ECE 8540
Analysis of Tracking Systems

Assignment 6
Particle Filter

Vivek Koodli Udupa
C12768888

November 20, 2018

1 Introduction

This report considers the problem of tracking a non-linear model with generalized noise distribution using Particle Filter. 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. Kalman and Extended Kalman Filters are generally used for such tracking problems. However, Kalman filter is only indented for linear systems. The extended Kalman filter works on nonlinear systems. Both filters assume that the state distribution, dynamic noise and observation noise are all Gaussian. In order to track a more generalized non-linear non-Gaussian distribution Particle Filter is preferred which uses a genetic mutation-selection sampling approach, with a set of particles (also called samples) to represent the posterior distribution of some stochastic process given noisy and/or partial observations. The state-space model can be nonlinear and the initial state and noise distributions can take any form required.

A distribution is called "intractable" if it is not easily modelled by an analytical function, or that even if it can be modelled, the function does not provide an easy mechanism for calculation. In order to deal with such distributions, Monte Carlo approximation is used. Monte Carlo approximation is the practice of using a set of samples to approximate a distribution. Each sample is given a state and a weight. These samples are simply a "guess" at the actual state of the object being tracked. Weighing each guess for its accuracy is difficult. To work around this problem importance sampling is used. The idea behind importance sampling is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. If these ”important” values are emphasized by sampling more frequently, then the estimator variance can be reduced. Thus with enough samples, weighted and positioned appropriately, any distribution can be approximated.

In general tracking problems, the desire is to determine the probability of all transitions over time given all the observations. However, realistically it is better to set up a recursive relationship where the next estimate is based on the previous estimate and the latest measurement. The recursive Bayesian estimation helps us to set up such a relationship.

This report is focused on tracking an object moving back and forth along a string. Two magnets are placed at fixed positions near the string. The object senses the magnetic field strength. Given the magnetic field measurement, the position of the object is to be tracked. In other words, what is the probability of position of the object given a field strength? This question is addressed in this report using a Particle Filter designed based on the concepts of Bayesian estimation, Monte Carlo approximation and sequential importance sampling.

2 Methods

Particle Filter(PF) is a continuous cycle of predict and update. When formulating the problem for the PF following steps are considered:

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.

2.1 Describing the Particle Filter Parameters

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

2.2 Defining the Particle Filter

The dataset "magnets-data.txt" 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 objective is to track the position of this object based on the magnetic field strength readings.

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 equation is as follows:

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

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 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. Here M represents the number of particles to be used.

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

1. Each particle 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 time t - 1 to t. This is randomly and independently calculated for individual particle separately. This can be viewed as each particle taking a random 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:

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

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. Normalize 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:

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

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.

All of the above implementation has been done in MATLAB. Refer Appendix for the implementation Code.

3 Result

This section consists of plots resulting from the implementation shown above.

Figure 1 and Figure 2 shows the plot of actual position of the object vs the filter's prediction of the position of the particle. The filter takes some time to lock on to the object. This can be observed in the initial iterations where the filter is having trouble predicting the position of the object. After around 300 iterations, the filter starts to track the object more accurately.

In Figure 1 The actual position of the object and the filter's prediction of the object's true position are in-phase. This means that the actual position of the object is where the filter is predicting it to be.

Figure 2 represents out-of-phase tracking where the filter's prediction is off by 180°. Here the actual position of the object is opposite to where the filter is predicting it to be.

Figure 3 and Figure 4 represent the normalized weight distribution over the span of one re-sampling cycle.
These plots are taken from iteration 120 through iteration 132 where 120 and 132 are the re-sampled iterations. The plot consists of normalized weights that add up to 1. After re-sampling the weights are re-initialized to $\frac{1}{M}$(M = 1000 in this case). ESS represents the number of particles that are still contributing to the approximation of the distribution. It can be observed that this number drops through each iteration and the re-sampling is performed when ESS drops below 50% of the total number of particles, M. In this case re-sampling is done whenever ESS drops below 500. It can also be observed that the weight distribution takes the form of a double Gaussian.

