ECE 8540 Analysis of tracking systems

Lab 6 - Particle Filter

Rahil Modi
C14109603

November 17, 2020

1 Introduction

In this lab we are expected to develop a code for particle filter on a 1D data. It is a data of a non linear model with noise. The filters we learned until now such as Kalman filter it can be used to track a linear system, Extended Kalman filter can be used for non-linear system but both the Kalman filters require the state distribution, observation noise and dynamic noise all of them to be Gaussian. For a non linear system with non-Gaussian distribution we require Particle Filter for tracking.
In this lab we are tracking an object moving back and forth along a string. There are two magnets on two sides and based on the magnetic strength sensed by the object we need to approximate the position of the object. The particle filter is designed based on Bayesian estimation, Monte Carlo approximation and sequential importance sampling.

2 Methods

Particle Filter(PF) is a continuous cycle of predict and update. Following are the steps of formulating the Particle Filter:

1. Determine the state variables.
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.

3 Implementation

Putting together the concepts of Bayesian estimation, Monte Carlo approximation and sequential importance sampling, particle filter can be described.
As with all other filters, the first step is to define the model that will be used in this problem. This model includes the following:

1. X_t, a set of state variables
2. a_t, the set of dynamic noises
3. f(), the state transition equation
4. y_t, the set of measurements

5. n_t, the set of measurement noises

6. g(), the observation equation

The data consists of sensor readings of magnetic field strength experienced by an object hovering over two magnets. The system consists of a 1D position, moving on a line. The system follows a motion pattern where the position ‘zig-zags’ back and forth on a line. The sensor on the system detects a field strength that is the sum of the distances from two fixed-position magnets.

The problem in hand comprises of two state variables:

$$X_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} \tag{1}$$

Where x_t is the position of the object and $\dot{x}_t$ represents the velocity of the object.

The state transition equations are:

$$f(x_t, a_t) = [l]x_{t+1} = x_t + \dot{x}_t T \dot{x}_{t+1} = \begin{cases} 2 & \text{if } x_t < -20 \\ \dot{x}_t + |a_t| & \text{if } -20 \leq x_t < 0 \\ \dot{x}_t - |a_t| & \text{if } 0 \leq x_t \leq 20 \\ -2 & \text{if } x_t > 20 \end{cases} \tag{2}$$

The velocity equation is a piecewise function that adds or subtracts a random amount a_t to the current velocity, depending on the current position. The values a_t are drawn from a zero-mean Gaussian distribution $N(0, \sigma_a^2)$ where $\sigma_a = 0.0625$. a_t represents the dynamic noise. The goal of the state transition equation is to keep the position oscillating about zero but between -20 and 20

The observation equation for this model is:

$$g(x_t, n_t) = \left[y_t = \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(\frac{-(x_t - x_{m1})^2}{2\sigma_m^2}\right) + \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(\frac{-(x_t - x_{m2})^2}{2\sigma_m^2}\right) + n_t \right] \tag{3}$$

where n_t is a random sample drawn from $N(0, \sigma_n^2)$ representing measurement noise. The value of $\sigma_m = 4.0$ and $\sigma_n = 0.003906$.

Since the particle filter is a Monte Carlo approximation, the distribution $p(x|y)$ is represented using a number of samples. In the context of the particle filter, the samples are usually called particles. They are denoted as:

$$\chi = \{x^{(m)}, w^{(m)}\}_{m=1}^{M} \tag{4}$$

where $x^{(m)}$ represents the state of particle m and $w^{(m)}$ represents the weight of particle m. Where, M is the number of particles.

The predict-update cycle for the given problem goes as follows:

1. Each particle m is propagated through the state transition equation

$$\{x_t^{(m)} = f(x_{t-1}^{(m)}, a_t^{(m)})\}_{m=1}^M \tag{5}$$

The value $a_t^{(m)}$ represents the dynamic noise from t 1 to t, and is randomly and independently calculated for each particle m. It may be envisioned as each particle taking a different "guess" at the dynamic noise undertaken for the current iteration.

