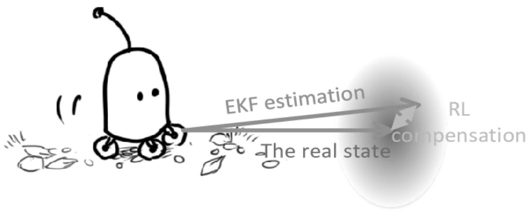
ROBOT LOCALIZATION AND NAVIGATION (ROB-6213) PROJECT 1 REPORT

Rushi Bhavesh Shah (rs7236)

1. Introduction

The Kalman filter was invented in the 1950s by Rudolph Emil Kalman, as a technique for filtering and prediction in linear systems. The Kalman filter implements belief computation for continuous states. It is not applicable to discrete or hybrid state spaces. The Kalman filter represents beliefs by the moments representation: At time t, the belief is represented by the mean μ_t and the covariance Σ_t. The assumptions of linear state transitions and linear measurements with added Gaussian noise are rarely fulfilled in practice. Hence, these assumptions , render plain Kalman filters, inapplicable to all but the most trivial robotics problems.



The extended Kalman filter (EKF) overcomes one of these assumptions: the linearity assumption. Here the assumption is that the next state probability and the measurement probabilities are governed by nonlinear functions. The extended Kalman filter (EKF) calculates an approximation to the true belief. It represents this approximation by a Gaussian. Thus, the EKF inherits from the Kalman filter, but it differs in that this belief is only approximate, not exact as was the case in Kalman filters. (ref. 1)

2. Problem Statement

An Extended Kalman Filter (EKF) is to be implemented to estimate the position, velocity, and orientation, and sensor biases of a Micro Aerial Vehicle. Vicon and IMU data are presented. The Vicon velocity is given in the world frame, whereas the angular rate in the body frame of the robot. The body frame acceleration and angular velocity from the on board IMU is to be used as inputs. Present two filter versions need to be presented. In the first one, the measurement update will be given by the position and orientation from vicon, whereas in the second one, only the velocity from the Vicon will be used. Euler angles convention to be used is ZYX.

3. Solution for Part One

Part one asked to implement EKF for estimating the position and orientation of the Micro Aerial Vehicle and then use the data from vicon to update the predicted values.
The solution starts by identifying the nature of the process model, i.e. *f (x, u, n),* where *x* is the state, *u* is the input control and *n* is the sensor noise present. EKF is implemented because it is real world data and hence the process model is continuous in nature. Hence, for applying EKF and doing the prediction, we'll first linearize the process model and then discreetize it. Further, to do the update step, we'll linearize the observation model and then proceed with the values from the predicted step.

It should be noted that the noise and the bias are considered with zero mean, hence would be null *3* x *1* vectors.

3.1 Prediction Step

ref. 1 : Thrun, S., Burgard, W. and Fox, D. (2006), "Probabilistic Robotics"

We start by defining the state space of the vehicle.

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

It is a 15 X 1 matrix containing all the state variables of the robot. We differentiate the state space matrix to get the process model.

3.1.1 Process model

To calculate the process model, we need to assume that the gyroscope and the accelerometer both give noisy estimates of angular velocity (*w_m*) and linear acceleration (*a_m*). It is also assumed that the drift in the gyroscope and accelerometer biases is described by a Gaussian, white noise process. Hence, the equations look like

Angular Velocity:

$$\boldsymbol{\omega}_m = \boldsymbol{\omega} + \mathbf{b}_a + \mathbf{n}_a$$
$$\dot{\mathbf{b}}_g = \mathbf{n}_{bg}$$
$$\mathbf{n}_{bg} \sim N(0, Q_g)$$

Linear Acceleration:

$$\mathbf{a}_m = R(\mathbf{q})^T(\ddot{\mathbf{p}} - \mathbf{g}) + \mathbf{b}_a + \mathbf{n}_a$$
$$\mathbf{n}_a \sim N(0, Q_a)$$
$$\dot{\mathbf{b}}_a = \mathbf{n}_{ba}$$
$$\mathbf{n}_{ba} \sim N(0, Q_{ba})$$

