NYU TANDON SCHOOL OF ENGINEERING

PROJECT REPORT

Robot Localization and Navigation

Submitted by:

Name: Harshavardhan Vibhandik
NET ID: hsv2015
NYU ID: N13023471

Spring 2024
Submitted on: March 15, 2024

ROB-GY6213 Robot Localization and Navigation

Instructor: Prof. Giuseppe Loianno

Introduction

In the project 1 we have to implement the Extended Kalman Filter to estimate the state by using the data from Vicon and IMU sensors, where the IMU provides body frame acceleration and angular velocity, and the Vicon provides the position and velocity data. Vicon data is stored in variables ‘time and ‘vicon.’

We have to deploy the Extended Kalman Filter to estimate the position, velocity, orientation, and sensor biases of the Micro Aerial Vehicle. In part one, we need to use position and orientation from Vicon, and in part two, we need to use only velocity from Vicon.

Objectives

1. To implement an Extended Kalman Filter to estimate the state using the sensor data from Vicon and IMU sensors.
2. To estimate an MAV's position, velocity, orientation, and sensor biases.
3. To edit the given skeleton code to deploy the EKF using the prediction and update steps.

Methodology

The Kalman Filter implementation in the provided file involves several steps to estimate the state of a system using sensor data. The Kalman Filter script loads a dataset according to the specified dataset number. Then, it initializes the initial state (`uPrev`) by extracting required data from the Vicon data and sets the initial covariance matrix (`covarPrev`) as an identity matrix. In the main loop of the script, it iterates over each timestamp in the dataset. For each timestamp, it extracts acceleration and angular velocity data from the sampled IMU data. The time difference (`dt`) between the current and previous timestamps is calculated to determine the time step. Within the loop, the prediction step of the Kalman Filter is executed using the previous state (`uPrev`), covariance (`covarPrev`), and the current angular velocity and acceleration data. This prediction step estimates the next state and covariance of the system.

Subsequently, the current measurement data from the Vicon system is obtained and used in the update step of the Kalman Filter. This step adjusts the predicted state and covariance based on the measurement data. After updating the state and covariance estimates, the current state is stored in the `savedStates` matrix for further analysis or visualization. Finally, the `plotData` function is called to generate plots illustrating the evolution of the estimated states over time.

Explanation & Results

Part I:

i. MainScript:

1. Take the user input for the Dataset number
2. Takes the initial conditions
3. Runs a For loop
   - Which calculates the prediction step from another file

[covarEst,uEst] = pred_step(uPrev,covarPrev,angVel,acc,dt);

   - Which calculates the update step from another file

[uCurr,covar_curr] = upd_step(z_t,covarEst,uEst);

ii. Prediction Step

1. In this step, first, we extract the previous state values
2. Then we calculate the various matrices:
   - Prediction Step

Prediction step:

$\bar{\mu}_t = \mu_{t-1} + \delta t \, f(\mu_{t-1}, u_t, 0)$

$\bar{\Sigma}_t = F_t \, \Sigma_{t-1} \, F_t^T + V_t \, Q_d \, V_t^T$

   - To calculate this step, we need to calculate x_dot values using:

State:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} = \begin{bmatrix} \text{position} \\ \text{orientation} \\ \text{linear velocity} \\ \text{gyroscope bias} \\ \text{accelerometer bias} \end{bmatrix} \in \mathbf{R}^{15}$$