2. Using the new measurement vector y_t, the weight of each particle is updated.

$$\tilde{w_t}^{(m)} = w_{t-1}^{(m)} \cdot p(y_t|x_t^{(m)}) \tag{6}$$

This weight update equation is based upon selecting the importance distribution as the prior importance function. Other choices for the importance distribution lead to different formulations for the weight update equation.
The value $p(y_t|x_t^{(m)})$ is determined by the measurement noise. It should be calculated by taking the ideal measurement of the particle, and comparing it against the actual measurement, in the model of the measurement noise. The ideal measurement of the particle is calculated as follows:

$$g(x_t^{(m)}, 0) = \left[y_t^{(m)} = \frac{1}{\sqrt{2\pi}\sigma_m}\exp(\frac{-(x_t^{(m)}-x_{m1})^2}{2\sigma_m^2}) + \frac{1}{\sqrt{2\pi}\sigma_m}\exp(\frac{-(x_t^{(m)}-x_{m2})^2}{2\sigma_m^2})\right] \tag{7}$$

The ideal measurement can then be compared against the actual measurement in the model of the measurement noise as follows:

$$p(y_t|x_t^{(m)}) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp(\frac{-(y_t^{(m)} - y_t)^2}{2\sigma_n^2}) \tag{8}$$

3. Normalization of the updated weights:

$$w_t^{(m)} = \frac{\tilde{w_t}^{(m)}}{\sum_{m=1}^M \tilde{w_t}^{(m)}} \tag{9}$$

The normalized weights must sum up to 1.

4. Compute the desired output, such as expected value:

$$E[x_t] \approx \sum_{m=1}^M x_t^{(m)} \cdot w_t^{(m)} \tag{10}$$

5. Check and re-sample if necessary.
The co-efficient of variation statistic can be calculated which helps in deciding if re-sampling is necessary.

$$CV = \frac{1}{M}\sum_{m=1}^M \{M \cdot w^{(m)} - 1\}^2 \tag{11}$$

The effective sample size can be calculated as

$$ESS = \frac{M}{1 + CV} \tag{12}$$

The effective sample size describes how many particles have appreciable weight. In order to check if re-sampling is necessary, the effective sample size can be tested against the number of particles.
In the context of this report, the threshold is set to 50% of the total particles used i.e M = 1000.

```
if (ESS < 0.5 M)
  resample
```

The most common re-sampling method is called the select with replacement. The concept here is to eliminate particles with negligible weights are replace them with particles that have large weights. The pseudo code for re-sampling is given below:

```
Assume particle states in P[1...M], weights in W[1...M].

Q=cumsum(W);          calculate the running totals
t=rand(M+1);          t is an array of M+1 uniform random numbers 0 to 1
T=sort(t);            sort them smallest to largest
T[M+1]=1.0;           boundary condition for cumulative hist
i=j=1;                arrays start at 1
while (i<=M)
  if (T[i] < Q[j])
    Index[i]=j;
    i=i+1;
  else
    j=j+1;
  end if
end while

loop (i=1; i<=M; i=i+1)
  NewP[i]=P[index[i]];
  NewW[i]=1/M;
end loop
```

This algorithm computes a list of indices of particles. The list may include 1 or more copies of the same index (particle). It may also skip over 1 or more indices (particles). After computing the list, it creates a new list of particles of equal weights.

6. Final step is to increment t, i.e t = t + 1 and then iterate.

The MATLAB Code of the implementation above is mentioned in the appendix.

4 Results

In this section all the plots from the above implementation are there.

