Lab 4: Particle Filter

Christopher Cornwall

11/17/2015

1 Introduction

This report considers the problem of tracking a given set of one dimensional measurement moving on a line between two magnets using a filtering technique. Tracking relates to observing the respective positions of the data in a timely ordered sequence. Filtering technique relates to the method used to track the position of the data. Position refers to a given location after a given time period.

This type of problem normally occurs when we are given a set of measurement data, and a model function cannot be used. This is a result of the data to be fit or track is usually not expected to follow the same behavior forever, hence, we use tracking technique to track the data. This is a problem because we want to know where a particular thing is, in terms of it position in relation to time. However, the answer to this can be a scalar but the reality in the tracking cases, is that the answer is rarely ever true. Hence, a filter makes this explicit by calculating a probability distribution for the variable of interest, rather than a scalar.

There are many different types of filters that can be used to track the true position of given set of measurement. People have tried to solve this problem by using a range of different methods. These methods range from the Kalman Filter (KF) that was demonstrate in lab 3, where the system was linear and all noise were Gaussian; Extended Kalman Filter (EKF) and Unscented transform (UT). However, even though the methods outline above are good methods, they are only good in relation to the type of tracking you are doing. Whether it is linear or nonlinear and the in terms of the distribution of the noise in the system and measurement.

Particle Filter (PF) was used for the filtering technique to track the true position using the measurement data. The Particle Filter (PF) is based on Bayesian theory and importance sampling. Similar to the Kalman Filter that was implement in lab 3, the Particle FIlter goal is to estimate x_{t} (true position) recursively based on the observations, y_{t} (sensor measurement). However, unlike the Kalman Filter, the Particle FIlter can track non-linear and non-Gaussian objects. Therefore, since we are working on a non-linear and non-Gausian tractable case, we use the Particle Filter to track the true position using the measurement data.

2 Methods

Like the Kalman Filter the Particle Filter is a continuous cycle of predict state to update state. This is a procedure that takes eight steps. These eight steps form the main loop of the filter. First you need an equation that will predict the next state particles base on the previous state particles. Hence, we use a transition equation to transit it from the previous to the next state. Secondly you obtain an equation to calculate the ideal measurement of the particles, which is called the observation equation. Thirdly we obtain our actual measurement data and compare it against the ideal measurement of the particles in a new equation called the importance distribution. The fourth step is to update the weight for each particle by using the prior importance distribution function. fifth step is to normalize the updated weights, so they sum to 1. The sixth step is to compute the desired output, such as the expected value (mean). The seventh step is to check if sampling is necessary, and if so, resample. The eighth and final step is to let t = t + 1; and iterate.

For the problem, there are two state variables:

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

Each particle m is propagated through the state transition equation:

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

The state transition equation is as follows:

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

The ideal measurement of the particle, the observation equation

$$y_t^m = \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(\frac{-(x_t^{(m)} - x_{m1})^2}{2\sigma_m^2}\right) + \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(\frac{-(x_t^{(m)} - x_{m2})^2}{2\sigma_m^2}\right) \quad (2)$$

Importance distribution equation

$$p(y_t|x_t^{(m)}) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(\frac{-(y_t^{(m)} - y_t)^2}{2\sigma_n^2}\right) \quad (3)$$

Updating the weights for each particle by using the prior importance function

$$\tilde{w}_t^{(m)} = w_{t-1}^{(m)} p(y_t|x_t^{(m)}) \quad (4)$$

normalize the updated weights, so they sum to 1

$$w_t^{(m)} = \frac{\tilde{w}_t^{(m)}}{\sum_{m=1}^{M} \tilde{w}_t^{(m)}} \tag{5}$$

Compute the desired output, as the expected value (mean)

$$E[x_t] = \sum_{m=1}^{M} x_t^{(m)}.\ w_t^{(m)} \tag{6}$$

Check if sampling is necessary, and if so, resample. where:

$$CV = \frac{1}{M} \sum_{m=1}^{M} (M.\ w^{(m)} - 1)^2 \tag{7}$$

And

$$ESS = \frac{M}{1 + CV} \tag{8}$$

Now if (ESS < 0.5 M) then resample
now let t = t + 1; and iterate

Setting up the Particle Filter using the above procedures, the tracking or filtered position was obtain from step six as time prolong. These position was store as the estimated position and was later used to plot graphs compared with the true position of the data against time step. All calculations was done with MATLAB. All plots for the Particle Filter can be seen in section 3.

