Extended Kalman Filter

Ali Mohamad Mahmud

Student Number: 018093670A

Course: Connected and Autonomous vehicles

Master in Information and Computer Sciences (MICS)
Université du Luxembourg

Winter Semester 2023-2024

December 21, 2023

1 Introduction

The objective of this assignment is to implement an Extended Kalman Filter (EKF) to estimate the state of a dynamic system using sensor data from IMU, GNSS, and Lidar. The main tasks include predicting motion, updating estimates with new measurements, and computing state covariance. Through this process, we aim to refine the system's position, velocity, and orientation predictions and analyze the results against ground truth data to evaluate the EKF's effectiveness in real-world state estimation scenarios.

2 Motion model

Prediction:

$$\begin{aligned}
\check{\mathbf{x}}_k &= \mathbf{f}_{k-1}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, 0) \\
\mathbf{p}_k &= \mathbf{p}_{k-1} + \Delta t \cdot \mathbf{v}_{k-1} + \frac{1}{2}\Delta t^2 \cdot (\mathbf{C}_{ns} \cdot \mathbf{f}_{k-1} - \mathbf{g}) \\
\mathbf{v}_k &= \mathbf{v}_{k-1} + \Delta t \cdot (\mathbf{C}_{ns} \cdot \mathbf{f}_{k-1} - \mathbf{g}) \\
\mathbf{q}_k &= \Omega(\mathbf{q}(\boldsymbol{\omega}_{k-1}\Delta t))\, \mathbf{q}_{k-1}
\end{aligned}$$

The motion model defined in the equations represents a predictive step in an Extended Kalman Filter (EKF) for estimating the state of a dynamic system. At each timestep k, it predicts the next state $\check{\mathbf{x}}_k$ based on the previous state $\hat{\mathbf{x}}_{k-1}$ and control inputs $\mathbf{u}_{k-1}$. The model updates the position $\mathbf{p}_k$ and velocity $\mathbf{v}_k$ using the specific force $\mathbf{f}_{k-1}$ from IMU data, adjusted by the rotation matrix $\mathbf{C}_{ns}$ and gravity $\mathbf{g}$. The orientation, represented by quaternion $\mathbf{q}_k$, evolves according to the angular velocity $\boldsymbol{\omega}_{k-1}$ over the timestep Δt. This model is crucial for understanding the system's dynamics and refining state estimates with sensor measurements.

Covariance:

$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$$

The covariance update model encapsulated by the equation represents the uncertainty propagation in an Extended Kalman Filter (EKF). At each timestep k, it predicts the next covariance $\check{\mathbf{P}}_k$ based on the previous covariance estimate $\hat{\mathbf{P}}_{k-1}$. The Jacobian of the motion model $\mathbf{F}_{k-1}$ linearizes the nonlinear dynamics around the previous state, projecting the uncertainty into the future. The term $\mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$ incorporates the process noise, modeled by covariance $\mathbf{Q}_{k-1}$, to account for inherent uncertainties and unmodeled dynamics. This step is crucial for maintaining an accurate representation of the system's uncertainty and is fundamental for weighing predictions and sensor measurements appropriately.

3 Measurement model

Kalman Gain:

$$K_k = P_K H_k^T (H_k P_K H_k^T + R_k)^{-1}$$

The Kalman Gain, K_k, dynamically adjusts the state estimate in a Kalman Filter by balancing the predicted state and the new measurement. It's calculated using the current estimate's uncertainty (P_K), the measurement Jacobian (H_k), and the measurement noise (R_k). The formula $K_k = P_K H_k^T (H_k P_K H_k^T + R_k)^{-1}$ reflects how K_k weighs the uncertainties to update the estimate, ensuring optimal correction based on the latest data.

Correction:

$$\hat{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}_k(y_k - h_{k-1}(\check{\mathbf{x}}_k, 0))$$

The state update equation $\hat{\mathbf{x}}k = \check{\mathbf{x}}k + \mathbf{K}k(yk - h_{k-1}(\check{\mathbf{x}}k, 0))$ combines the predicted state $(\check{\mathbf{x}}k)$ with the innovation term, which is the discrepancy between the actual measurement (y_k) and the expected measurement given the prediction. The Kalman Gain $(\mathbf{K}_k)$ scales this discrepancy to correct the prediction, producing the updated state estimate $(\hat{\mathbf{x}}_k)$.

corrected covariance:

$$\check{\mathbf{P}}_k = \mathbf{F}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$$

```
I = np.eye(p_cov_check.shape[0])
    p_cov_hat = (I - K_k @ H_k) @ p_cov_check
```

The corrected covariance $\check{\mathbf{P}}_k$ is updated to reflect the uncertainty in the new state estimate. It's calculated by projecting the prior covariance $\hat{\mathbf{P}}_{k-1}$ forward with the motion model $\mathbf{F}_{k-1}$ and adding the process noise through $\mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T$. The Python implementation further adjusts this projection using the Kalman Gain and the measurement Jacobian to refine the estimate's uncertainty.