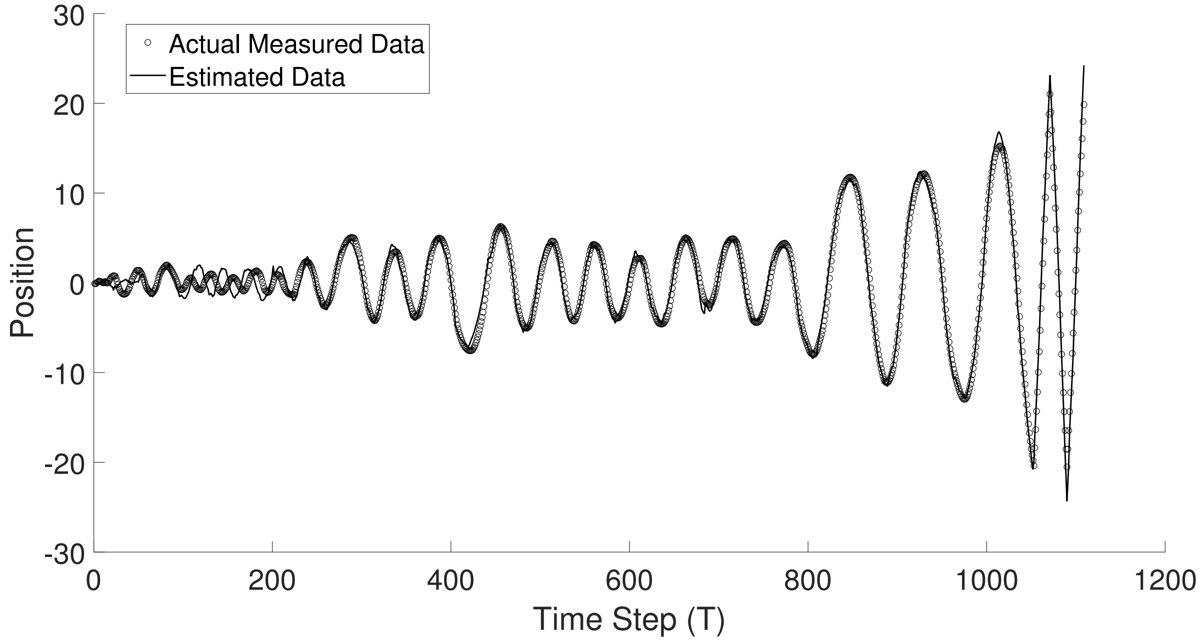


Figure 1: In-phase tracking

In the above image the actual position of the object is plotted with the prediction, the tracking takes some time to lock on but it gets locked properly after some iterations and tracks the object properly. This shows the object is where the filter is tracking.

