), \tag{5b}$$

assuming no external accelerations except gravity. The two angles are compared and combined to obtain the final angle by using complementary filter.

$$\theta = \alpha\theta_{\text{gyro}} + (1-\alpha)\theta_{\text{accel}}, \tag{6a}$$
$$\phi = \alpha\phi_{\text{gyro}} + (1-\alpha)\phi_{\text{accel}}. \tag{6b}$$

3.2 Problems

- The model assumes the absence of external accelerations.
- The model does not account for changes in the yaw axis, which is an unrealistic assumption.
- Implementing a Kalman filter can improve accuracy.
- The model should be extended to incorporate three-axis motion integration.
- There is no acceleration filtering mechanism, nor a method to compute the sensor's linear velocity.

4 9-DOF configuration for position

A magnetometer or compass is a navigation instrument that can identify a specific reference direction in the horizontal plane, allowing horizontal angles to be measured with respect to this direction.

5 Code

The code is modular, with functions dedicated to specific tasks like orientation estimation, filtering, and velocity computation. This structure allows selective use based on application needs, improving clarity and flexibility.

5.1 Main Function

This section handles data collection and runs continuously in the background, serving as the main body that coordinates and calls the required sub-functions. It also includes configurable design parameters that can be adjusted during testing.

```
function imu_kalman_processing()
%% Load IMU Dataset
imu_data = [...
    % Replace this with your 9-column dataset (ax ay az gx gy gz mx my mz)
];

dt = 0.2; % Time step
Q = 0.01 * eye(6); % Parameters for the kalman filter
R = 0.1 * eye(3); % Parameters for the kalman filter
alpha = 0.95; % Parameters for Velocity
beta = 0.1; % Parameters for acceleration

acc_prev = []; % Empty vectors to store acceleration data
vel_prev = []; % Empty vectors to store Velocity data
vel_prev_hp = []; %Empty vectors to store Previous time step velocity data

% Step 1: Process IMU Data
imu = processIMUData(imu_data);

% Step 2: Compute Orientation
[roll_acc, pitch_acc, yaw_mag] = computeOrientationFromIMU(imu);

% Step 3: Kalman Filter Orientation
[filtered_roll, filtered_pitch, filtered_yaw] = kalmanOrientationFilter(...
    roll_acc, pitch_acc, yaw_mag, imu.gyro, dt, Q, R);

% Step 4: Transform Acceleration to Global Frame
[acc_filtered, vel] = transformAccelerationStep(...
    imu.accel, filtered_roll, filtered_pitch, filtered_yaw, acc_prev, vel_prev, beta
    , dt);
% Step 5: High-Pass Filter Velocity
[vel_hp, vel_prev_hp] = highPassFilterVelocityStep(...
    vel, vel_prev, vel_prev_hp, alpha);


% Update state
acc_prev = acc_filtered;
vel_prev = vel;

```

```
end
```

This function serves as the main routine that coordinates and calls all supporting sub-functions.

5.2 Data Extraction

This function extracts and separates the raw sensor input into accelerometer, gyroscope, and magnetometer data for further processing.

```
    %% Data extraction

function imu = processIMUData(data)
    imu.accel = data([3, 2, 1]);            % ax, ay, az
    imu.gyro  = deg2rad(data([6, 5, 4]));   % gx, gy, gz in radians
    imu.mag   = data([7, 8, 9]);            % mx, my, mz

end
```

5.3 Angles

5.3.1 Orientation

This function uses accelerometer and magnetometer data to compute roll, pitch, and yaw, which represent the angle of deflection and serve as measurements for the Kalman filter.

```
%% Orientation
% The accelerometer is used to calculate roll and pitch, while yaw is calculated
    using the magnetometer.
% These values serve as measurement data.
function [roll_acc, pitch_acc, yaw_mag] = computeOrientationFromIMU(imu)

    ax = imu.accel(1); ay = imu.accel(2); az = imu.accel(3);
    mx = imu.mag(1);   my = imu.mag(2);   mz = imu.mag(3);


    roll_acc = atan2(ay, sqrt(ax.^2 + az.^2));
    pitch_acc = atan2(-ax, sqrt(ay.^2 + az.^2));


    roll = roll_acc;
    pitch = pitch_acc;
    mx_c = mx*cos(pitch) + mz*sin(pitch);
    my_c = mx*sin(roll)*sin(pitch) + my*cos(roll) - mz*sin(roll)*cos(pitch);
    yaw_mag = atan2(-my_c, mx_c);

end
```