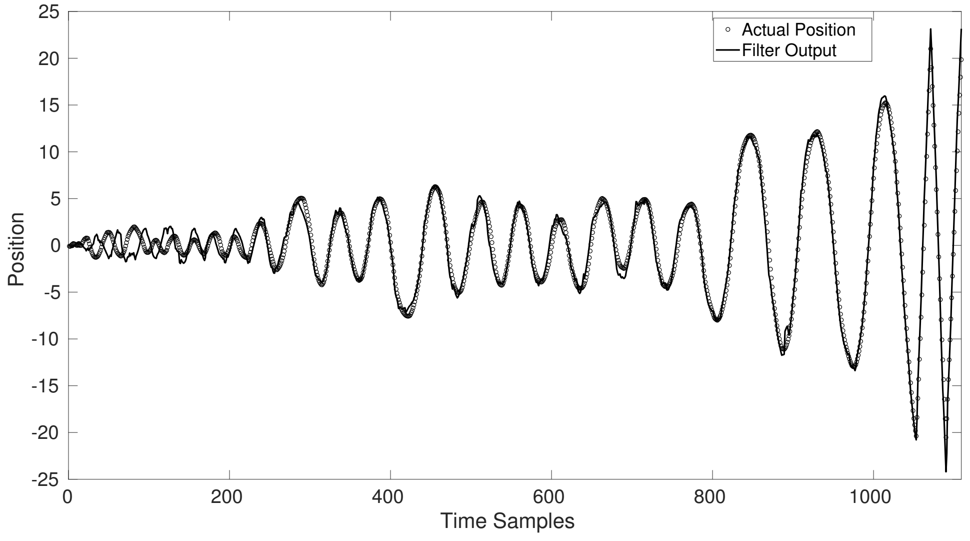


Figure 1: In-phase tracking

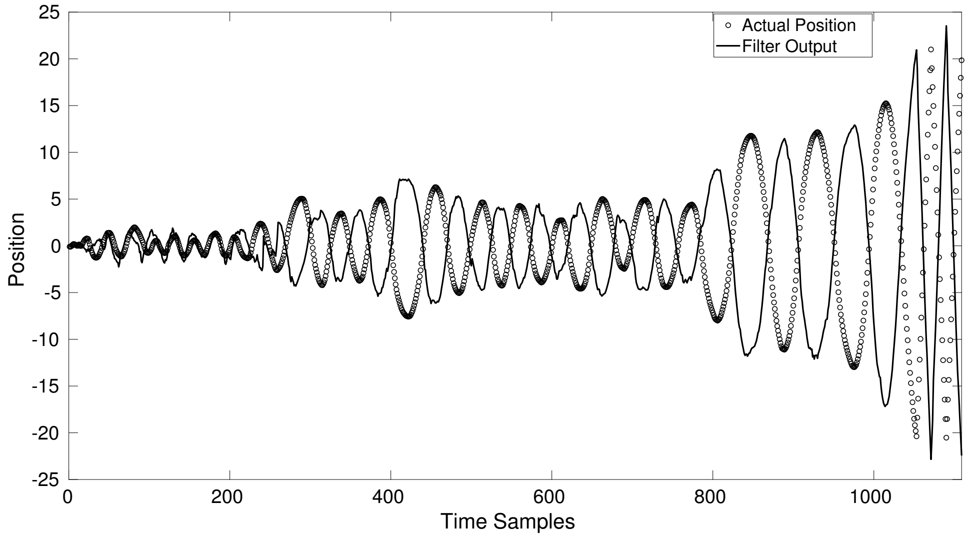
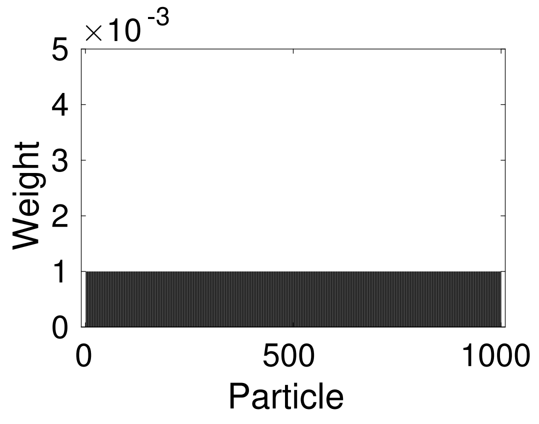
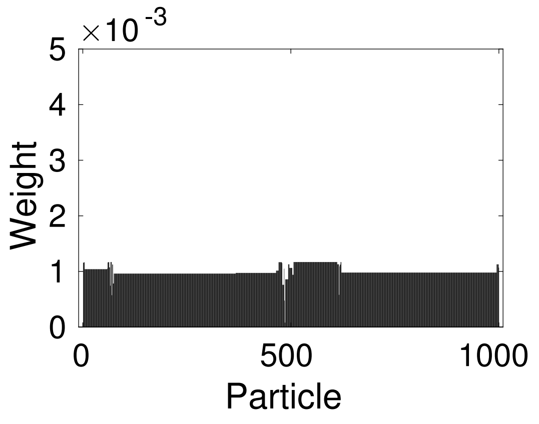