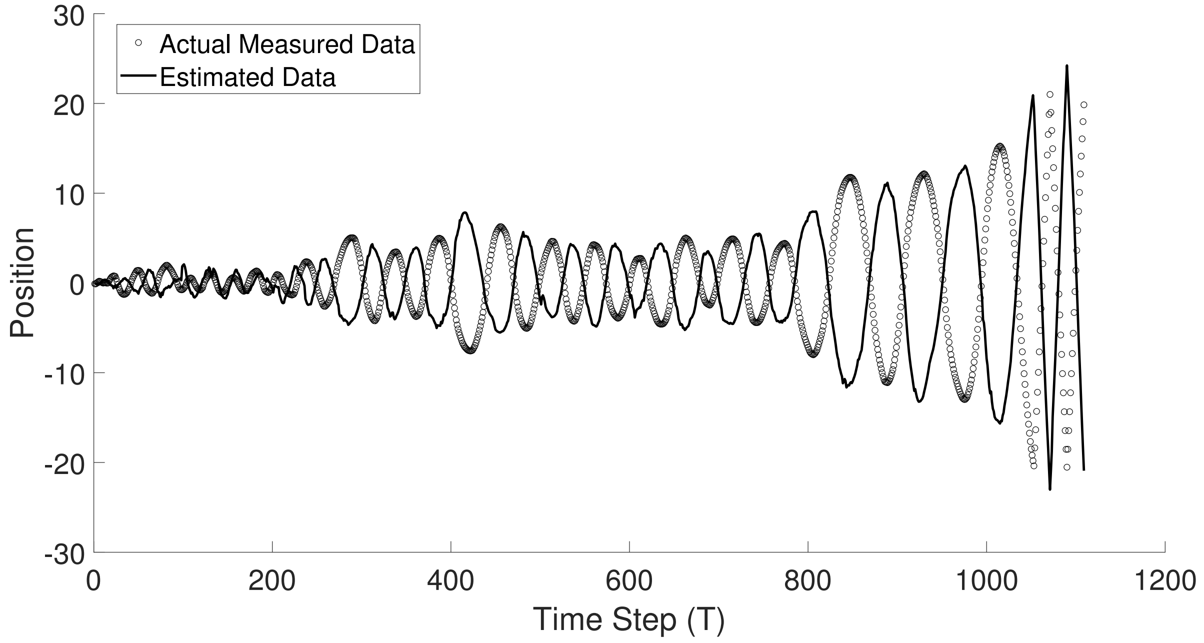
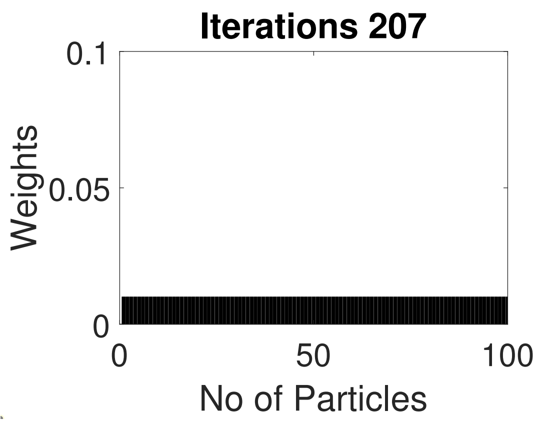
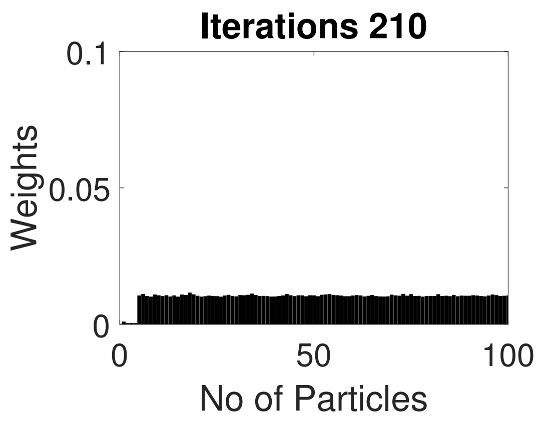


Figure 2: Out-phase tracking

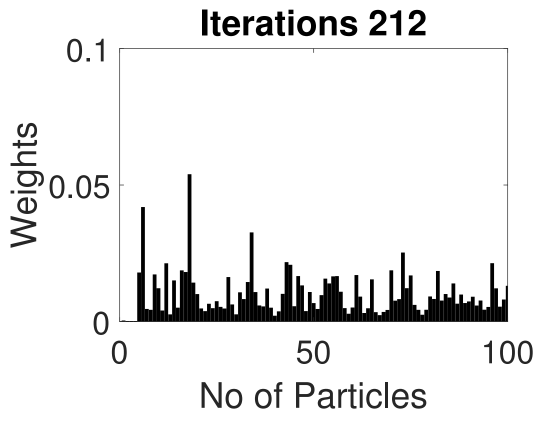
In the above image the tracking is out of phase as due to the mirror effect caused by the two magnet measurement data and the particle filter due to randomness considers the the opposite of the actual position as the original position and starts tracking it.



