ECE 8540
Analysis of Tracking Systems

Assignment 5
Extended Kalman Filtering

Vivek Koodli Udupa
C12768888

October - 25, 2018

1 Introduction

In this report, we will consider the problem of Extended Kalamn Filtering. In filtering, there are two prominent equations that describe the model that was designed to address a problem. The two equations are observation equation and state transition equation. The observation equation describes the measurements that are obtained through the sensors and the state transition equation describes how the system is expected to change over time. The Kalman filter is used to predict the the future states of the system. But Kalman filter works only with functions that are linear. To solve non-linear functions, Extended Kalman Filter is used.

Extended Kalman Filtering uses Taylor series expansion to get a linear approximation of the non-linear function. Taylor series works by taking a point and performing multiple derivatives at that point. In case of Extended Kalman Filter, mean of the Gaussian is taken on the non-linear curve and numerous derivatives are performed to approximate it. Only the first derivative of the Taylor series is considered to linearize the non-linear curve. For every non-linear function, a tangent is drawn around the mean to try to approximate the function linearly.

This report is focused on tracking something that is moving sinusoidally, like a spool of thread unwinding in a machine. Tracking the position of the spool from which the thread is originating would need a sinusoidal model because the thread position moves up and down as the spool unwinds, which is a sinusoidal motion when plotted over time. This report will include the methods and results of applying Extended Kalman filter to such a sinusoidal function.

2 Methods

Extended Kalman Filter(EKF) is a continuous cycle of predict and update. When formulating the problem for the EKF we consider the following steps:

1. Determine the state variables. Here we are answering the question "what are we tracking?".
2. Write the state transition equations i.e. How things evolve over time.
3. Define the dynamic noise(s). This describes the uncertainties in state transition equation.
4. Determine the observation variables i.e. Sensor readings.
5. Write the observation equations (relating the sensor readings to the state variables).
6. Define the measurement noise(s). These are the uncertainties in observation variables.
7. Characterize the state transition matrix and observation matrix.
8. Check all matrices in the EKF equations to make sure the sizes are appropriate.

2.1 Deriving the equations for EKF sinusoidal model

It is difficult to formulate state transition equations for the sinusoidal problem mentioned in Introduction, because the dependent variable is expected to be time. For example:

$$f(x_t, a_t) = \left[x_t = \sin \tfrac{t}{10} + a_t\right] \tag{1}$$

However, using equation 1, the following state does not depend upon the previous state; it only depends upon the time. This is not a viable formulation for state-space filtering, because there is no uncertainty in the state of the system with respect to time.

However we can modify the state model and state transition equation to get something similar that can be used for filtering:

$$f(x_t, a_t) = \begin{bmatrix} x_{t+1} = x_t + \dot{x}_t T \\ \dot{x}_{t+1} = \dot{x}_t + a_t \\ h_{t+1} = \sin \frac{x_t}{10} \end{bmatrix} \tag{2}$$

In equation 2. variable x represents something like time and $\dot{x}$ along with dynamic noise a_t to represent uncertainty in the propagation of the sinusoid over time. The variable h provides the actual value of the sinusoid.

Using this model, we have three state variables:

$$X_t = \begin{bmatrix} x_t \\ \dot{x}_t \\ h_t \end{bmatrix} \tag{3}$$

The state transition equations 2 and 3 given above, where a_t is a random sample drawn from $N(0, \sigma_a^2)$ representing an uncertainty in the propagation of the sinusoid over time.

For observation, the sensor that detects the current height of the sinusoid, d_t will be considered:

$$Y_t = \begin{bmatrix} d_t \end{bmatrix} \tag{4}$$

For example, this could be accomplished by using a light sensor to detect the thread position as it spins off a spool. The observation equations for this model are:

$$g(x_t, n_t) = \begin{bmatrix} d_t = h_t + n_t \end{bmatrix} \tag{5}$$

where n_t is a random sample drawn from $N(0, \sigma_n^2)$ representing measurement noise.