4 results

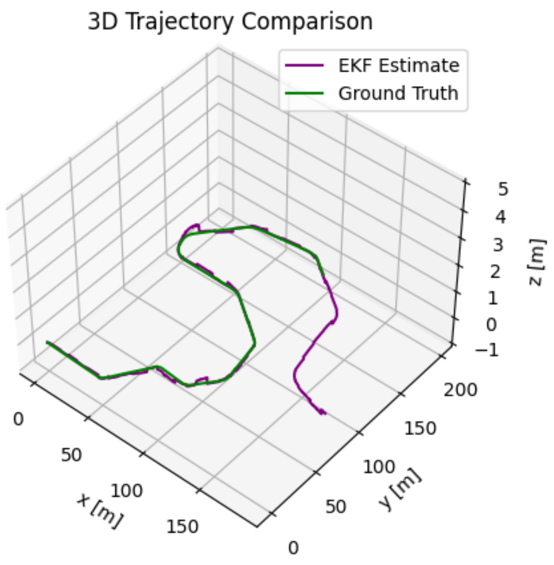

Figure 1: EKF estimation vs Ground Truth

In Figure 1,The purple line represents the EKF estimated trajectory over time, while the green line represents the actual trajectory as provided by ground truth data. The plot assesses the accuracy of the EKF by comparing its estimated trajectory with the actual path. From the plot, we can observe that both trajectories seem to start at similar points where the EKF estimation overshoots a bit more more (around 80m), suggesting that the EKF has a good initial and semi-good final state estimate. However, there are discrepancies between the estimated and actual paths, which are evident in the deviation of the purple line from the green line. These deviations represent the estimation error of the EKF. The plot also shows that both the estimate and ground truth seem to have a non-linear path with changes in direction and altitude (z-coordinate). Overall, the plot suggests the EKF is tracking the trajectory with some level of accuracy but with visible errors that may need further tuning of the EKF parameters or model to reduce.

In Figure 2,in each plot, the blue line indicates the instantaneous error, which is the difference between the EKF estimate and the ground truth. The dashed orange lines represent the three standard deviation (3 Std Dev) bounds for the estimate, providing a measure of the uncertainty in the EKF's estimates. The presence of these bounds allows for an assessment of the confidence in the EKF's accuracy over time.For position (X, Y, Z), the units are in meters, while for orientation (Roll, Pitch, Yaw), the units are in radians. The X and Y plots show a fluctuating error with instances where the error exceeds the 3 Std Dev bounds, indicating moments when the EKF's performance deviates significantly from the ground truth. The Up plot, however, shows a more constrained error, with most values within the 3 Std Dev bounds. The orientation plots (Roll, Pitch, Yaw) exhibit a similar pattern, with errors that oscillate over time and sometimes exceed the uncertainty bounds. This indicates that while the EKF provides a reasonable estimate of the orientation, there are periods of greater uncertainty, possibly due to abrupt movements or sensor noise.