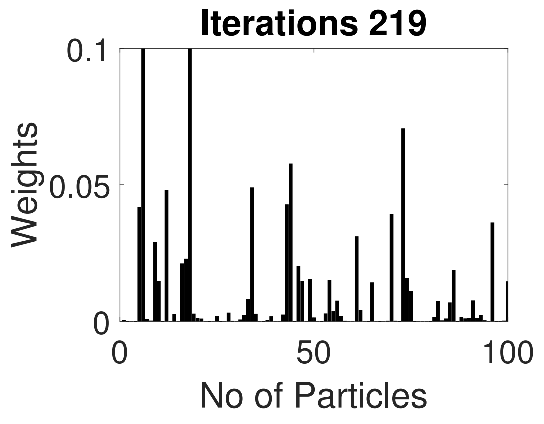
Iterations = 207 ESS = 9.3696



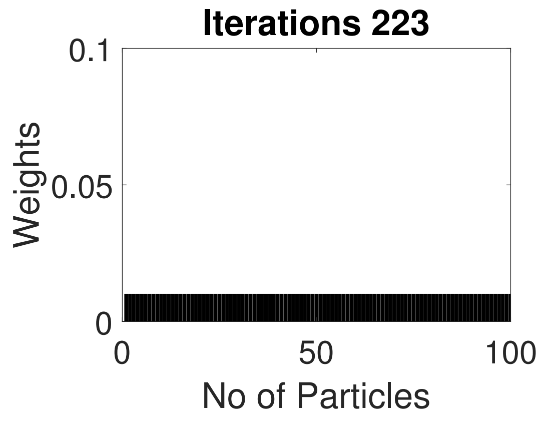
Iterations = 210 ESS = 96.2194



Iterations = 212 ESS = 59.3565



Iterations = 219 ESS = 15.2596



Iterations = 223 ESS = 9.0353

Figure 3: Normalized Weight Distribution through one re-sampling cycle

The above image shows the plots from iterations 207 to 223 which is one sampling cycle. It can be seen in iteration 207 and 223 where resampling occurs and the weights are reset to 1/M which would be 1/100 = 0.01 in my case. ESS represents the no of particles that are still contributing to the approximation and that number keeps falling after every iteration and when it falls below 10 resampling occurs in my case.

5 Conclusion

In this Lab we learned to implement Particle filter to data of an object hovering between two fixed magnets with the help of a string. It was also observed that the this method can have out of phase tracking as seen in Results section. The reason for that is the symmetry in the measurement data of the two magnets with respect to the object so sometimes it starts tracking the mirror of the object position.
I also learned about resampling as in the particle filter after some time many of the particles would have almost zero weight so they would not be contributing to the approximation so we need to resample where the particles with bad weights are replaced with particles with good weights, they might even overlap each other but would contribute more to the approximation of the position of the object. I also learned about Recursive Bayesian estimation, Monte Carlo approximation and sequential importance sampling.