Further, since we are asked to follow '*ZYX*' Euler angle convention, the matrix to map euler angles to angular velocity *G* will be

$$G = \begin{pmatrix} \cos(\phi)\cos(\theta) & -\sin(\phi) & 0 \\ \cos(\theta)\sin(\phi) & \cos(\phi) & 0 \\ -\sin(\theta) & 0 & 1 \end{pmatrix}$$

where **q** = [$\boldsymbol{\Psi}$, Θ, $\boldsymbol{\phi}$]^T = [roll, pitch, yaw]$^{T.}$ Hence, further, since the angular velocity is given in the body frame, we need to convert the **body from to the world frame**. Hence, we need to obtain the rotation matrix *R* for the given Euler angle representation. Now, by rearranging the terms, we have our process model, *x_dot = f(x, u, n)* as

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

Now, we need to find the predicted mean and the covariance, i.e

$$\bar{\mu}_t = \mu_{t-1} + \delta t\, f(\mu_{t-1}, u_t, 0)$$
$$\bar{\Sigma}_t = F_t\, \Sigma_{t-1}\, F_t^T + V_t\, Q_d\, V_t^T$$

3.1.2. Linearization

To find the predicted mean and the covariance, we need to linearize and discretize the process model by using the Jacobians. Hence, for finding the Jacobians *A_t* and *U_t*, I first defined [*3 x 1*] vectors for all the elements from the state space i.e. *p, q, p_dot, bg* & *ba*. I symbolized every element of every vector, since I wanted to calculate the Jacobians in symbolic form. Next, I defined a matrix *f(xun)* to calculate the process model.

Further, to calculate the Jacobian *A_t*, I used the *Jacobian()* function from Matlab and calculated the Jacobian of *f(xun)* w.r.t *X i.e.* wrt each element from the state space. There, I obtained *A_t* as a *15 x 15* matrix. Similarly, I calculated the Jacobian of *f(xun) w.r.t the* noises and the sensor biases to obtain a *15 x 12* matrix. This I did in a separate script and obtained symbolic Jacobians *A_t* and *U_t* using the results as they were in the prediction function to reduce the runtime of the program.

3.1.3 Discretization

To calculate the predicted covariance, we need F_t, V_t and *Q*. These can be calculated using the equations

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

Where I is a 15×15 identity matrix and Q is 12×12. t is the time difference between the moments when the sensors read the data.

3.1.4 Writing *pred_step* function

The first two input parameters for the *pred_step* function are previous mean & previous covariance for the position, orientation and velocity PDFs. Next input parameters are angular velocity, acceleration and dt i.e same as t. Prediction step returns the estimated covariance and mean for position, orientation and velocity PDFs.

3.2 Update Step

This step takes the predicted values from the predicted step to update it using the measured data from the sensors. For Part one, we'll update only the position and the orientation of the robot by using the data from vicon.

3.2.1 Observation Model

Update step starts by identifying the observation model $g(x_t, v_t)$ where x_t is the predicted state and v_t is the noise with zero mean. Further we need to calculate the Kalman Gain, updated mean and the covariance using the following equations.

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

3.2.2 Linearization

To calculate the Kalman Gain, updated mean and the covariance, we first need to linearize the observation model. We'll do that using the Jacobian as in the prediction step. In this case, we'll calculate the Jacobian wrt to the state X of the robot. Since the observation model is a function of the state only, the Jacobian C_t will be an identity matrix (6 x 6) concatenated with a zero matrix with dimension (6 x 9) making the dimensions 6×15. This is so, because position and orientation constitute the first six rows only. Furthermore, we'll assume Jacobian W_t , i.e. the Jacobian of the observation model wrt noise as identity matrix too. And, the covariance of the noise R, we'll assume to be a 6×6 diagonal matrix with value 10e-5.

3.1.3 Writing *upd_step* function

Further, the above equations are included in the function to calculate the updated mean and covariance of the position and orientation of the robot. The inputs to this function are the observed data for position and orientation from vicon, the predicted mean and the predicted covariance and the outputs are the updated mean and covariance for position and orientation PDFs.

3.3 Main Program

To obtain Extended Kalman Filter, we need to integrate both the steps viz. prediction and the update step. To do that, we first initialize the data files and parse the data to the main script. We take the observed data for the position and the orientation from the vicon. Also, we'll initialize the values of the mean by taking the initial values of the vicon.

While writing the main loop, we run the *for loop* for the number of timestamps available in the dataset. We need to calculate the time-difference i.e. *dt*. We can do this by defining two variables *t2* and *t1*. For the first loop, $t1 = 0$, and the first element in the array will be *t2*. Then we'll calculate $dt = t2 - t1$. Further, we'll extract the values for angular velocity and acceleration from the IMU data, corresponding to the iteration variable. We'll pass values of *uPrev, covarPrev, angVel, acc* and *dt* to the *pred_step* function; from which we'll get *covarEst* and *uEst* i.e. estimated covariance and mean. Then, assign *t2*, the value of *t1*.

