Particle Filter

ECE 8540: Lab 6

John Lawler

November 19, 2021

1 Introduction

This report considers a fundamental implementation of the particle filter. In systems that observe varied sinusoidal behavior, a particle filter may prove advantageous for non-linear data and hidden Markov models. These filters also perform well in systems where assumptions about the distribution of states or the state model may be unknown. This implementation considers the recorded data of a metal ball moving between two magnets. The succeeding sections outline the initialization, the update model, and the results obtained for these data.

2 Methods

The particle filter methodology follows the same structure as the Kalman filter and extended Kalman filter; however, instead of tracking a model utilizing one state matrix at a given time, the particle filter implements a set of weighted state matrices for tracking non-linear models referred to as particles.

The following 3 sections outline how each particle is initialized, how each particle's characteristics are propagated via a predict-update cycle similar to that of a Kalman filter, and lastly, how re-sampling is performed to avoid particle weights from approaching zero. The structure of how re-sampling is implementing in code is shown in appendix A.

2.1 Particle Initialization

The set of particles is represented as a set of two entities: the state variables, $x^{(m)}$, and the weighting for a given particle's state, $w^{(m)}$, shown in equation (1). The particle count, M, is set to 100 particles.

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

Equations (2) and (3) show what the state variables and weighting is initialized to for each particle prior to execution. The first value of the state matrix represents the position and the second value represents the velocity.

$$x^{(m)} = \begin{bmatrix} x_t^{(m)} \\ \dot{x}_t^{(m)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2}$$

$$w^{(m)} = \frac{1}{M} \tag{3}$$

Equation (4) below describes the state transition equations used. These equations are shown implemented in code under appendix A. On a given iteration, each particle's state variables are used as input to find x_{t+1}. The value a_t represents the dynamic noise from the previous and current state variable. Each value for a_t is drawn from a zero-mean Gaussian distribution $N(0, \sigma_a)$, where $\sigma_a = 2^{-4}$.

$$f(x_t, a_t) = \begin{bmatrix} x_{t+1} = x_t + \dot{x}_t T \\ \dot{x}_{t+1} = \begin{cases} 2 & \text{if } x_t < -20 \\ \dot{x}_t + |a_t| & \text{if } -20 \le x_t < 0 \\ \dot{x}_t - |a_t| & \text{if } 0 \le x_t \le 20 \\ -2 & \text{if } x_t > 20 \end{cases} \end{bmatrix} \tag{4}$$

2.2 Predict-Update Equations

The predict-update cycle begins with a for-loop that iterates for the amount of data provided. The first step uses each particle's state variables as input into the state transition equations from equation (4). Equation (5) shows each particle's previous state variable being inputted into the transition equation accompanied by a randomly distributed Gaussian value. The result is placed back into the respective particle's state variable matrix.

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

Once the predicated state is found for every given particle, the sensor value is retrieved. This value is used for updating the weights for each particle.

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

$$p(y_t|x_t^{(m)}) = \frac{1}{\sqrt{2\pi}\sigma_n}\exp(\frac{-(y_t^{(m)} - y_t)^2}{2\sigma_n^2}) \tag{7}$$

$$y_t^{(m)} = \frac{1}{\sqrt{2\pi}\sigma_m}\exp(\frac{-(x_t^{(m)} - x_{m1})^2}{2\sigma_m^2}) + \frac{1}{\sqrt{2\pi}\sigma_m}\exp(\frac{-(x_t^{(m)} - x_{m2})^2}{2\sigma_m^2}) \tag{8}$$

Equation (6) represents the weight update equation. The probability distribution is based upon the prior importance function, which is shown derived in equation (7). The ideal measurement used in the probability distribution function, y_t^m, is derived from equation (8), where x_t^{m1} and x_t^{m1} are -10 and 10, respectively.

The above three equations produce the non-normalized weights for a given sensor reading. These weights are calculated for every pixel for 1 to M. The weights are normalized such that they sum to 1 using equation (9) below.

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

The desired output is computed using the normalized weights in equation (10).

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

The expected value, or average amongst a set of M particles, is found for each recorded measurement. This equation is performed on both x_t^m and $\dot{x}_t^m$, the position and velocity states of a give particle. The expected values for these two state variables are plotted against time and their actual position and velocity data points under section 3.

After calculating expected values, re-sampling is performed if necessary. The process of re-sampling is described in section 2.3. Once performed, the for-loop increments 1 time unit and the aforementioned equations are recalculated for each sensor recording.