3 Results

Figure 1 is just the measurement data obtain form the sensor readings. This data show a lot of modal, which implies that the noise in the data is non-Gaussian. In figure 2 we see how the Particle Filter track the actual position when 100 particles were used. This shows a output of uniform tracking in the middle but at both ends the tracking was off by some degree. This output was improve in figure 3 where more particles were use to improve the monte carlo to get a better estimation of the output. Figure 3 shows the output of when the particle filter algorithm is added to the measurement data, but this time with far more particles. M = 1000 particles was used in the Particle Filter to improvement the monte carlo when the particles were generated. The particle Filter output track the actual/true position almost completely smooth except for at the begin. At this point we see that the tracking is just slightly out of phase with the actual position. With the resambling added to the particle filter, we see that that those particles with the highest probability, or weight, are used more often to create the particles at the next time instance. Those particles with a small probability are dropped. In figure 4 we see the distribution of the particles before resampling. We see from the graph that each particle is weighted with different weights to get the best approximation. The distribution show two modal, which is just two gaussian distribution of particles as they move. In figure 5 we see the distribution of the particles after resampling. We see from the graph that each particle is weighted with the same weight

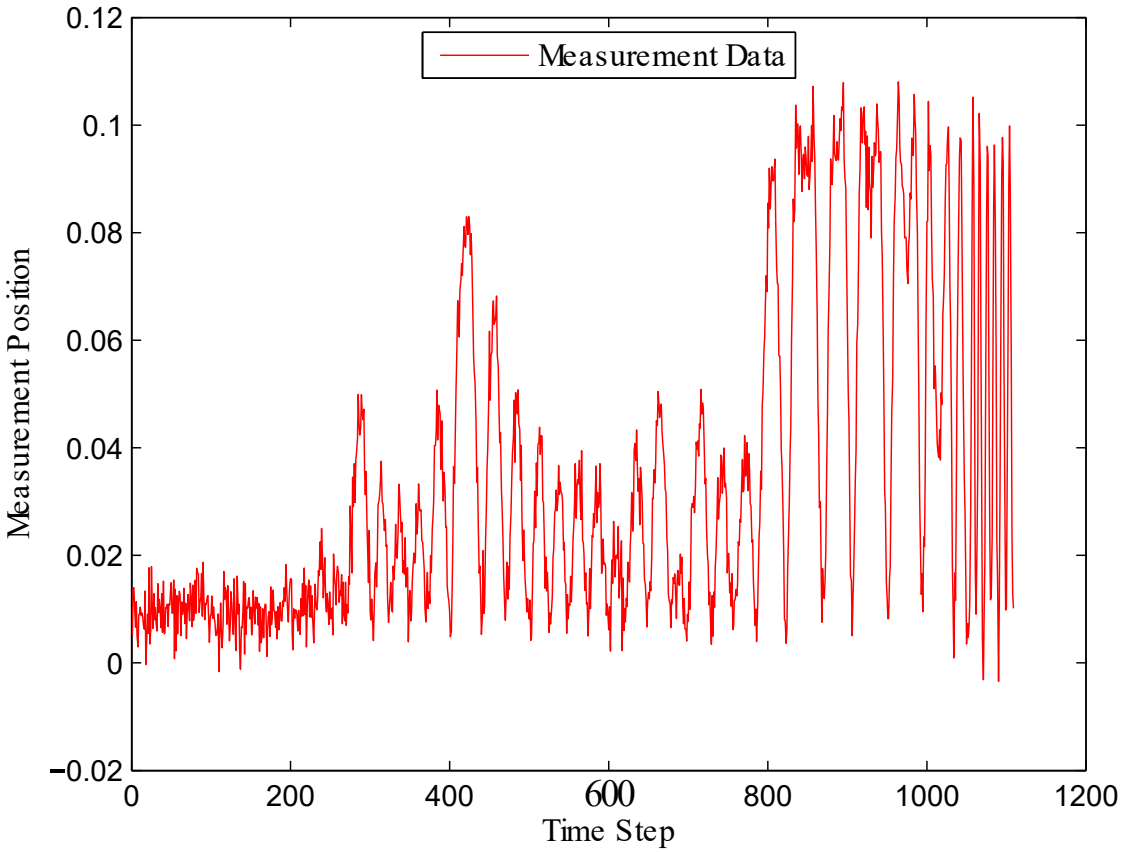

Figure 1: Figure Showing the measurement data obtain from the sensor readings.