Next, for the update step, we'll extract the observed data *z_t* from the vicon data for position and orientation of the robot, again, corresponding to the iteration variable. We'll pass the output from the *pred_step* function as input to the *upd_step* function, along with the observed data *z_t*. Here, we'll get updated or current mean and covariance (*uCurr* & *covar_curr*) as outputs. Then, to continue the loop, we'll do

covarPrev = covar_curr
uPrev = uCurr

and save the updated/current mean in the *savedStates* matrix, which will help in plotting the data.

4.1 Prediction Step

The prediction step for this part will be the same as that for the previous one. It is so because while predicting the values, we parsed velocity of the robot as well, however we just updated the position and the orientation values of the robot.

4.2 Update Step

The structure and the equations involved in this function are the same as in section 3.3. However, since we intend to update the velocity, we need to have *C_t* with dimensions *9* x *15* with a *3 x* 3 identity matrix, with (7,7) and (9,9) as its diagonal coordinates in *C_t*; and rest all as zero. This is how we'll perform computation on the velocity components.

4.3 Main Program

The structure of the main program will be the same as the part1. However, the change would be in the measured data *z_t* that is extracted from the vicon for the update step. This time, we'll extract the velocity data from vicon and use it in the update step.

5. Observation

For part We observe that the position and orientation from part one, when estimated using EKF, we get an almost exact overlap for all three data sets as shown below.