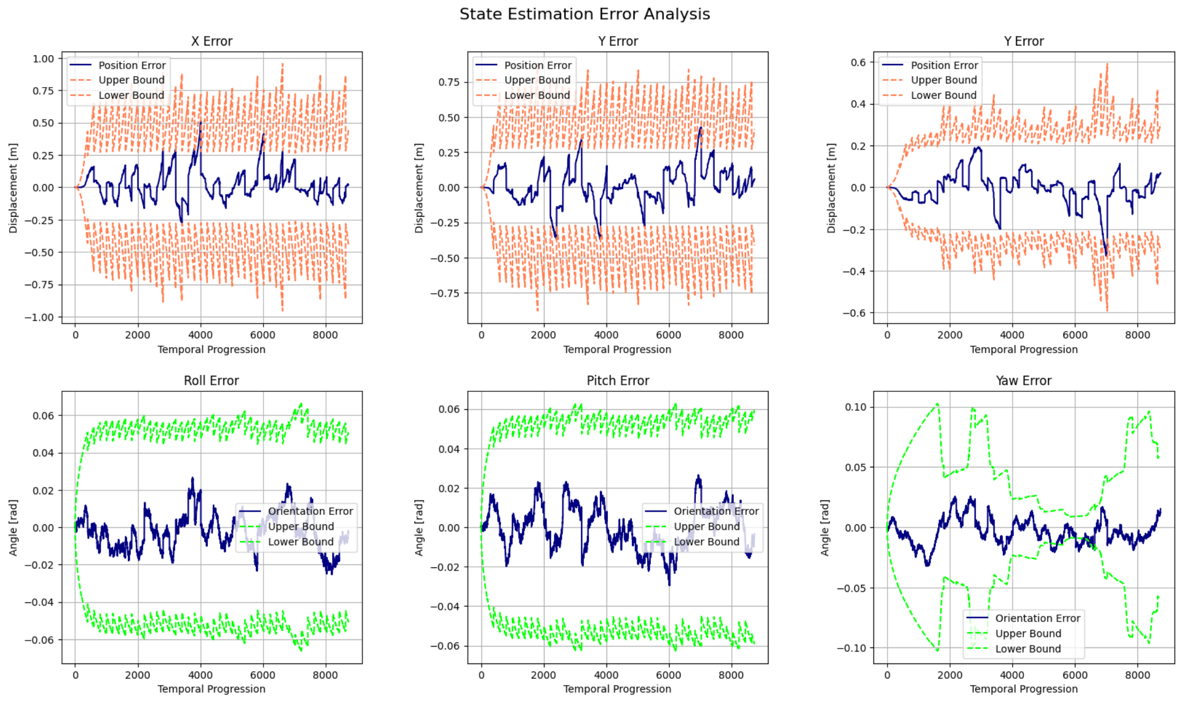

Figure 2: Error Analysis

5 Conclusion

The Extended Kalman Filter (EKF) implemented in this study demonstrates the efficacy of state estimation in dynamic systems using sensor data. Despite certain deviations between the estimated trajectory and ground truth, the EKF's predictions for position and orientation offer a foundational accuracy that can be refined through further model adjustments and parameter tuning. The error analysis reveals instances of significant divergence, underscoring the need for enhanced processing or additional data to improve reliability. Future work could focus on addressing the observed discrepancies, possibly incorporating machine learning techniques for pattern recognition and error reduction, to enhance the EKF's precision and applicability in real-world autonomous navigation systems.

6 Annex(Code)

6.1 Prediction

```python
delta_t = imu_f.t[k] - imu_f.t[k - 1]

# Rotation matrix from navigation frame to sensor frame
C_ns = Quaternion(*q_est[k - 1]).to_mat()

# Predict motion model
f_k_minus_1 = imu_f.data[k - 1]
p_check = p_est[k - 1] + delta_t * v_est[k - 1] + ((delta_t ** 2)/2) * (np.dot(C_ns,
    f_k_minus_1) + g)
v_check = v_est[k - 1] + delta_t * (np.dot(C_ns,f_k_minus_1) + g)
# Update quaternion using angular velocity
axis_angle = imu_w.data[k - 1] * delta_t
quaternion_1 = Quaternion(axis_angle=axis_angle)
q_check = quaternion_1.quat_mult_right(q_est[k - 1], out='Quaternion')

# f Jacobian
f_jac = np.eye(9)
f_jac[0:3, 3:6] = delta_t * np.eye(3)  # Position update part
# Jacobian for the quaternion part
f_jac[3:6, 6:9] = -skew_symmetric(np.dot(C_ns,f_k_minus_1))*delta_t

# Process noise covariance matrix
Q_k_minus_1 = block_diag(delta_t**2 * var_imu_f * np.eye(3), delta_t**2 * var_imu_w *
    np.eye(3))

p_cov_check= f_jac @ p_cov[k - 1] @ f_jac.T + l_jac @ Q_k_minus_1 @ l_jac.T
```

6.2 Kalman gain

```python
H_k = h_jac  # Measurement model Jacobian
R_k = np.eye(3) * sensor_var  # Measurement noise covariance
# Compute Kalman Gain
S = H_k @ p_cov_check @ H_k.T + R_k
K_k = p_cov_check @ H_k.T @ np.linalg.inv(S)
```

6.3 correction

```python
x_error = K_k @ (y_k - p_check)
p_hat, v_hat= p_check+x_error[0:3], v_check+x_error[3:6]

axis_angle =  x_error[6:9]
quat_from_ag = Quaternion(axis_angle=axis_angle)
q_hat = quat_from_ag.quat_mult_left(q_check, out='Quaternion')
I = np.eye(p_cov_check.shape[0])
p_cov_hat = (I - K_k @ H_k) @ p_cov_check
```