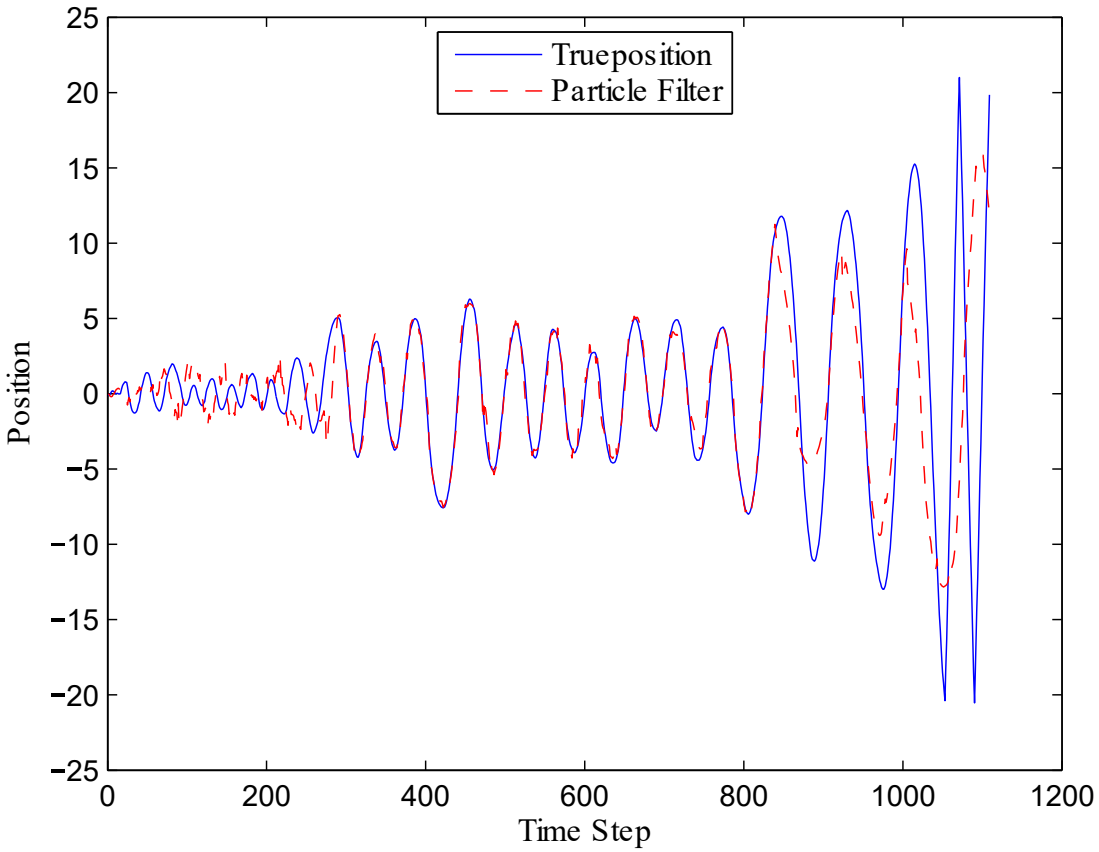

Figure 2: Figure Showing how the Position is track over time using the Particle Filter with 100 particles.