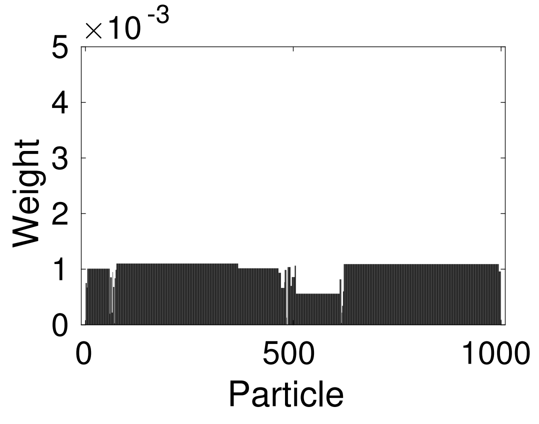
Figure 2: Out-of-phase tracking



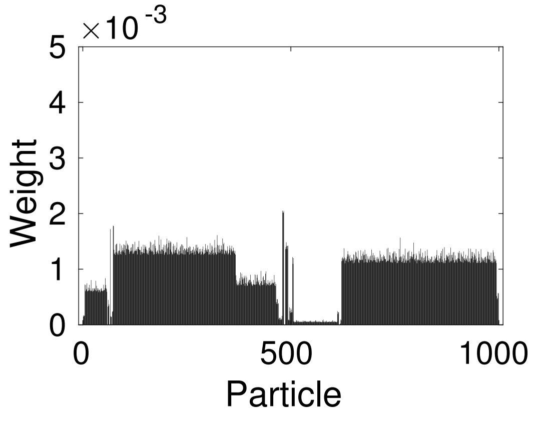
Iterations = 120 ESS = 381.7177



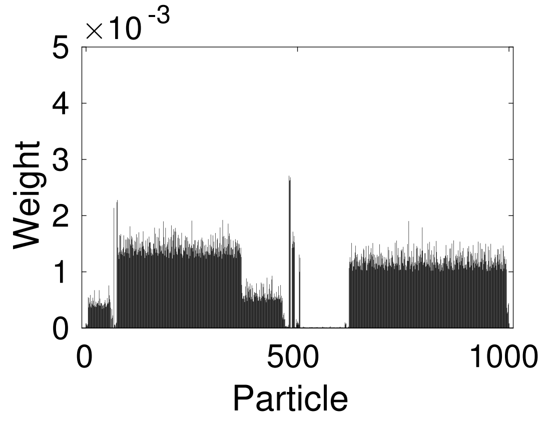
Iterations = 121 ESS = 993.3567



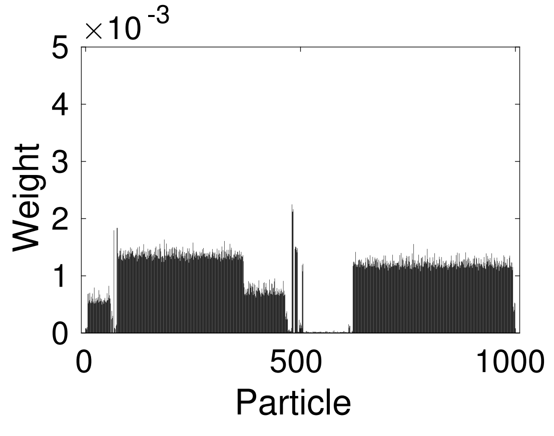
Iterations = 122 ESS = 963.4150



Iterations = 123 ESS = 827.667

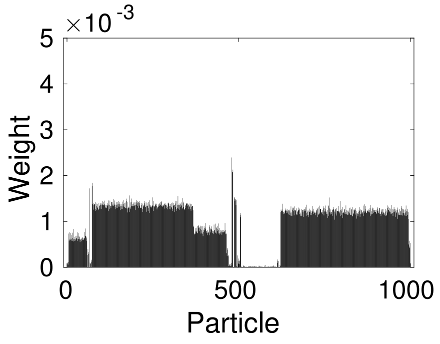


Iterations = 124 ESS = 772.4266

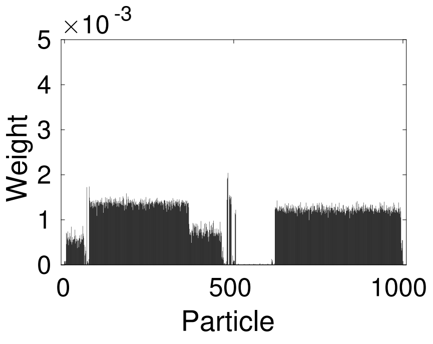


Iterations = 125 ESS = 808.7435

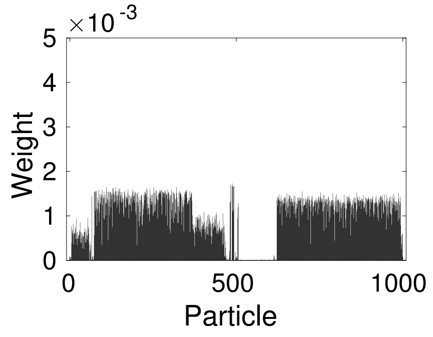
