Rallapalli Dinesh Sai, Naidu

N19602446
NetID: Dr3696

ROBOT LOCALIZATION AN NAVIGATION

Project – 1

Masters in Mechatronics and Robotics Engineering, Department of Mechanical and Aerospace Engineering, Tandon School of Engineering

ROB-GY_6213_1_A **Spring'24 - (03/15/2020)**

Instructor: Prof. Giuseppe Loianno

Statement: It is time to put together everything that you've learned in this course until today! In this phase, you'll use Vicon and IMU for estimation. For this phase of the project, you will implement an Extended Kalman Filter (EKF) for state estimation. You will use the body frame acceleration and angular velocity from the onboard IMU as your control inputs. The measurement will be given by the pose or velocity from the Vicon. The body frame of the robot is coincident with the IMU frame.

Rallapalli Dinesh Sai, Naidu

N19602446
NetID: Dr3696

1. Overview and Sensor Data

Overview

It is time to put together everything that you've learned in this course until today! In this phase, you'll use Vicon and IMU for estimation. For this phase of the project, you will implement a Extended Kalman Filter (EKF) for state estimation. You will use the body frame acceleration and angular velocity from the onboard IMU as your control inputs. The measurement will be given by the pose or velocity from the Vicon. The body frame of the robot is coincident with the IMU frame.

Sensor Data

The data for each trial is provided in a mat file. The mat file also contains Vicon data. The Vicon data is stored in two matrix variables, time and Vicon. The time variable contains the timestamp while the Vicon variable contains the Vicon data in the following format:

$$[x\ y\ z \text{ roll pitch yaw } vx\ vy\ vz\ \omega x\ \omega y\ \omega z]\ T$$

The on-board processor of the robot collects synchronized camera and IMU data and sends them to the desktop computer. At this stage, the camera data should not be used. The sensor data is decoded into standard MATLAB format. Note that since the sensor data is transmitted via wireless network, there may or may not be a sensor packet available during a specific iteration of the control loop.

2. The Extended Kalman Filter

The Extended Kalman Filter (EKF) is a variant of the Kalman Filter, which is used for state estimation in dynamic systems. It's particularly useful when the system dynamics are nonlinear, as it employs linearization techniques to handle nonlinearity.

Overview of the Extended Kalman Filter:

1. Prediction Step:
   - State Prediction: Predict the next state of the system based on the current state estimate and the system dynamics model.

- State Covariance Prediction: Predict the covariance of the state estimate accounting for process noise.

2. Update Step:
- Measurement Prediction: Predict the expected measurement based on the predicted state.
- Innovation Covariance: Calculate the innovation covariance, which represents the uncertainty in the predicted measurement.
- Kalman Gain: Compute the Kalman Gain, which determines how much the predicted state should be corrected based on the difference between the predicted and actual measurements.
- State Update: Update the state estimate using the Kalman Gain and the difference between the predicted and actual measurements.
- Covariance Update: Update the covariance matrix to incorporate the information from the measurement.

The key difference between the EKF and the standard Kalman Filter lies in the linearization step. In the EKF, the system dynamics and measurement models are locally linearized around the current state estimate, enabling the use of the traditional Kalman Filter equations. The implementation you provided earlier appears to be the prediction step of an Extended Kalman Filter. It predicts the next state and updates the covariance matrix based on the system dynamics and process noise. Typically, after the prediction step, there would be an update step where the predicted state is corrected based on the actual measurements.

Overall, the Extended Kalman Filter is widely used in various applications such as navigation, robotics, and control systems, where nonlinear dynamics are common. However, it's worth noting that the EKF has limitations, especially in highly nonlinear systems where linearization errors can significantly affect performance.

3. Code Explanation and Plots

Part 1

Kalman Filter Code (Logic in for loop):

Extracting relevant data and performing the prediction and update steps of the Kalman filter for each timestamp.

1. **Iteration over Time:**

- The loop iterates over each timestamp in the 'sampledTime' vector.