In order to use this model in the EKF, four Jacobians must be calculated. The derivative of the state transition equations with respect to the state variables is:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial (x, \dot{x}, h)} = \begin{bmatrix} 1 & T & 0 \\ 0 & 1 & 0 \\ \frac{1}{10}\cos\frac{x}{10} & 0 & 0 \end{bmatrix} \tag{6}$$

The derivative of the observation equations with respect to the dynamic noises is:

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial (0, a_t, 0)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$

The derivative of the observation equations with respect to the state variables is:

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial (x, \dot{x}, h)} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The derivative of the observation equations with respect to the measurement noises is:

$$\frac{\partial g}{\partial n} = \frac{\partial g}{\partial (n_t)} = \begin{bmatrix} 1 \end{bmatrix} \tag{9}$$

$$Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma_a^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{10}$$

The covariance of the measurement noises is

$$R = \begin{bmatrix} \sigma_n^2 \end{bmatrix} \tag{11}$$

Equation 1 through 11 theoretically describes the working of the EKF sinusoidal model. The implementation of the same will be described in the following section.

2.2 Implementation

As mentioned in previous section 2.1 the EKF is a continuous predict and update loop and the equations for EKF has been derived. This section describes the implementation of this filter using the equations for the main loop.

1. Predicting the next state:
$$X_{t,t-1} = f(X_{t-1,t-1}, 0) \tag{12}$$
where $f(X_{t-1,t-1}, 0)$ is the approximated state $\tilde{x}_t$.

2. Predicting next state covariance:
$$S_{t,t-1} = \left(\frac{\partial f}{\partial x}\right) S_{t-1,t-1} \left(\frac{\partial f}{\partial x}\right)^T + \left(\frac{\partial f}{\partial a}\right) Q \left(\frac{\partial f}{\partial a}\right)^T \tag{13}$$
where $\left(\frac{\partial f}{\partial x}\right)$ and $\left(\frac{\partial f}{\partial a}\right)$ are the Jacobians of the state transition equations.

3. Obtain measurement(s) Y_t

4. Calculate the Kalman gain (weights)
$$K_t = S_{t,t-1} \left(\frac{\partial g}{\partial x}\right)^T \left[\left(\frac{\partial g}{\partial x}\right) S_{t,t-1} \left(\frac{\partial g}{\partial x}\right)^T + \left(\frac{\partial g}{\partial n}\right) R \left(\frac{\partial g}{\partial n}\right)^T\right]^{-1} \tag{14}$$
where $\left(\frac{\partial g}{\partial x}\right)$ and $\left(\frac{\partial g}{\partial n}\right)$ are the Jacobians of the measurement equations.

5. Update state
$$X_{t,t} = X_{t,t-1} + K_t[Y_t - g(\tilde{x}_t, 0)] \tag{15}$$
where $g(\tilde{x}_t, 0)$ is the ideal (noiseless) measurement of the approximated state from above.

6. update state covariance
$$S_{t,t} = \left[I - K_t \left(\frac{\partial g}{\partial x}\right)\right] S_{t,t-1} \tag{16}$$

7. Now increment the loop iterator t

The implementation of the above described models and equations are done in MATLAB. Please refer appendix for the implementation code.

3 Results

This section consists of plots of EKF predictions against actual state and sensor measurement for various combinations of measurement noise(R) and dynamic noise(Q).

The ratios of R and Q i.e measurement noise and dynamic noise decides what the filter output tends towards. We set one of the above mentioned nosie and vary the other one to see their effects. In this report, Q has been set to 0.001 and the R is varied. The plots for these combinations are given below.

Figure 1 shows the output of the modeled EKF for high measurement noise of 1 and dynamic noise set to 0.001. Since the measurement noise is high, the filter output tends towards the true position and away from the measurement readings.

In Figure 2 the measurement noise is decreased to 0.1 from 1. The filter output now balances between sensor measurement and true position. This seems to be the best combination of measurement noise and dynamic noise to fit the given data.

In Figure 3 the measurement noise is further reduced to 0.03. Because of this reduction in measurement noise, the EKF filter now believes the sensor reading much more and this is evident in the EKF output, which is inclined towards the sensor reading. This spike can be seen clearly around sample 520 where the filter output is spiked in order to reach the sensor measurements.