Figure 3: Normalized Weight Distribution through one re-sampling cycle(Part 1)



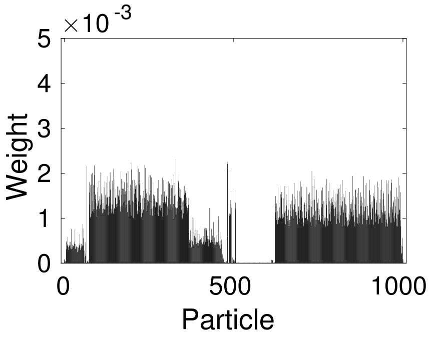
Iterations = 126 ESS = 816.2481



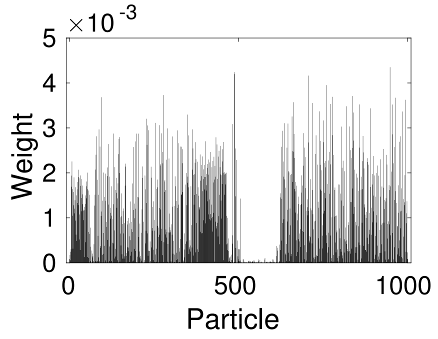
Iterations = 127 ESS = 805.7169



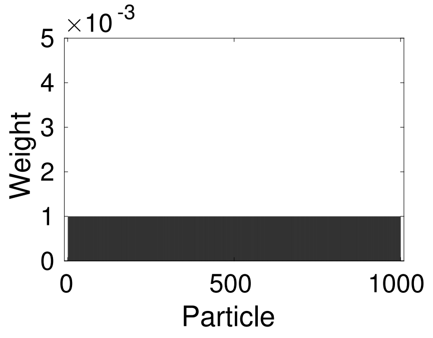
Iterations = 128 ESS = 788.1496



Iterations = 129 ESS = 743.2891



Iterations = 131 ESS = 517.3971



Iterations = 132 ESS = 431.4715

Figure 4: Normalized Weight Distribution through one re-sampling cycle(Part 2)