Process model:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_3 \\ G(\mathbf{x}_2)^{-1}(\boldsymbol{\omega}_m - \mathbf{x}_4 - \mathbf{n}_g) \\ \mathbf{g} + R(\mathbf{x}_2)(\mathbf{a}_m - \mathbf{x}_5 - \mathbf{n}_a) \\ \mathbf{n}_{bg} \\ \mathbf{n}_{ba} \end{bmatrix} = f(\mathbf{x}, \mathbf{u}, \mathbf{n})$$

x_dot =

$$\begin{pmatrix} vx \\ vy \\ vz \\ -\frac{\cos(yaw)\,(bgx + ngx - wmx)}{\cos(pitch)} - \frac{\sin(yaw)\,(bgy + ngy - wmy)}{\cos(pitch)} \\ \sin(yaw)\,(bgx + ngx - wmx) - \cos(yaw)\,(bgy + ngy - wmy) \\ wmz - ngz - bgz - \frac{\cos(yaw)\sin(pitch)\,(bgx + ngx - wmx)}{\cos(pitch)} - \frac{\sin(pitch)\sin(yaw)\,(bgy + ngy - wmy)}{\cos(pitch)} \\ (\cos(roll)\sin(yaw) - \cos(yaw)\sin(pitch)\sin(roll))\,(bay - amy + nay) - (\sin(roll)\sin(yaw) + \cos(roll)\cos(yaw)\sin(pitch))\,(baz - amz + naz) - \cos(pitch)\cos(yaw)\,(bax - amx + nax) \\ (\cos(yaw)\sin(roll) - \cos(roll)\sin(pitch)\sin(yaw))\,(baz - amz + naz) - (\cos(roll)\cos(yaw) + \sin(pitch)\sin(roll)\sin(yaw))\,(bay - amy + nay) - \cos(pitch)\sin(yaw)\,(bax - amx + nax) \\ \sin(pitch)\,(bax - amx + nax) - \cos(pitch)\sin(roll)\,(bay - amy + nay) - \cos(pitch)\cos(roll)\,(baz - amz + naz) - \frac{981}{100} \\ nbgx \\ nbgy \\ nbgz \\ nbax \\ nbay \\ nbaz \end{pmatrix}$$

x_dot =

[vel_x;
vel_y;
vel_z;
((sin(yaw)*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)))/cos(pitch) - (cos(yaw)*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch)))/cos(pitch))*(bias_gyro_z - wmz) - ((sin(yaw)*(cos(roll)*cos(yaw) + sin(pitch)*sin(roll)*sin(yaw)))/cos(pitch) - (cos(yaw)*(cos(roll)*sin(yaw) - cos(yaw)*sin(pitch)*sin(roll)))/cos(pitch))*(bias_gyro_y - wmy) - (bias_gyro_x - wmx)*(cos(yaw)^2 + sin(yaw)^2);
(bias_gyro_z - wmz)*(cos(yaw)*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)) + sin(yaw)*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch))) - (bias_gyro_y - wmy)*(cos(yaw)*(cos(roll)*cos(yaw) + sin(pitch)*sin(roll)*sin(yaw)) + sin(yaw)*(cos(roll)*sin(yaw) - cos(yaw)*sin(pitch)*sin(roll)));
- (bias_gyro_z - wmz)*(cos(pitch)*cos(roll) + (cos(yaw)*sin(pitch)*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch)))/cos(pitch) - (sin(pitch)*sin(yaw)*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)))/cos(pitch)) - (bias_gyro_y - wmy)*(cos(pitch)*sin(roll) - (cos(yaw)*sin(pitch)*(cos(roll)*sin(yaw) - cos(yaw)*sin(pitch)*sin(roll)))/cos(pitch) + (sin(pitch)*sin(yaw)*(cos(roll)*cos(yaw) + sin(pitch)*sin(roll)*sin(yaw)))/cos(pitch)) - (bias_gyro_x - wmx)*(sin(pitch)*cos(yaw)^2 + sin(pitch)*sin(yaw)^2 - sin(pitch));
(amz - bias_acc_z)*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch)) - (amy - bias_acc_y)*(cos(roll)*sin(yaw) - cos(yaw)*sin(pitch)*sin(roll)) + cos(pitch)*cos(yaw)*(amx - bias_acc_x);
(amy - bias_acc_y)*(cos(roll)*cos(yaw) + sin(pitch)*sin(roll)*sin(yaw)) - (amz - bias_acc_z)*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)) + cos(pitch)*sin(yaw)*(amx - bias_acc_x);
cos(pitch)*cos(roll)*(amz - bias_acc_z) - sin(pitch)*(amx - bias_acc_x) + cos(pitch)*sin(roll)*(amy - bias_acc_y) - (981/100);
(0);
(0);
(0);
(0);
(0);
(0)]