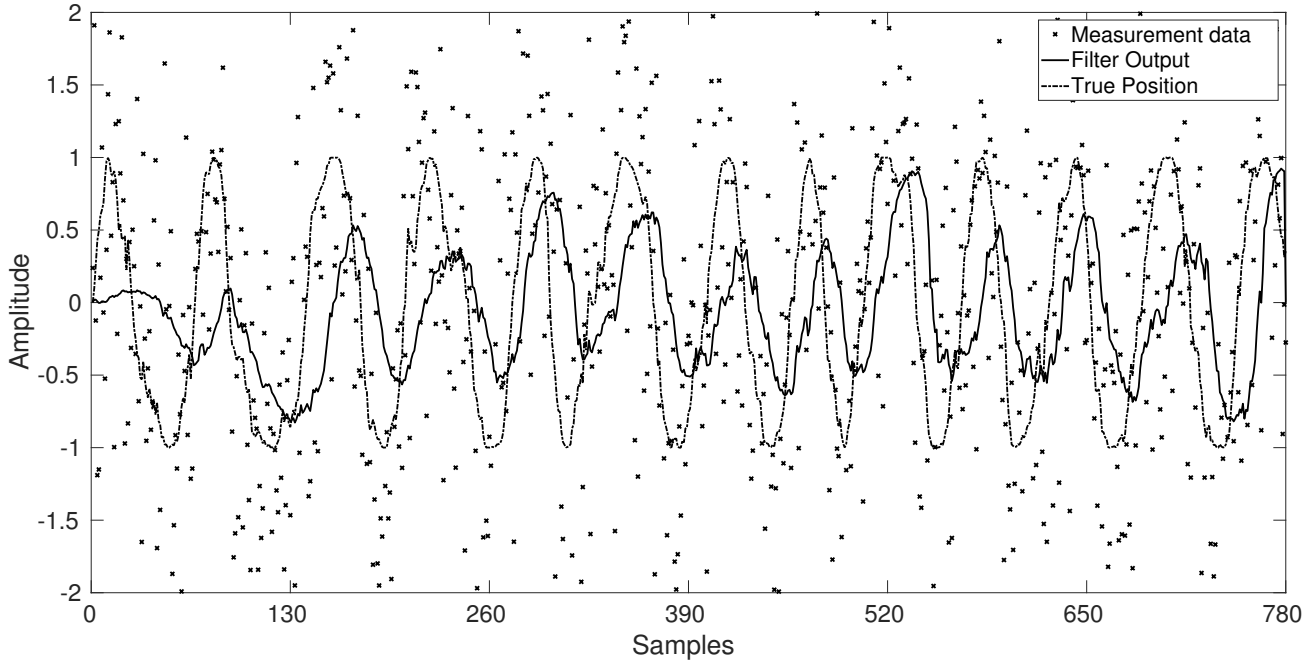

Figure 1: EKF prediction for R = 1 and Q = 0.001

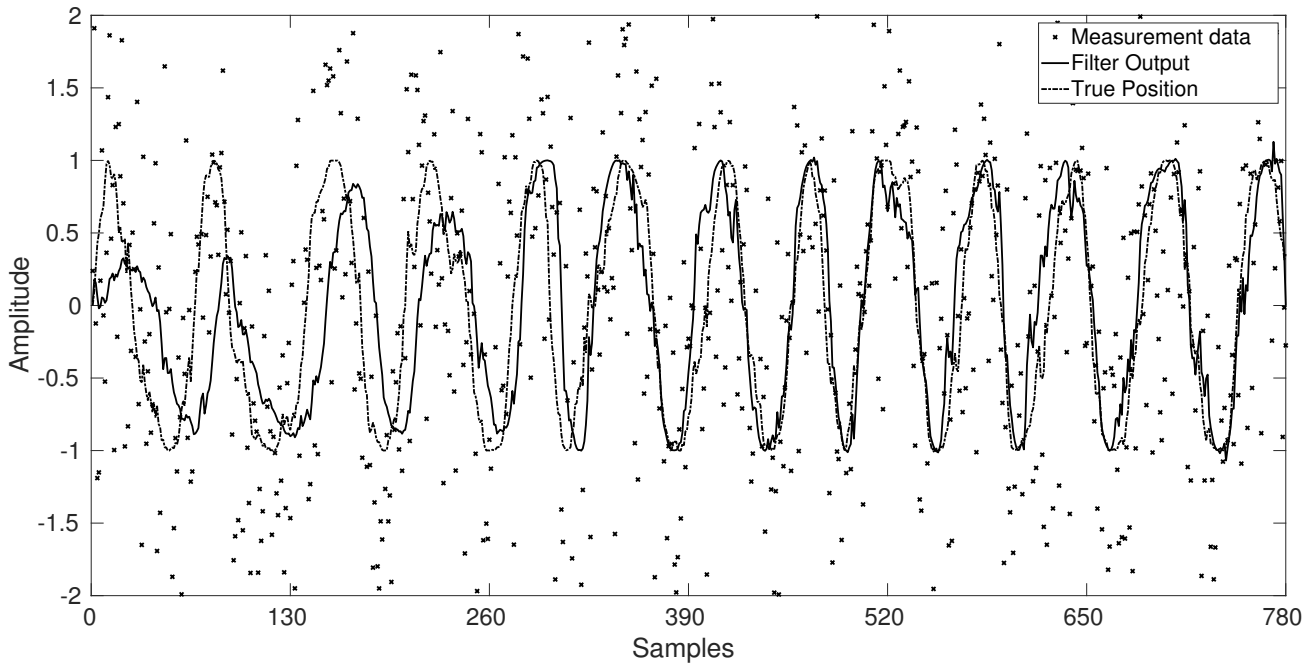

Figure 2: EKF prediction for R = 0.1 and Q = 0.001

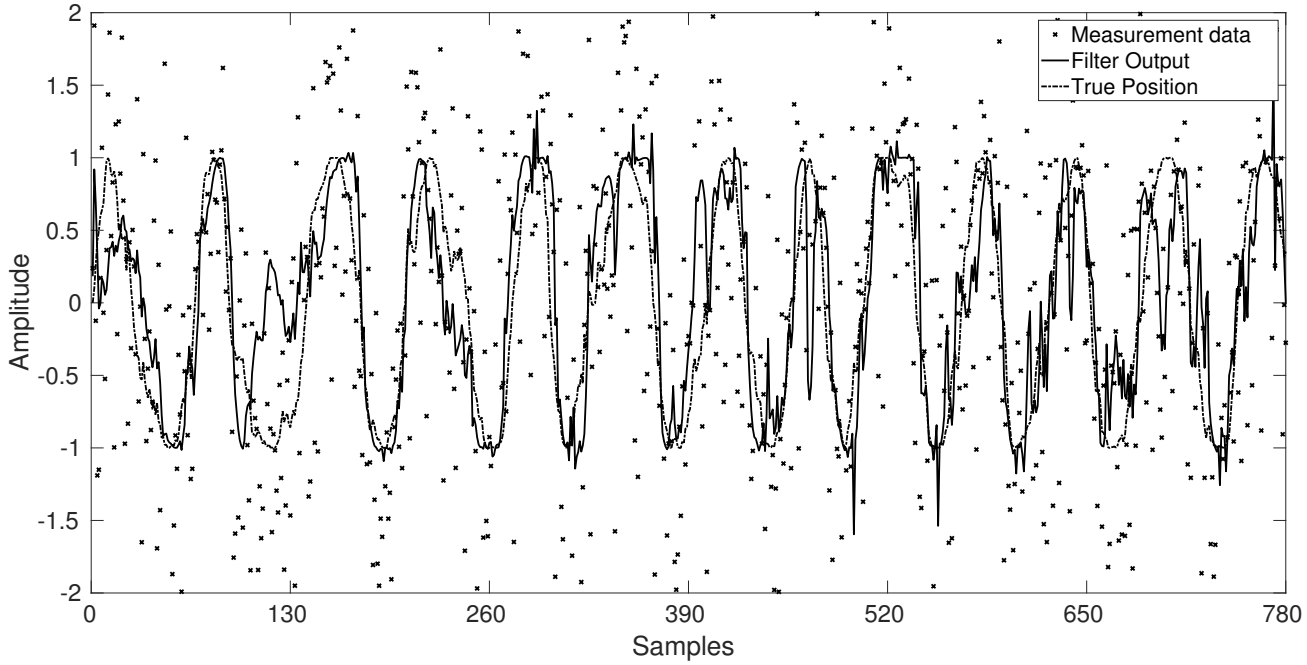

Figure 3: EKF prediction for R = 0.03 and Q = 0.001

4 Conclusion

The goal of this lab was to implement EKF to track the position of a spool of a thread unwinding in a machine. To track the position of the thread, a 1D sinusoidal model was designed and and its model parameter equations were defined. The implementation was done on MATLAB. Sensor measurement readings were given in "sin-data.txt".