5.3.2 Kalman filter

The Kalman filter fuses the orientation measurements from the previous step with gyroscope data to produce the final filtered roll, pitch, and yaw values for the sensor.

```
   %% Kalman filter for imu angle

function [filtered_roll_deg, filtered_pitch_deg, filtered_yaw_deg] =
    kalmanOrientationFilter(roll_acc, pitch_acc, yaw_acc, gyro, dt, Q, R)



    X_k = zeros(6,1); P_k = eye(6);


    A = [eye(3), dt*eye(3); zeros(3), eye(3)];
    B = [zeros(3); dt * eye(3)];
    H = [eye(3), zeros(3)];


    u = gyro(:);

    % Prediction step

    X_pred = A * X_k + B * u;
    P_pred = A * P_k * A' + Q;

    % Update step

    Z_k = [roll_acc; pitch_acc; yaw_acc];
    K_k = P_pred * H' / (H * P_pred * H' + R);
    X_k = X_pred + K_k * (Z_k - H * X_pred);
    P_k = (eye(6) - K_k * H) * P_pred;

    roll = X_k(1);
    pitch = X_k(2);
    yaw = X_k(3);

    % Convert to degrees

    filtered_roll_deg = rad2deg(roll);
    filtered_pitch_deg = rad2deg(pitch);
    filtered_yaw_deg = rad2deg(yaw);

end
```

In this implementation, the **filtered pitch angle** is the primary value of interest. The output corresponds to a **single IMU sensor**. For estimating the hip angle, the pitch from this IMU can be used directly. However, **to calculate the knee angle, the pitch angle obtained from the thigh IMU must be subtracted from the pitch angle of the shin IMU**. Therefore, both IMUs must be processed separately, and the resulting pitch values must be subtracted to obtain the relative knee joint angle.For this function to operate correctly, **the data extraction function must be executed beforehand.**

5.4 Acceleration

This function applies a low-pass filter to the accelerometer data to obtain smoothed acceleration values along all three axes.

```
   %% Acceleration

function [acc_filtered, vel] = transformAccelerationStep(acc_body, ...
        roll, pitch, yaw, acc_prev, vel_prev, beta, dt)

    R_bw = angle2dcm(yaw, pitch, roll, 'ZYX');
    acc_body = acc_body(:);  % 3x1
    acc_w = R_bw * acc_body;

    % Gravity compensation
    acc_w(3) = acc_w(3) - 9.81;
    acc_raw = acc_w'; % 1x3

    % Low-pass filter
    if isempty(acc_prev)
        acc_filtered = acc_raw;
    else
        acc_filtered = beta * acc_raw + (1 - beta) * acc_prev;
    end

    % Integrate velocity
    if isempty(vel_prev)
        vel = [0 0 0];
    else
        vel = vel_prev + acc_filtered * dt;
    end
end
```

The forward acceleration corresponds to the acceleration along the **X-axis, which is stored as the first value in the 1×3 acceleration vector**. The other two components, representing lateral and vertical accelerations, are currently disregarded.**This function also plays a crucial role in calculating velocity in the subsequent step by providing the filtered acceleration as input.**For this function to work properly, **the data extraction, orientation computation, and Kalman filter functions must be executed beforehand.**

5.5 Velocity

This function calculates velocity along all three axes, **with the X-axis component being the primary focus**, consistent with the acceleration analysis. The forward velocity can be extracted in the same manner as the X-axis acceleration.

```
    %% Velocity

function [vel_hp, vel_prev_hp] = highPassFilterVelocityStep(...
    vel_curr, vel_prev, vel_prev_hp, alpha)

```

```
    if isempty(vel_prev_hp)
        vel_hp = [0 0 0];
    else
        vel_hp = alpha * (vel_prev_hp + vel_curr - vel_prev);
    end

    vel_prev_hp = vel_hp;
end
```

For this function to work properly, **the data extraction, orientation computation, Kalman filter and acceleration functions must be executed beforehand.**

References

[1] Abdullah Ersan Oğuz and Mustafa Emre Aydemir. A practical implementation of a low-cost 6-dof imu by kalman algorithm. *European Journal of Science and Technology*, 61(4):167–170, 2021.