2.3 Particle Re-sampling

Re-sampling occurs when the effective sample size (ESS) cross the threshold set for it. Re-sampling helps avoid the problem of an increasing number of particles' weights approaching zero. As more particle weights approach zero, there becomes less particles used in correctly approximating the data. If ESS becomes lower than a threshold of $0.5 * M$, the particles and weights are considered for re-sampling. The algorithm for re-sampling is shown in appendix A. Below represents the equations for calculating ESS and the coefficient of variation (CV).

$$\text{ESS} = \frac{M}{1 + \text{CV}} \tag{11}$$

$$\text{CV} = \frac{\text{VAR}(w^{(m)})}{E^2[w^{(m)}]} = \frac{\frac{1}{M}\sum_{m=1}^{M}\left(w^{(m)} - \frac{1}{M}\sum_{m=1}^{M} w^{(m)}\right)^2}{\left(\frac{1}{M}\sum_{m=1}^{M} w^{(m)}\right)^2} = \frac{1}{M}\sum_{m=1}^{M}(M \cdot w^{(m)} - 1)^2 \tag{12}$$

The algorithm shown under appendix A for re-sampling follows a "select with replace-

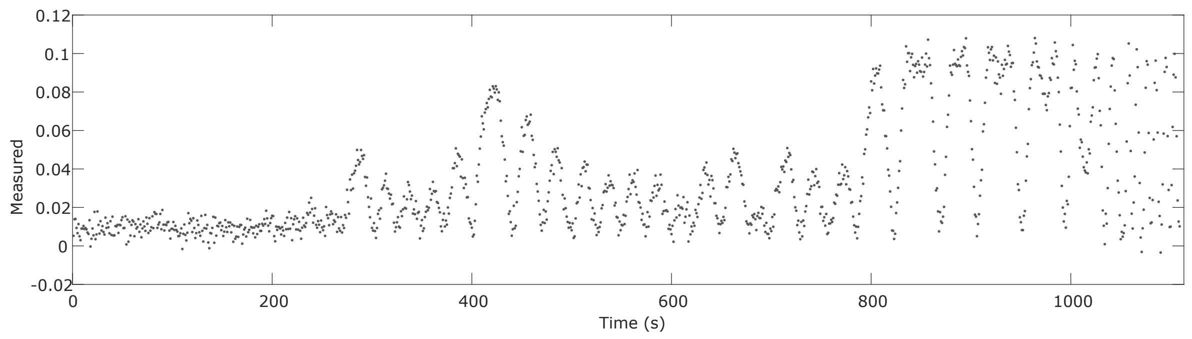

Figure 1: Recorded sensor data of the system.