To observe the effects of dynamic noise and measurement noise on the filter performance, we varied measurement noise, R while keeping the dynamic noise, Q set at 0.001. With the decrease in R, EKF filter output inclined towards measurement reading and further away from true position. **An acceptable performance was achieved for the ratio R = 0.1 and Q = 0.001**.

Thus the EKF can be applied to a non-Gaussian non-linear function like a sinusoidal signal by linearizing the equations using Taylor series.

References

[1] http://cecas.clemson.edu/~ahoover/ece854/lecture-notes/lecture-ekf-sine.pdf

[2] http://cecas.clemson.edu/~ahoover/ece854/lecture-notes/lecture-ekf.pdf

[3] https://towardsdatascience.com/extended-kalman-filter-43e52b16757d

Appendix

MATLAB Code

```
clc
clear

%Read the data set
T = dlmread("sin-data.txt");

%Actual position for reference
trueData = T(:,1);

%Sensor measurement data
mesData = T(:,2);

%Length of dataset
n = length(mesData);

%Plot the raw dataset
x = 1:n;

% figure (1)
% plot(x,mesData, "O")
% hold on
% plot(x, trueData)
% hold off
% xlabel("time samples");
% ylabel("amplitude");
% set(gca, "FontSize",26);
% legend("Measurement data", "Actual position");

%Initialize variables

t = 1; %time

%Noises
mesNoise = 0.0001;
dynNoise = 0.001;

% Initialize matrices
%PArtial derivatives

Dfa = [0 0 0; 0 1 0; 0 0 0];
```

```
Dgx = [0  0  1];

Dgn = 1;

%Dynamic noise Co-variance
Q = [0  0  0; 0 dynNoise 0; 0 0 0];

%Measurement noise co-variance
R = mesNoise;

%Identity matrix (3x3)
I = eye(3) ;

%State Co-variance
SPrev = I;

%State Matrix
XPrev = [0.001; 0.01; mesData(1,1)];

%Result array
Output = zeros(1, n);

%DeltaT
delta = 0.001;

while(t <  n)

    %Predicting the next state
    XPredict = [ (XPrev(1,1) + (delta*(t-1) * XPrev(2,1))) ;
                 XPrev(2,1);
                 sin( XPrev(1,1) * 0.1) ];

    Dfx = [1 delta*t 0; 0 1 0; 0.1*cos(0.1 * XPredict(1,1)) 0 0];

    %Predicting next state co-variance
    SPredict = (Dfx * SPrev * Dfx') + (Dfa * Q * Dfa');

    %Measurement data
    Y_t = mesData(t);

    %Calculating kalman gain
    K_t = (SPredict * Dgx') / ( (Dgx * SPredict * Dgx') + (Dgn * R * Dgn') );
```

```
    %Update state
    XNext = XPredict + (K_t * (Y_t - XPredict(3,1) ));

    %Update state co-variance
    S_t_t = (I - K_t * (Dgx)) * SPredict;

    %Update prev variables
    SPrev = S_t_t;
    XPrev = XNext;

    Output(1,t) = XNext(3,1);

    %Incriment loop counter
    t = t  + 1;

end
x = 1:n;
% close all

figure (2)
plot(x,mesData, "kx", "LineWidth", 2)
hold on
plot(x, Output(1,:), "K", "Linewidth",2)
% hold on
plot(x, trueData, "k-.","Linewidth",2)
hold off
xlabel("Samples");
ylabel("Amplitude");
set(gca, "FontSize",26);
legend("Measurement data", "Filter Output", "True Position");
xticks([0:130:780])
axis([0 780 -2 2]);

% %Ratios
% dynNoise = 0.001 mesNoise = 1
% dynNoise = 0.001 mesNoise = 0.1
% dynNoise = 0.001 mesNoise = 0.003
```