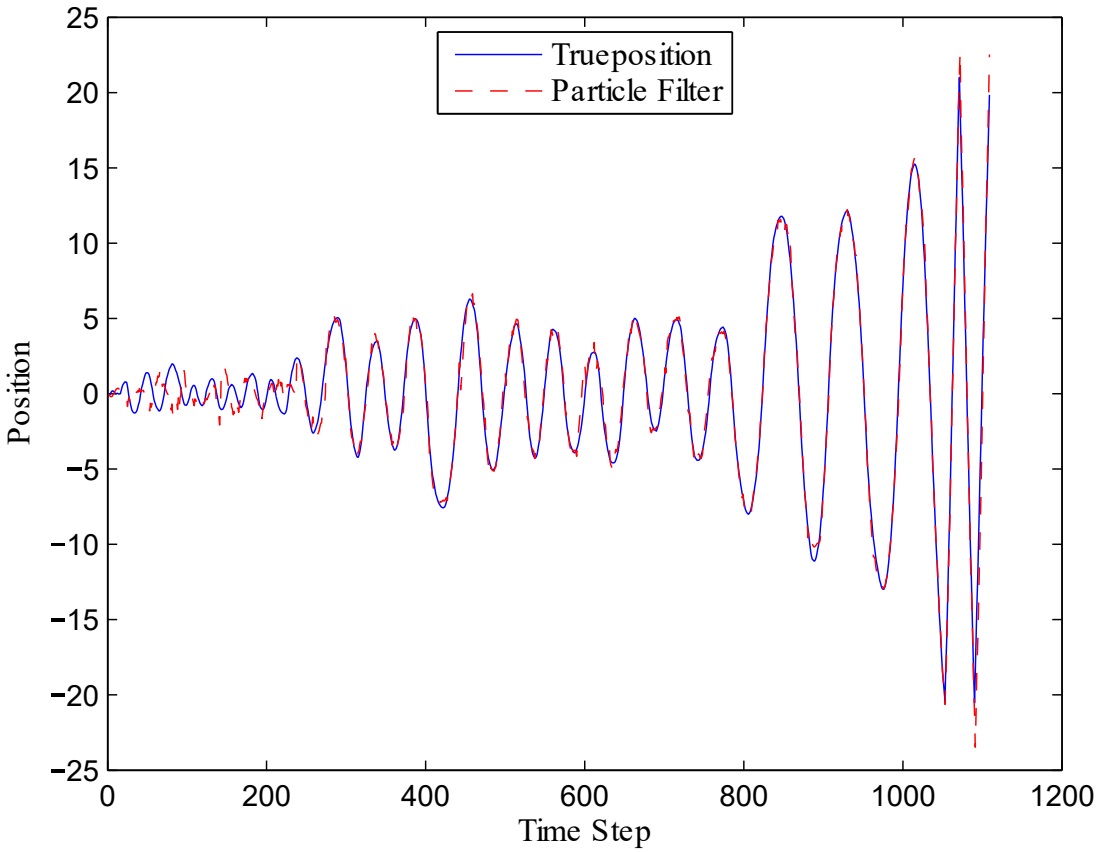

Figure 3: Figure Showing how the Position is track over time using the Particle Filter with 1000 particles.