2. **Data Extraction:**
   - 'angVel' and 'acc' are extracted from the 'sampledData' structure at index 'i'. These represent the angular velocity and acceleration at the current time step, respectively.
   - 'currTime' is retrieved from the 'sampledTime' vector at index 'i'.
3. **Time Difference Calculation:**
   - 'dt' is calculated as the time difference between the current time ('sampledTime(i)') and the previous time ('prevTime').
4. **Previous Time Update:**
   - 'prevTime' is updated to the current time ('currTime') for the next iteration.
5. **Measurement Extraction:**
   - 'z_t' is extracted from the 'Z' matrix at column index 'i'. This represents the measurement vector at the current time step.
6. **Prediction Step:**
   - 'pred_step' function is called with parameters 'uPrev', 'covarPrev', 'angVel', 'acc', and 'dt'. This function predicts the next state based on the previous state, system dynamics, and control inputs. It returns the covariance estimate ('covarEst') and mean estimate ('uEst') for the predicted state.
7. **Update Step:**
   - 'upd_step' function is called with parameters 'z_t', 'covarEst', and 'uEst'. This function updates the state estimate based on the measurement and predicted state. It returns the updated state estimate ('uCurr') and updated covariance ('covar_curr').
8. **State and Covariance Update:**
   - 'uPrev' is updated to'uCurr' for the next iteration.
   - 'covarPrev' is updated to 'covar_curr' for the next iteration.
9. **State Saving:**
   - The current mean estimate ('uCurr') is saved into the 'savedStates' matrix at column index 'i'.

Prediction Step:

This `pred_step` function appears to implement the prediction step of a Kalman filter. Here's a breakdown of what it does:

1. **Input Parameters:**
   - 'uPrev': Previous mean state vector.
   - 'covarPrev': Previous covariance matrix.
   - 'angVel': Angular velocity.
   - 'acc': Acceleration.
   - 'dt': Sampling time.
2. **State Extraction:**
   - Extracts the individual elements of the previous mean state vector 'uPrev'.
3. **State Dynamics:**
   - Computes the derivatives of the state variables ('x_dot') based on the given state equations and input values.
4. **State Prediction:**

- Updates the state estimate ('uEst') based on the derivatives of the state variables and the sampling time.

5. **State Transition Matrix:**
   - Constructs the state transition matrix ('tf') based on the computed derivatives.
6. **State Transition Matrix Derivative:**
   - Constructs the derivative of the state transition matrix ('ta') based on the computed derivatives.
7. **Control Matrix:**
   - Constructs the control matrix ('tu') based on the computed derivatives.
8. **Process Noise Covariance Matrix:**
   - Constructs the process noise covariance matrix ('Q') based on the sampling time.
9. **Prediction Update:**
   - Updates the covariance estimate ('covarEst') using the state transition matrix, previous covariance matrix, control matrix, and process noise covariance matrix.
10. **Output:**
    - Returns the updated covariance estimate and mean state estimate.

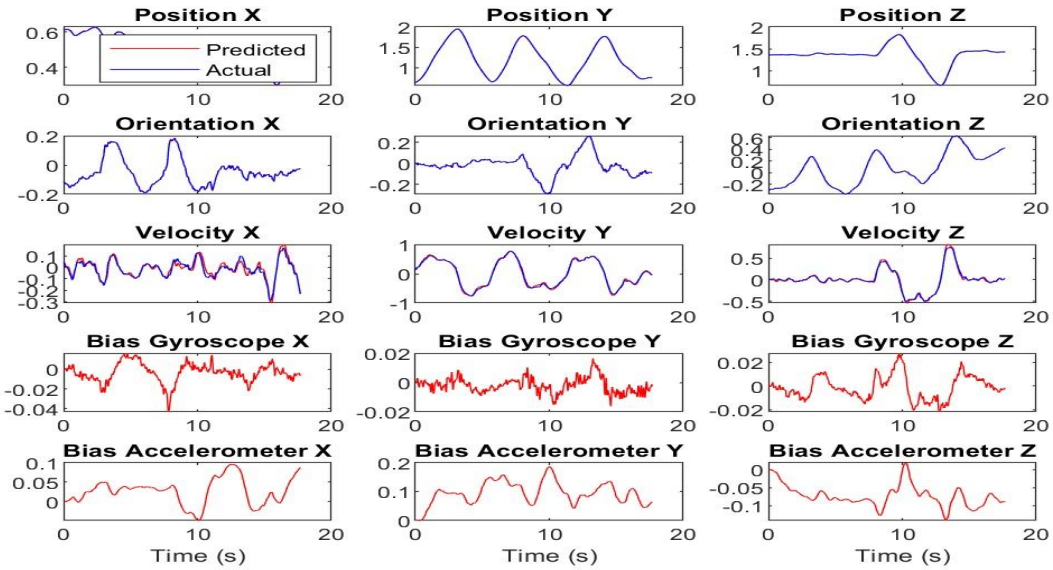
Update Step:

This 'upd_step' function seems to implement the update step of a Kalman filter. Here's a breakdown of what it does:

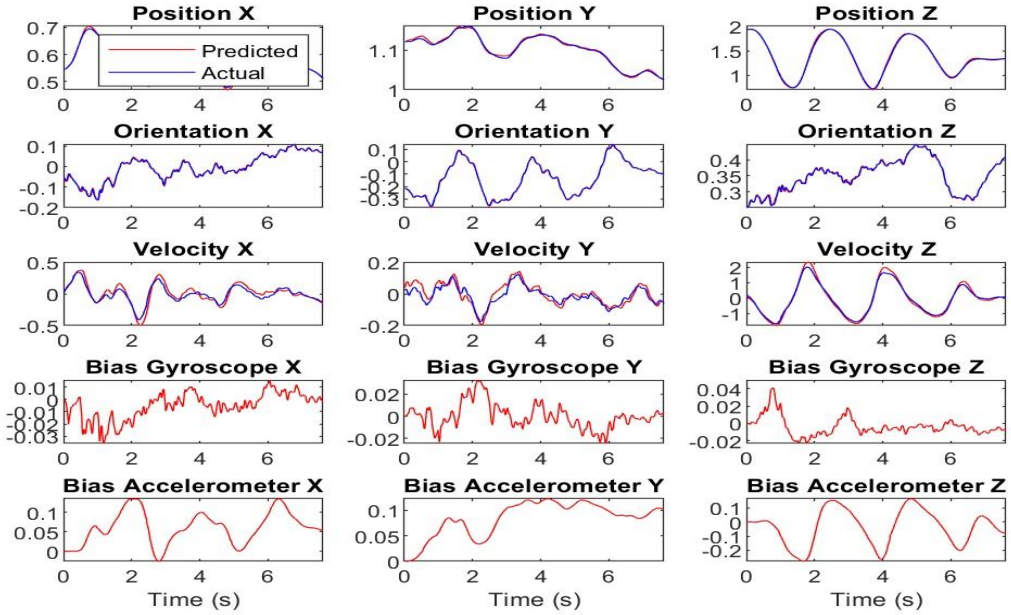
1. **Input Parameters:**
   - 'z_t': Measurement vector.
   - 'covarEst': Predicted covariance matrix.
   - 'uEst': Predicted mean state vector.
2. **Measurement Transformation:**
   - Transforms the predicted state vector 'uEst' into the measurement space using the transformation matrix 'It'. The transformed measurement is denoted as 'z'.
3. **Measurement Covariance:**
   - Constructs the measurement covariance matrix 'Ct'.
4. **Kalman Gain Calculation:**
   - Calculates the Kalman gain 'Kt' based on the predicted covariance matrix, the measurement transformation matrix, and the measurement covariance matrix.
5. **State Update:**
   - Updates the mean state estimate ('uCurr') based on the Kalman gain and the difference between the actual measurement 'z_t' and the predicted measurement 'z'.
6. **Covariance Update:**
   - Updates the covariance matrix ('covar_curr') based on the Kalman gain and the predicted covariance matrix.
7. **Output:**
   - Returns the updated mean state estimate and covariance matrix.