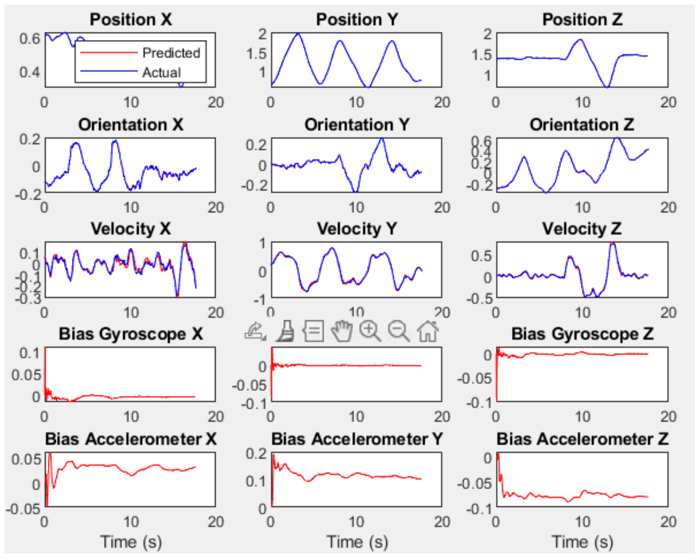


Fig 1: Part 1- Dataset 1

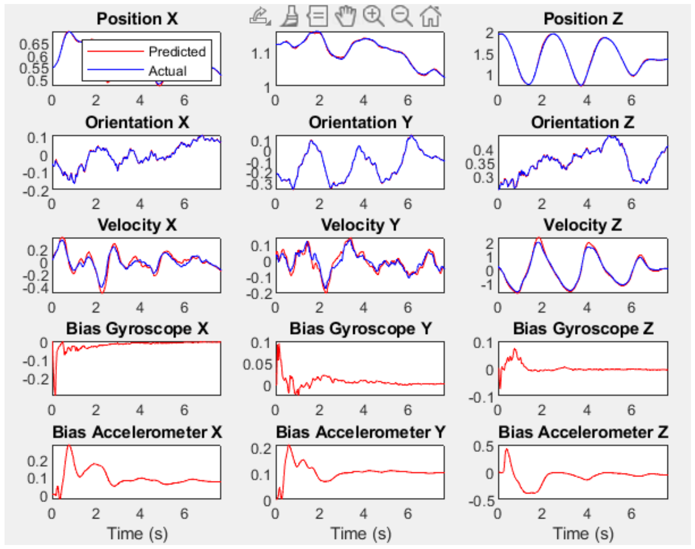


Fig 2: Part 1- Dataset 4

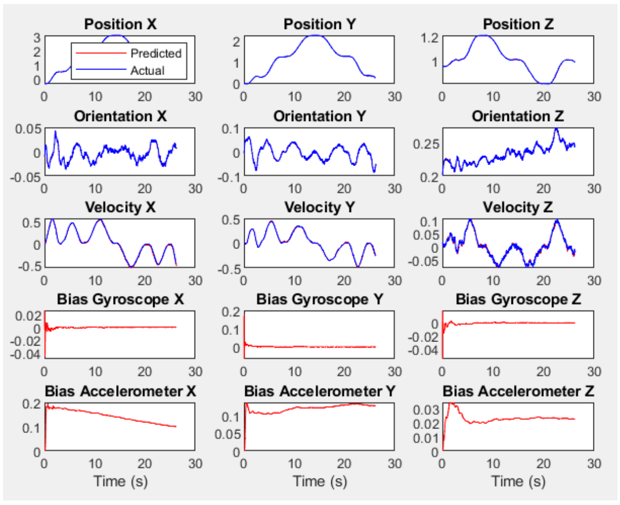


Fig 3: Part 1- Dataset 9

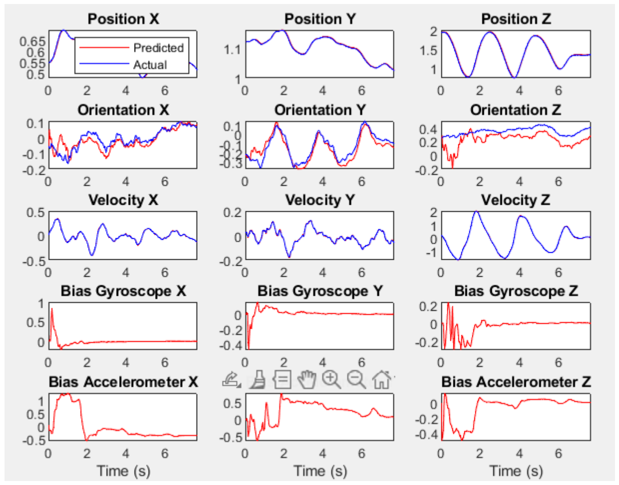


Fig 5: Part 2- Dataset 4

On the other hand, we can observe that the overlap of the curves for part two is seen only for data set 1, that too it's not accurate. However, the overlap is very less for the other two data sets. It can be overlapped by tuning the bias values for the sensors.

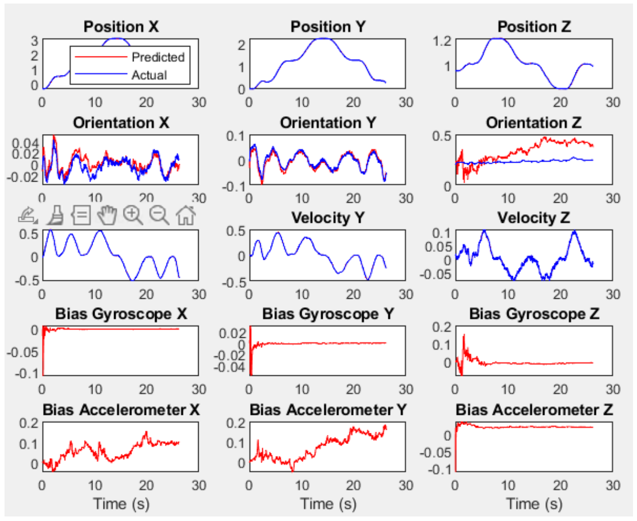


Fig 6: Part 2- Dataset 9

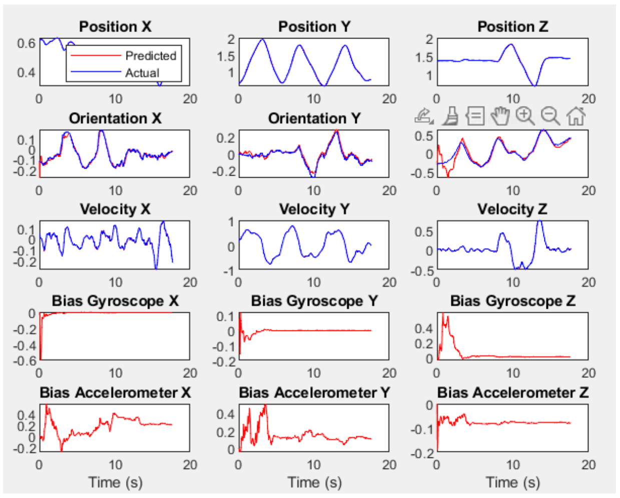


Fig 4: Part 2- Dataset 1

6. Conclusion

EKF is proved to be a very efficient technique for robot localization which repeatedly predicts and updates the state of the robot for a continuous time system as well.