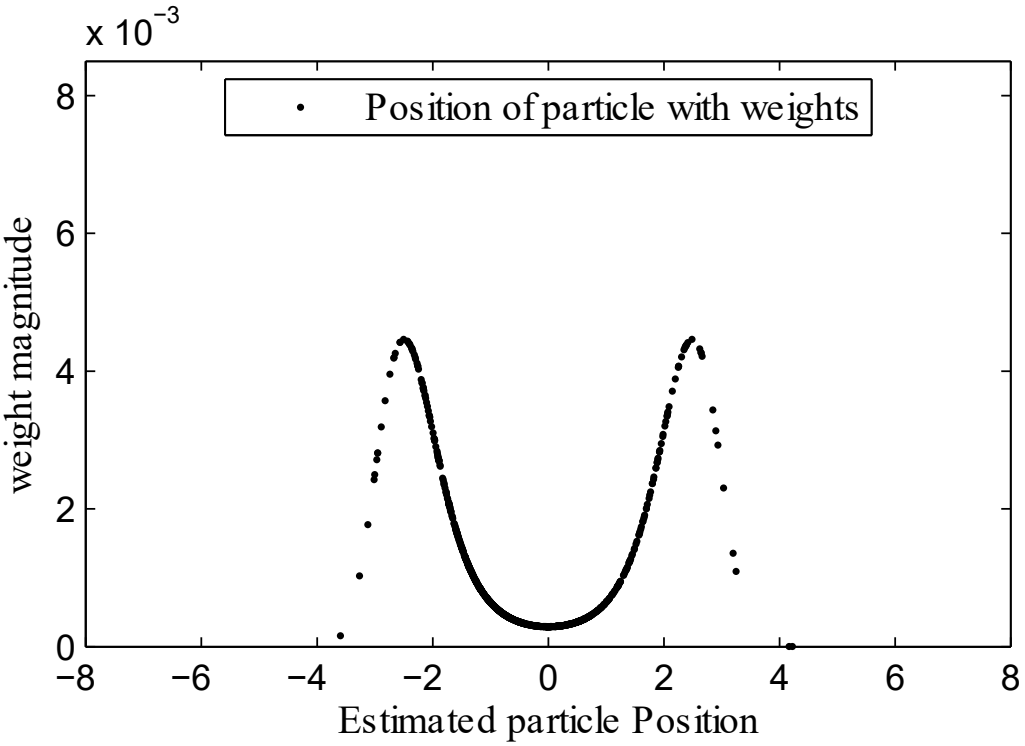

Figure 4: Figure Showing the distribution of the particle before resampling at time step t = 26.

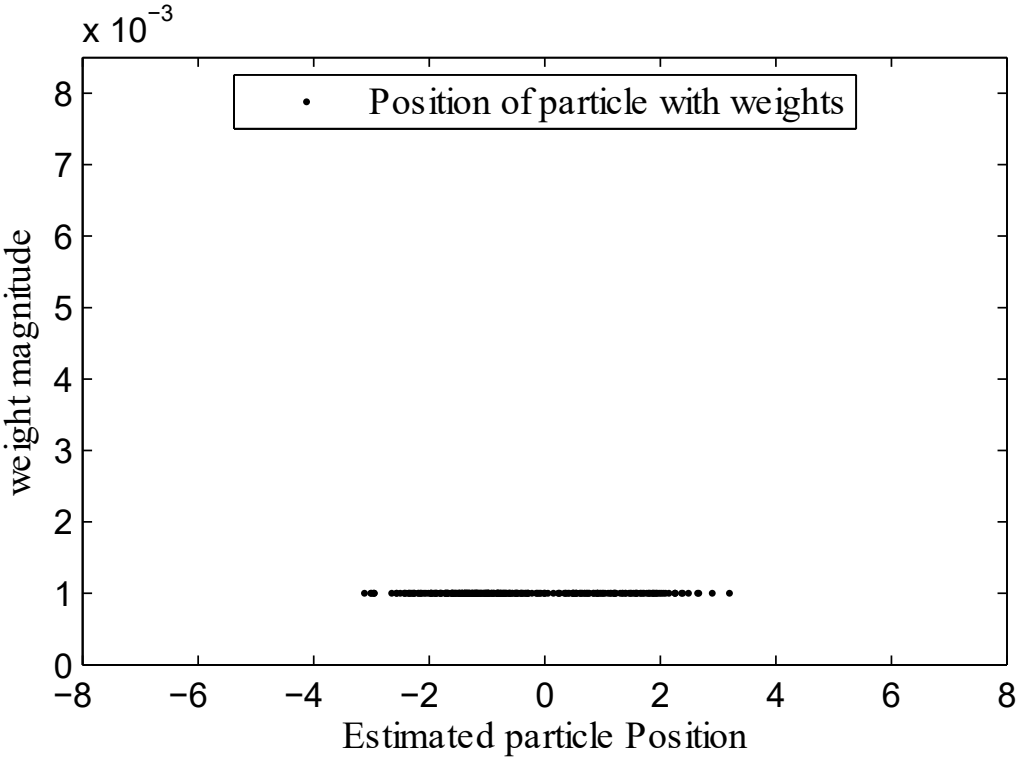

Figure 5: Figure Showing the distribution of the particle after resampling at time step t = 26.

due to the fact that each particle got reintialize to 1/M after resampling. The distribution show a horizontal line which is what is expected after resampling.