4 Conclusion

In this report, the particle filter was used to track an object hovering over two magnets along a string. The given dataset consisted of the magnetic field strength readings experienced by the object. The implemented filter can have an in-phase and an out-of-phase tracking as shown in Figure 1 and Figure 2 due to the symmetry of the measurement data of two magnets with respect to the object being tracked.

The advantage of using particle filter over EKF and KF is that Particle Filter can work with state transition equations of any form and non-Gaussian noises. Recursive Bayesian estimation was used to set up a recursive relationship where the next estimate is based on the previous estimate and current measurement. Monte Carlo approximation along with importance sampling was used to weigh the samples, which are guesses of the actual state of the object. The idea of this algorithm is that the particles move to a distribution of possible new states at each time step. It is impossible to know where the system has actually transitioned to, but the hope is that some of the particles have transitioned in similar direction. An observation is taken and weights of the particles are updated based on how well its transition matches against the observation. Finally the weights are normalized so that their sum equal to 1, so that they properly represent a probability distribution.

This is the basic operation of the particle filter. However there is an issue. There might be certain cases where some particles move so far away that their weight approaches zero. This reduces the number of particles that contribute to the approximation of the distribution. The solution to this issue is re-sampling. Re-sampling is performed to replace particles with bad weights with particles that have good weights so that they can contribute more towards the estimation.

In conclusion, particle filter is not one specific set of equation. It has many parameters other than the usual selection of model variables and equations. These include proposal distribution q() that determines the weight update equation, number of particles M, when to re-sample and re-sampling method.

In the experiment conducted, recursive Bayesian distribution was used along with 1000 particles(M), re-sampling at 50% ESS and select with replacement re-sampling method.