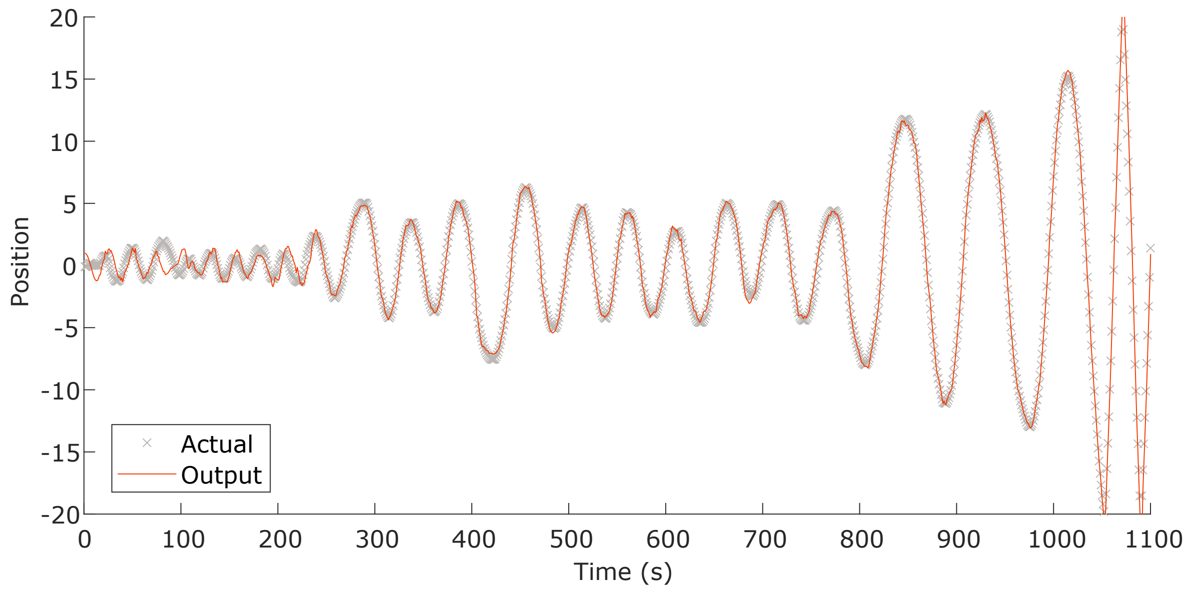

Figure 2: Filtered estimated output shown against the actual position.

ment" approach.

3 Results

Results shows the figures produced via particle filtering the wave data. The recorded measurements are shown in figure 1. Figure 2 shows the result of the particle filter in red versus the provided actual position data. Another observation to note is that, assuming due to the symmetrical property of the system, Matlab occasionally produced a plot of expected output data that perfectly reflected the actual data about the x-axis. After a few iterations of executing the code, it would swap back and forth from reflection to perfect alignment.

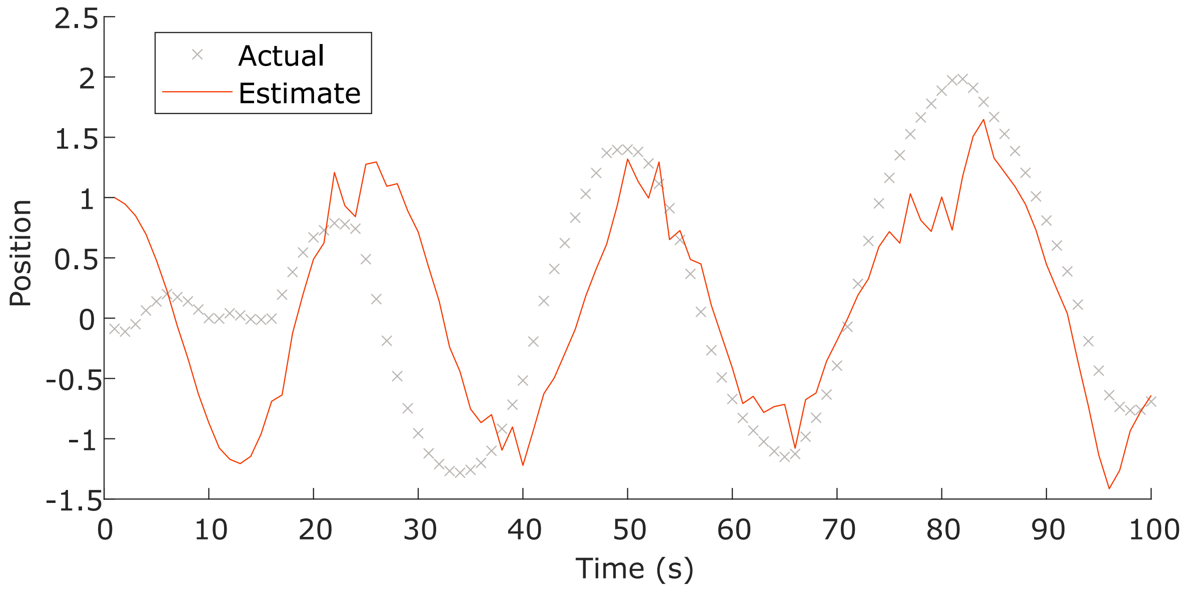

Figure 3: Output of position for time 0 to 100.