This is my C-code:

```
clear;

%load data
H = importdata('datafile.txt');
ydata = H(:,3);
trueposition = H(:,1);

%initialization
Qa = 0.0625;
xm1 = -10;
xm2 = 10;
Qn = 0.003906;
T = 1;

M = 1000;
E_xt = 0;
x_P = zeros(M,1);
x_P_update = zeros(M,1);
v_P = zeros(M,1);
v_P_update = zeros(M,1);
Y_t_m = zeros(M,1);
P_ytm = zeros(M,1);
w_P = (1/M)*ones(M,1);
w_P_update = (1/M)*ones(M,1);

for t = 1:numel(ydata)

%State Transition Equations
x_P_update = x_P + (v_P * T);
for i=1:M
if(x_P(i) < -20)
v_P_update(i) = 2;
elseif (x_P(i) >= -20 && x_P(i) < 0)
v_P_update(i) = v_P(i) + abs(normrnd(0,(Qa)));
elseif (x_P(i) >= 0 && x_P(i) <= 20)
v_P_update(i) = v_P(i) - abs(normrnd(0,(Qa)));
elseif (x_P(i)>20)
v_P_update(i) = -2;
```

```
end
end
x_P = x_P_update;
v_P = v_P_update;


p_1 = (1 /((sqrt(2*pi))*4));
p_2 = p_1 * exp ( (-1*((x_P - xm1).^2)) / (2 * 16) );
p_3 = p_1 * exp ( (-1*((x_P - xm2).^2)) / (2 * 16) );

%with these new updated particle locations, update the observations
%for each of these particles.
Y_t_m = p_2 + p_3;

q_1 = (1/((sqrt(2*pi))*0.003906));
P_ytm  = q_1*(exp((-(Y_t_m - ydata(t)).^2)/(2 * (2^-8)* (2^-8))));


%Using the new measurement vector yt
%the weight for each particle is updated:
w_P_update = w_P.*P_ytm;


% Normalize to form a probability distribution (i.e. sum to 1).
 nw_P_update = w_P_update ./ sum(w_P_update);

% Show distrubution of Particle with Weight before resampling
 figure(1)
    clf
    plot(x_P,nw_P_update,'.k')
    axis([-8,8,0,8.5*10^-3])
    xlabel('Estimated particle Position')
    ylabel('weight magnitude')
    legend('Position of particle with weights','Location','north')


 %Calculating Expected Values

 E_xt = sum(x_P .* nw_P_update);
 Ees_x(t) = E_xt;


% % %Check if sampling is necessary, and if so, resample
CV = (1/M) * sum(((M.*nw_P_update) - 1).^2);
    ESS = (M/(1 + CV));
```

```
if (ESS < (0.5*M))
 Q = cumsum(nw_P_update);
 T_1 = rand(M+1,1);
 T_2 = sort(T_1);
 T_2(M+1) = 1;
 a = 1; b = 1;
 while(a <= M)
     if(T_2(a)<Q(b))
         Index(a) = b;
         a = a + 1;
     else
         b = b + 1;
     end
 end


for a = 1:M

    x_P(a) = x_P(Index(a));
    v_P(a) = v_P(Index(a));
    nw_P_update(a) = 1/M;
end
end

% Show distrubution of Particle with Weight after resampling
 figure(2)
    clf
    plot(x_P,nw_P_update,'.k')
    axis([-8,8,0,8.5*10^-3])
    xlabel('Estimated particle Position')
    ylabel('weight magnitude')
    legend('Position of particle with weights','Location','north')


end


% %Displaying output
figure(3)
clf
tt = 1:1109;
plot(tt,trueposition,'b')
xlabel('Time Step')
ylabel('Position')
hold on
```

```
plot(tt,Ees_x,'--r')
legend({'Trueposition','Particle Filter'},'Location','north')
hold off


figure(4)
clf
tt = 1:1109;
plot(tt,ydata,'-r')
xlabel('Time Step')
ylabel('Measurement Position')
legend('Measurement Data','Location','north')
```

4 Conclusion

Using the Particle Filter to track the true position with measurement data turns out to be very effective. Once the noise in the measurement data is non-Gaussian and tractable, then the Particle Filter can be used. Therefore, the main reason for using the Particle Filter is because it can track non-linear and non-Gaussian objects, which is the case in this problem. Otherwise, other filtering technique such as Kalman Filter and Extended Kalman Filter would have been used. In figure 3 which is the Particle Filter best output that was obtain against the actual position. At the begin the tracking is out of phase, this could be due to the randomness of noise added each time the particles were propagated through the state transition equation. It was note that as the number of particles increase the Particle Filter converge more uniformly, but this took a lot more computational time. The resampling improve the tracking of the position. The resampling technique disregards particles that are not significant in tracking the position and just consider the particles that are contributing.