First, we extract relevant components from the previous state `uPrev`, including linear velocity (`vel_x`, `vel_y`, `vel_z`) and Euler angles (`roll`, `pitch`, `yaw`). We also extract gyroscope biases (`bias_gyro_x`, `bias_gyro_y`, `bias_gyroz`) and accelerometer biases (`bias_acc_x`, `bias_acc_y`, `bias_acc_z`).
Then, we utilize the extracted values to calculate the derivative of the state vector, denoted as `x_dot`. This involves correcting the angular velocity from the world frame to the body frame, incorporating gyroscope bias and noise, and adjusting accelerometer measurements for biases and noise.
After computing the state derivative, we use it to predict the next state (`uEst`) by integrating over a small time interval `dt`.
Additionally, we linearize the system by computing the Jacobian matrix `A`, which represents the rate of change of the state with respect to itself, and `F`, which is the state transition matrix.
Finally, we update the covariance matrix `covarEst` to account for the uncertainty in our predictions.

Linearization & Discretization

- $A_t = \frac{\partial f}{\partial x}\Big|_{\mu_{t-1}, u_t, 0}$
- $U_t = \frac{\partial f}{\partial n}\Big|_{\mu_{t-1}, u_t, 0}$

Linearization

- $F_t = I + \delta t \, A_t$
- $V_t = U_t$
- $Q_d = Q \, \delta t$

Discretization

iii. Update State

The update step is given using the function z = g(x,v), where x is the Prev state. It is calculated using

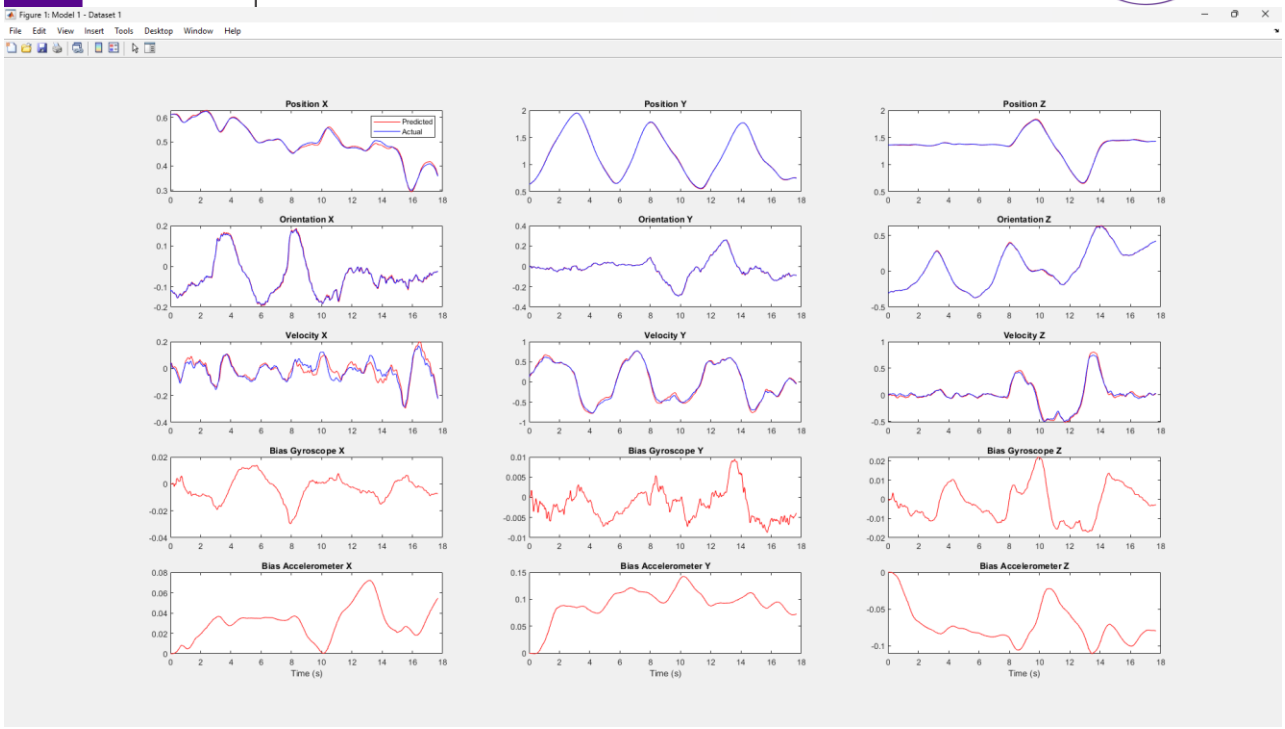
Update step:

- $\mu_t = \bar{\mu}_t + K_t \, (z_t - g(\bar{\mu}_t, 0))$
- $\Sigma_t = \bar{\Sigma}_t - K_t \, C_t \, \bar{\Sigma}_t$
- $K_t = \bar{\Sigma}_t \, C_t^T \, (C_t \, \bar{\Sigma}_t \, C_t^T + W_t \, R \, W_t^T)^{-1}$

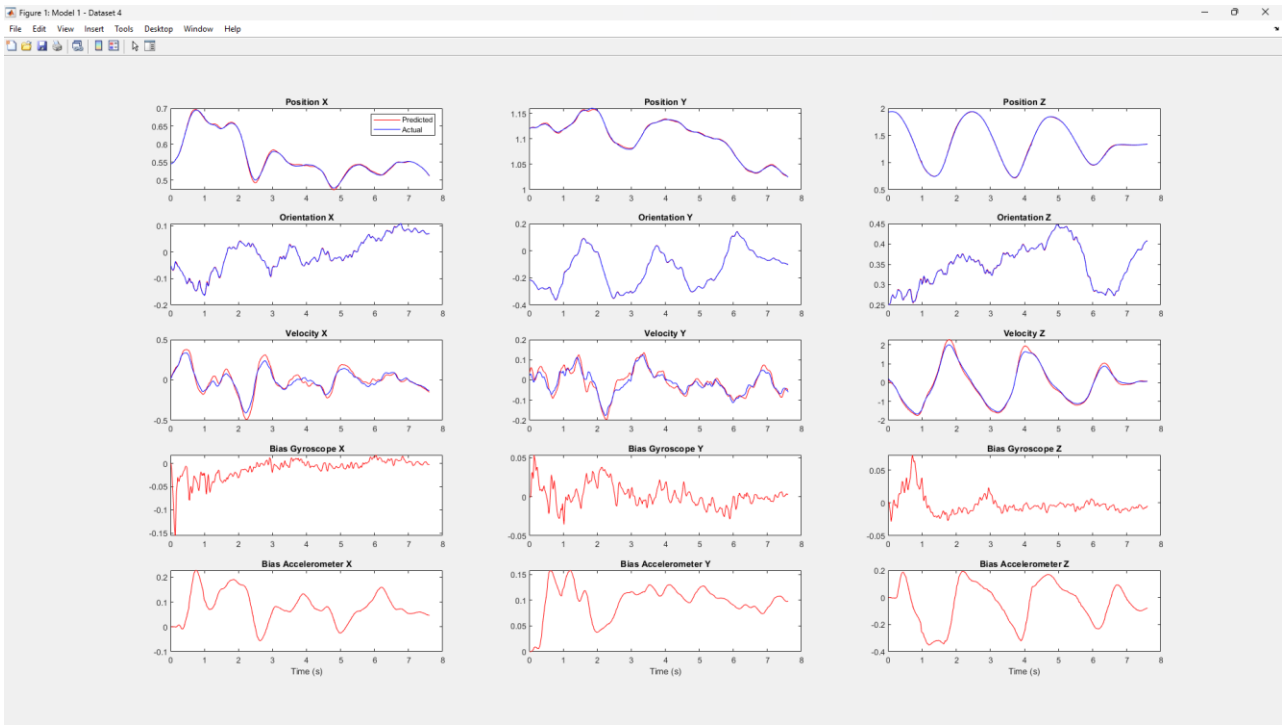
$$\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \end{bmatrix} + \mathbf{v} = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} + \mathbf{v} = C\,\mathbf{x} + \mathbf{v}$$