6 Appendix

```
%ECE 8540 - Analysis of Tracking Systems
%Author: Rahil Modi
clc;
clear
close all

data = importdata("magnets-data.txt");

gt_pos             = data(:,1);
gt_vel             = data(:,2);
sensor_measurement = data(:,3);

%Number of Particles
M   = 100;

%Magnet Position
xm1 = -10;
xm2 = 10;

%S.D
sigmaM = 4;
```

```
%State Transition
x_state     = zeros(1,M);
x_vel       = zeros(1,M);
xprev_state = zeros(1,M);
xprev_vel   = zeros(1,M);

%Weights
w_t      = ones(1,M) * 1/M;
w_t_prev = ones(1,M) * 1/M;
wts      = ones(1,M) * 1/M;
y_t      = zeros(1,M);

%Number of resampling
counter_resample = 0;

%Flag to track weight plot
start = 0;

%Plots start when resampling count is:
resample = 10;

%Resample when particles go below RS percentage
threshold = 0.1;

output  = zeros(1,length(sensor_measurement));
X1      = 1 : length(sensor_measurement);
X2      = 1 : M;

for t = 1:length(sensor_measurement)

    for i = 1:M

            %State Transition
            x_state(i) = xprev_state(i) + xprev_vel(i) ;

            if( xprev_state(i) < -20)
                x_vel(i) = 2;

            elseif (xprev_state(i) > 20)
                x_vel(i) = -2;

            elseif (xprev_state(i) >= 0 && xprev_state(i) <= 20 )
                x_vel(i) = xprev_vel(i) - abs(randn * 0.0625);
```

```
        elseif (xprev_state(i) >= -20 && xprev_state(i) < 0)
            x_vel(i) = xprev_vel(i) + abs(randn * 0.0625);
        end

        %Update weight

        y_t(i)      = (1 / (sqrt(2*pi) * sigmaM))  * exp
        ( -((xprev_state(i) - xm1  )^2) / (2 * (sigmaM^2) ))
        + (1 / (sqrt(2*pi) * sigmaM))  *
        exp( -((xprev_state(i) - xm2  )^2) / (2 * (sigmaM^2) ));

        prob        = ((1 / (sqrt(2*pi) * 0.003906)) * exp
        ( - ((y_t(i) - sensor_measurement(t) )^2) / (2 * (0.003906^2) )));

        wts(i) = w_t_prev(i) * prob;
        w_t_prev(i) = wts(i);

        xprev_state(i)  = x_state(i);
        xprev_vel(i)    = x_vel(i);

end

%Normalize weights
Exp     = 0;
Index   = zeros(1,M);
w_t     = wts ./ sum(wts);

%Calculated Expected Filter Output and Co-eff of variation
for k = 1:M
    %Expected Filter Output
    Exp = Exp +  (w_t(k) *  x_state(k));
end

%Store Expected filter output for plotting
output(t)   = Exp;
CV          = var ( w_t ) /(mean( w_t ) ^2) ;
%Effective Sampling Size
ESS = M / (1 + CV);
if(start)
    figure(t)
    bar(X2, w_t,'k')
    xlabel('No of Particles');
    ylabel('Weight');
    set(gca,'FontSize',24)
    disp(strcat('iteration = ',num2str(t), ' ESS = ', num2str(ESS) ))
```

```
end
%Resampling
if (ESS<threshold*M)
    %Track the number of times Resampled
    counter_resample = counter_resample + 1;

    if (counter_resample == resample)
        start = 1;
    end
    %Cumulative Weights Q
    Q       = cumsum( w_t ) ;
    %Guesses
    T       = rand (M+1 ,1) ;
    %Sorting the guesses
    T       = sort(T) ;
    T(M+1)= 1.0;
    g       =1;
    j       =1;
    while(g<=M)
        %Find good particle indices
        if (T(g) < Q(j))
            Index (g) = j;
            g = g+1;
        else
            j = j +1;
        end
    end
    %Replace bad particles with good particles
    for i = 1:M
        x_state(i)      = x_state(Index(i));
        xprev_state(i)  = xprev_state(Index(i));

        x_vel(i)        = x_vel(Index(i));
        xprev_vel(i)    = xprev_vel(Index(i)) ;

        w_t(i)          = 1/M;
        w_t_prev(i)     = 1/M;
    end
end
%Plot resampled weights
if(start)
    figure(t)
    bar(X2, w_t, 'k');
    set(gca,'FontSize',24);
```

```
        xlabel('No of Particles');
        ylabel('Weights');
        axis([0, M, 0, (1/M)*10]);
        disp(strcat('iteration = ',num2str(t), ' ESS = ', num2str(ESS) ))
    end
    if(counter_resample == resample + 1)
        start = 0;
    end
end
figure
xlabel ('Time Step (T)');
ylabel ('Position');
hold on
plot(X1 ,gt_pos, ' kO ','MarkerSize', 5) ;
plot( X1 ,output, 'k ', 'Linewidth', 1.2 ) ;
legend ('Actual Measured Data' , 'Estimated Data');
set(gca,'FontSize',24)
hold off
```