References

[1] https://en.wikipedia.org/wiki/Particle_filter

[2] https://en.wikipedia.org/wiki/Importance_sampling

[3] http://cecas.clemson.edu/~ahoover/ece854/

Appendix

MATLAB Code

```
clc;
clear
close all

Table = dlmread("magnets-data.txt");

actualPos = Table(:,1);
actualVel = Table(:,2);
sensorRead = Table(:,3);


%Number of Particles
M = 1000;

%Initialization

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

%S.D
sigmaM = 4;

%State Transition
XState = zeros(1,M);
XVel = zeros(1,M);

XPrevState = zeros(1,M);
XPrevVel = zeros(1,M);

%Weights
wts = ones(1,M) * 1/M;
wtsPrev = ones(1,M) * 1/M;
wtsUN = ones(1,M) * 1/M;

ytP = zeros(1,M);

%Resampling
Q = zeros(1,M);
T = zeros(1,M+1);
```

```
%For Weights Plotting
weightsBeforeReset = zeros(1,M);
weightsReset = zeros(1,M);

%Number of resampling
ResampleCount = 0;

%Flag to track weight plot
start = 0;

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

%Resample when particles go below RS percentage
RS = 0.5;

%Output
output = zeros(1,length(sensorRead));

%X-axis to plot filter output
X1 = 1 : length(sensorRead);
%X-axis to plot weights
X2 = 1 : M;

%Storing plots
store = 0;

for t = 1:length(sensorRead)

    for i = 1:M

            %State Transition
            XState(i) = XPrevState(i) + XPrevVel(i) ;

            if( XPrevState(i) < -20)
                XVel(i) = 2;

            elseif (XPrevState(i) > 20)
                XVel(i) = -2;

            elseif (XPrevState(i) >= 0 && XPrevState(i) <= 20 )
                XVel(i) = XPrevVel(i) - abs(randn * 0.0625);
            elseif (XPrevState(i) >= -20 && XPrevState(i) < 0)
                XVel(i) = XPrevVel(i) + abs(randn * 0.0625);
            end

            %Update weight

            ytP(i) = (1 / (sqrt(2*pi) * sigmaM))  * exp( -((XPrevState(i) - xm1
)^2) / (2 * (sigmaM^2) )) + (1 / (sqrt(2*pi) * sigmaM))  * exp( -((XPrevState(i
)^2) / (2 * (sigmaM^2) ));

            prob = ((1 / (sqrt(2*pi) * 0.003906)) * exp ( - ((ytP(i) - sensorRe

            wtsUN(i) = wtsPrev(i) * prob;
            wtsPrev(i) = wtsUN(i);

            XPrevState(i) = XState(i);
            XPrevVel(i) = XVel(i);

    end

    %Normalize weights
    %Initialization
    wtsSum = 0;
    Exp = 0;
    CV = 0;
    Index = zeros(1,M);

    %Find cumulative weight sum
    for k = 1:M
        wtsSum = wtsSum + wtsUN(k) ;
    end

    %Normalize weights
    for k = 1:M
        wts(k) = wtsUN(k) / wtsSum ;
    end

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

        %Coefficient of variation
        CV = CV + (((M * wts(k)) - 1) ^ 2);
```

```
end
CV = 1/M * CV;

%Store Expected filter output for plotting
output(t) = Exp;

%Effective Sampling Size
ESS = M / (1 + CV);

%Plot weights if flag is on
if(start)
    figure(t)
    bar(X2, wts,'k')
    axis([-10 M+10 0 (1/M)*5])
    xlabel('Particle');
    ylabel('Weight');
    set(gca,'FontSize',24)
    fname = strcat('/Report/Figures/', num2str(t), '.eps');
    if(store)
        saveas(figure(t),[pwd fname]);
    end
    disp(strcat('iteration = ',num2str(t), ' ESS = ', num2str(ESS) ))


end

%Resampling
 if( ESS < RS * M )

    %Track the number of times Resampled
    ResampleCount = ResampleCount + 1;

    %Check for lucky resampling number
    if(ResampleCount == luckyNum)
        start = 1;
    end

    %Cumulative Weights Q
    Q(1) = wts(1);
    for k = 2:M
        Q(k) = Q(k-1) + wts(k);
    end

    %Guesses
```

```
        T = rand(1,M);
        T(k+1) = 1;

        %Sorting the Guesses
        T = sort(T);

        i = 1;
        j = 1;

        while( i <= M )

            %Find good particle indices
            if( T(i) < Q(j) )
                Index(i) = j;
                i = i+1;
            else
                j = j+1;
            end

        end

%         Replace Bad Particles with good particles
        for i = 1:M
            XState(i) = XState(Index(i));
            XPrevState(i) = XPrevState(Index(i));

            XVel(i) = XVel(Index(i));
            XPrevVel(i) = XPrevVel(Index(i)) ;

            wts(i) = 1/M;
            wtsPrev(i) = 1/M;
            wtsUN(i) = 1/M;
        end

        %Plot resampled weights
        if(start)
            figure(t)
            bar(X2, wts, 'k');
            axis([-10 M+10 0 (1/M)*5]);
            set(gca,'FontSize',24);
            xlabel('Particle');
            ylabel('Weight');
            fname= strcat('/Report/Figures/', num2str(t), 'resampled.eps');
            if(store)
```

```
                saveas(figure(t),[pwd fname]);
            end
            disp(strcat('iteration = ',num2str(t), ' ESS = ', num2str(ESS) ))
        end

        %After 1 full cycle after last resample
        if(ResampleCount == luckyNum + 1)
            start = 0;
        end

    end

end

close all

figure(1)
plot(X1,actualPos, 'kO','MarkerSize', 5);
hold on
plot(X1,output, 'k','LineWidth',2)
hold off
xlabel("Time Samples");
ylabel("Position");
legend("Actual Position", "Filter Output");
axis([0 1109 -25 25])
set(gca,'FontSize',24)



disp("SUCCESS!!!!!!!")
```