iv. Results

Dataset number: 1



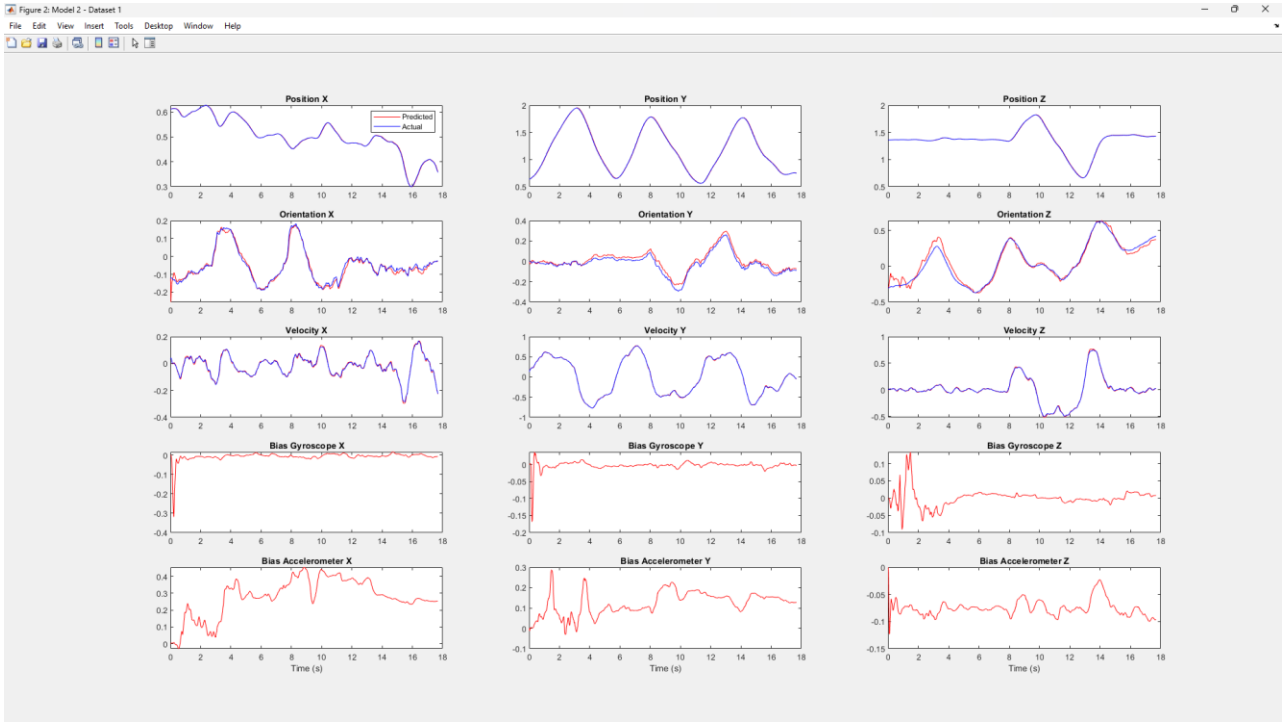
Dataset number: 4



Part II:

For Part 2, we implement the Extended Kalman filter to estimate the state by updating only the measurement of velocity from the vicon. We implement the prediction step exactly in the way as we did in the Part-1

Dataset number: 1



Dataset number: 4