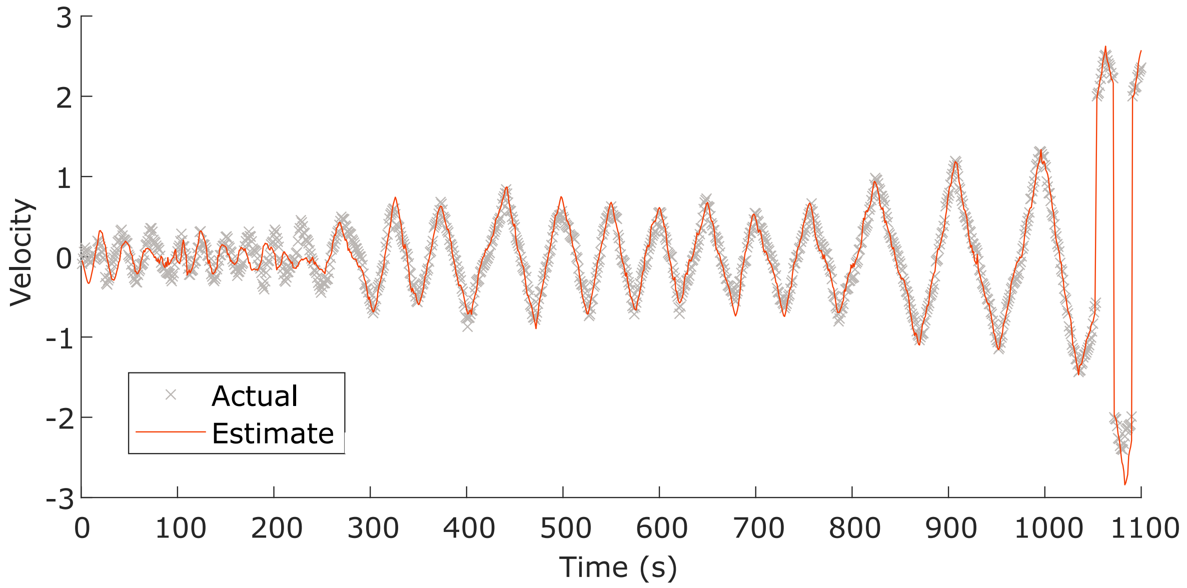

Figure 4: Output of estimated velocity shown against the actual velocity.

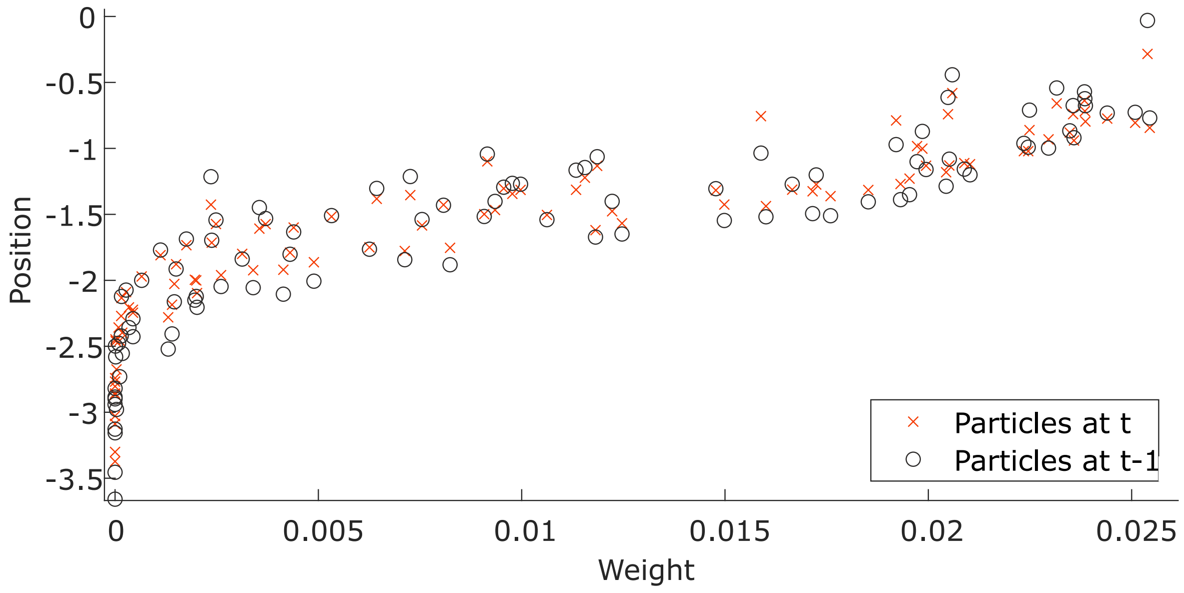

Figure 5: Output of the position of particles for a time t = 15.