Plot:

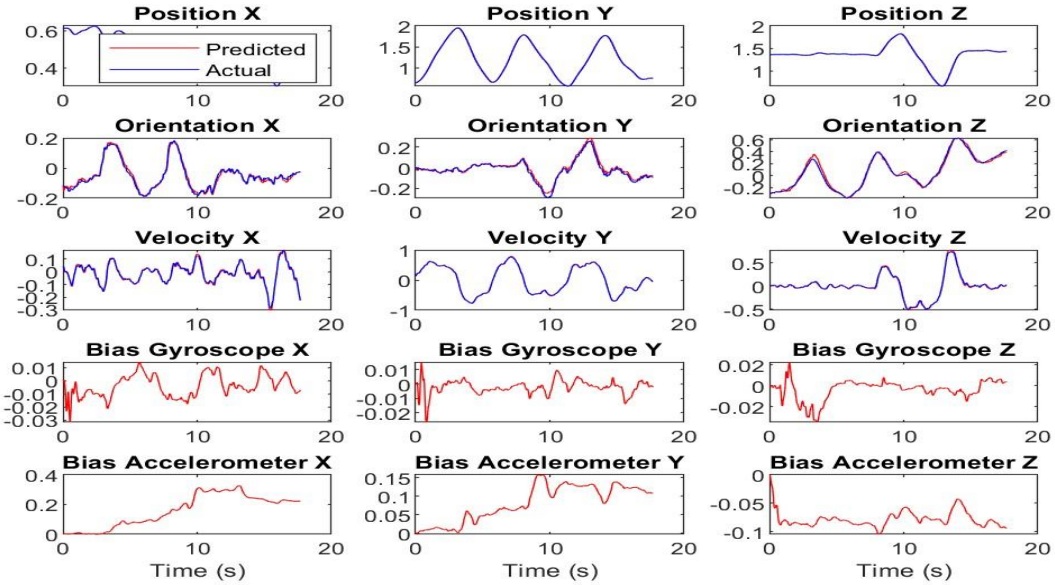
Part-1-Dataset-1



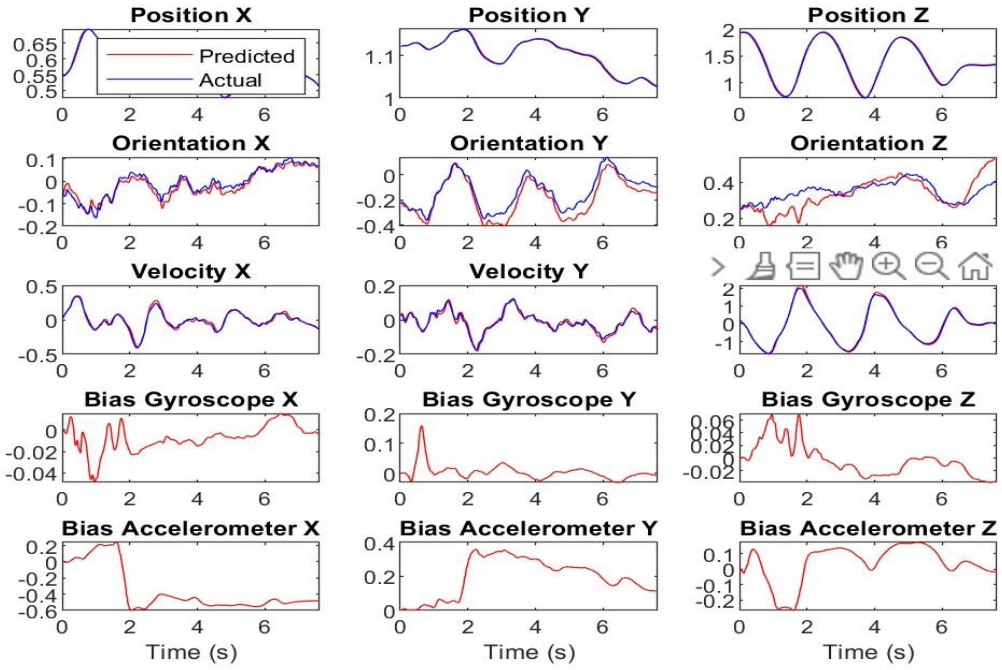
Part-1-Dataset-4



Part-2-Dataset-1



Part-2-Dataset-4



Note:

In Part 2, we employ the Extended Kalman filter to estimate the state by only updating the Vicon's velocity measurement. We carry out the prediction stage precisely as we did in Part 1.