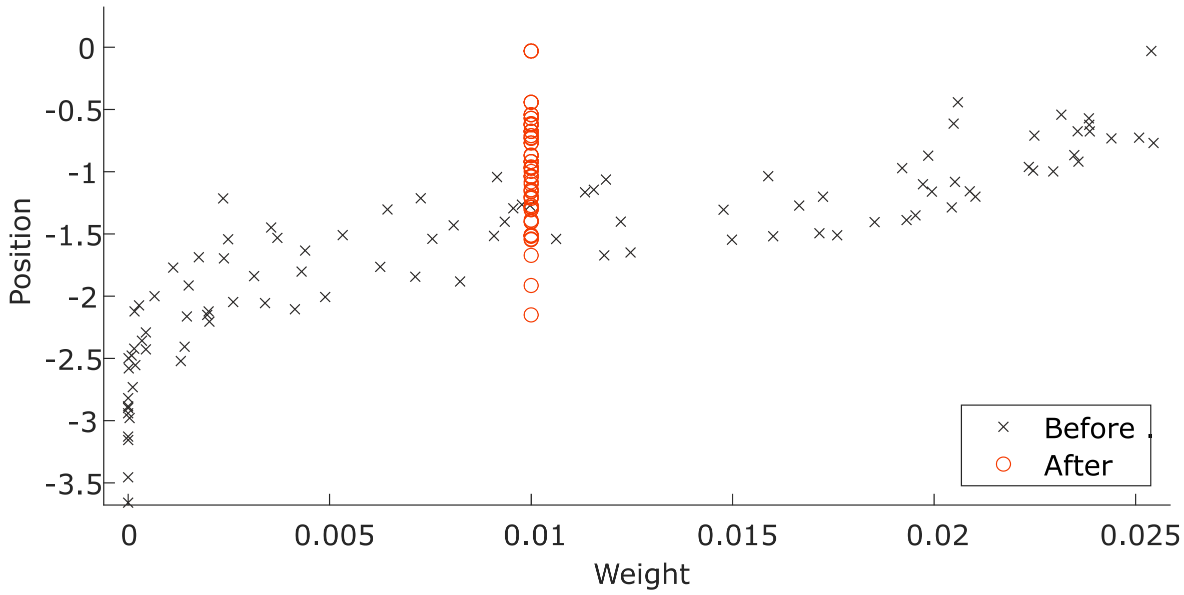

Figure 6: Output of the result of re-sampling the particles at time t = 15.

4 Discussion

As a result, the particle filter performs quite well at accurately predicting varied sinusoidal data. Figure 1 shows a perfect match up, past a time of $t = 200$, of the actual position data and the expected position data found from equation (10). Figure 3 shows the results of figure 2 from a time t of 0 to 100. Figure 4 shows the expected output data for velocity against the provided actual velocity data.

The particle data shown in figure 5 shows particles at a time of $t-1 = 14$ and $t = 15$. This graph provides an interesting look at the behavior of particles between a given time slice. For a given particle, it can be observed that there is a greater difference in a particles before and after position versus its respective weight. The graph of particles chosen at this time is due to also requiring the need to re-sample; this result is shown in figure 6. Another observation to make concerns the large concentration of particles with their weights at zero in the bottom left of figure 5; this induces the need to re-sample and resets all of the particles' weights back to a value of $\frac{1}{M} = 0.01$.

5 Conclusion

As shown in the results from section 3, the filter's use of particle sets provides a very accurate estimation of varied sinusoidal measurement data in a given system. By marginally adjusting various parameters in the model, the filter may accurately predict all types of varied system response data. Particle filters aim to compute the distribution of unknown quantities when provided segmented or noisy data depicted in the measurement data.

A

Matlab Code

- Matlab code for implementing the state transition equations

```
function Xt = state_eq(xt, xd, at)
    T = 1;
    xtp1 = @(xt, xd) (xt + xd*T);


    if xt < -20
        xdp1 = 2;
    elseif (-20 <= xt) && (xt < 0)
        xdp1 = xd + abs(at);
    elseif (0 <= xt) && (xt <= 20)
        xdp1 = xd - abs(at);
    elseif xt > 20
        xdp1 = -2;
    end


    Xt = [xtp1(xt, xd); xdp1];
end
```

- Code for implementing the re-sampling algorithm

```
    if ESS < 0.5*M
      pstates = particles(:,1);
      Q = cumsum(weights);
      t=rand(M+1);
      T=sort(t);
      T(M+1)=1.0;
    k=1;
    r=1;
    while (k <= M)
        if T(k) < Q(r)
            Index(k)=r;
            k=k+1;
        else
            r=r+1;
        end
    end
    for c=1:M
       newP = cell2mat(pstates(Index(c)));
       newW = 1/M;

       particles(c, :) = {newP, newW};
    end

    % Record points for plotting
    if i == get_plot_points
       pts_r = particles(:, 1);
       wts_r = particles(:, 2);
    end
end
```