Homework 8 Report

AERO626, Spring 2023

Name: David van Wijk UIN: 932001896

Problem 1

In order to implement the extended Kalman Filter (EKF), the state and measurement Jacobians must first be computed. Using a first order Taylor series expansion about the mean, these Jacobians can be computed as follows:

$$\boldsymbol{F}_x(t) = \frac{\partial \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{w}(t))}{\partial \boldsymbol{x}(t)}\bigg|_{\substack{\boldsymbol{x}(t)=\boldsymbol{m}_x(t) \\ \boldsymbol{w}(t)=\boldsymbol{0}_w}} \tag{1a}$$

$$\boldsymbol{F}_w(t) = \frac{\partial \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{w}(t))}{\partial \boldsymbol{w}(t)}\bigg|_{\substack{\boldsymbol{x}(t)=\boldsymbol{m}_x(t) \\ \boldsymbol{w}(t)=\boldsymbol{0}_w}} \tag{1b}$$

$$\boldsymbol{H}_x = \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{v}(t))}{\partial \boldsymbol{x}(t)}\bigg|_{\substack{\boldsymbol{x}(t)=\boldsymbol{m}_x(t) \\ \boldsymbol{v}(t)=\boldsymbol{0}_v}} \tag{2a}$$

$$\boldsymbol{H}_v = \frac{\partial \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{v}(t))}{\partial \boldsymbol{v}(t)}\bigg|_{\substack{\boldsymbol{x}(t)=\boldsymbol{m}_x(t) \\ \boldsymbol{v}(t)=\boldsymbol{0}_v}} \tag{2b}$$

where $\boldsymbol{f}(\boldsymbol{x}(t))$ describes the dynamics of the system, and $\boldsymbol{h}(\boldsymbol{x}(t))$ is the measurement function. For this specific problem,

$$\boldsymbol{F}_x = 1 - .01\cos\left(x_{k-1}\right) \tag{3a}$$

$$\boldsymbol{F}_w = 1 \tag{3b}$$

$$\boldsymbol{H}_x = \cos\left(2x_k\right) \tag{4a}$$

$$\boldsymbol{H}_v = 1 \tag{4b}$$

In Figure 1, the state estimation error of the EKF versus measurement number is plotted. The 3σ interval of state covariance is also plotted. Note that both the prior and posterior errors and covariances are plotted. The state error remains within the 3σ interval for the entirety of the simulation run. While it is acceptable for the state error to occasionally walk outside of the 3σ interval, doing so consistently is a sign that the estimator may not be performing well. However, this is not a concern here.

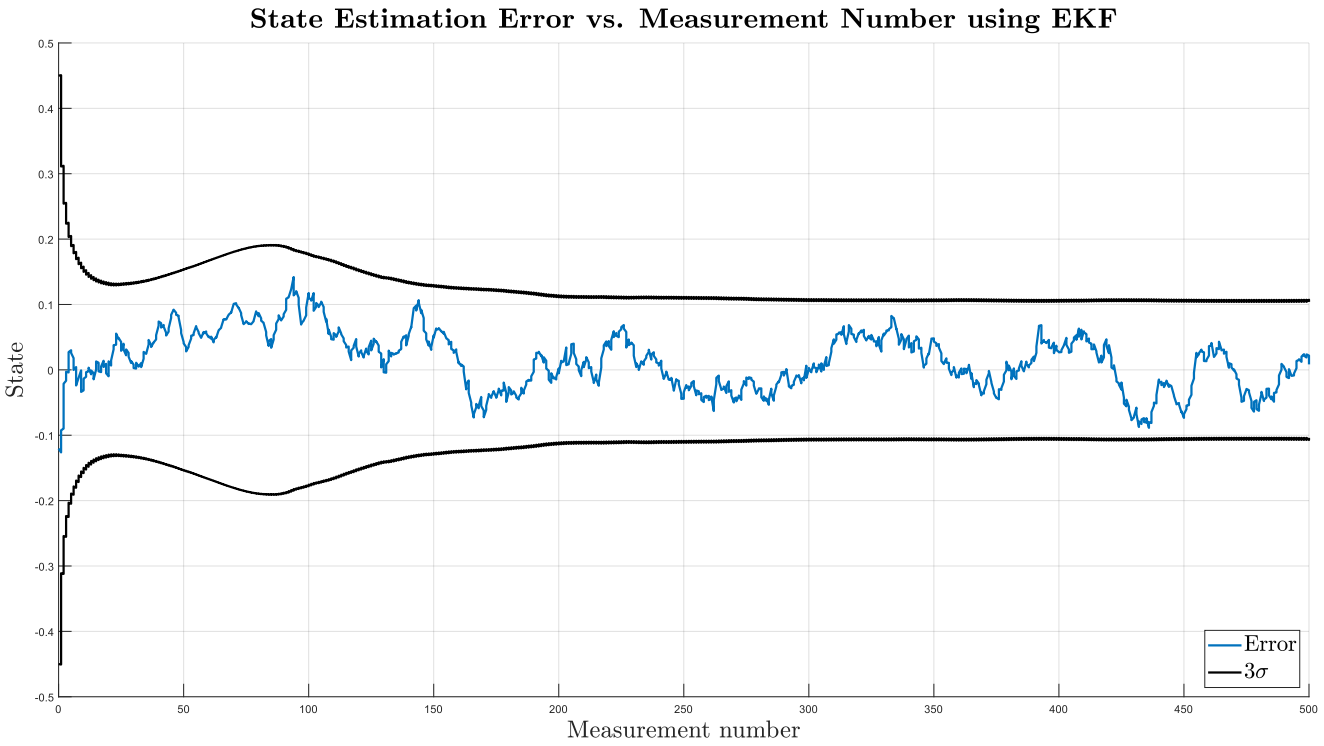

Figure 1: State Prior and Posterior Estimation Error versus Measurement Number using EKF.

In Figure 2, the innovations are plotted, along with the 3σ intervals of innovation covariance and measurement noise covariance. As expected, the innovation covariance and the measurement noise covariance approach very similar values as all the data is processed.

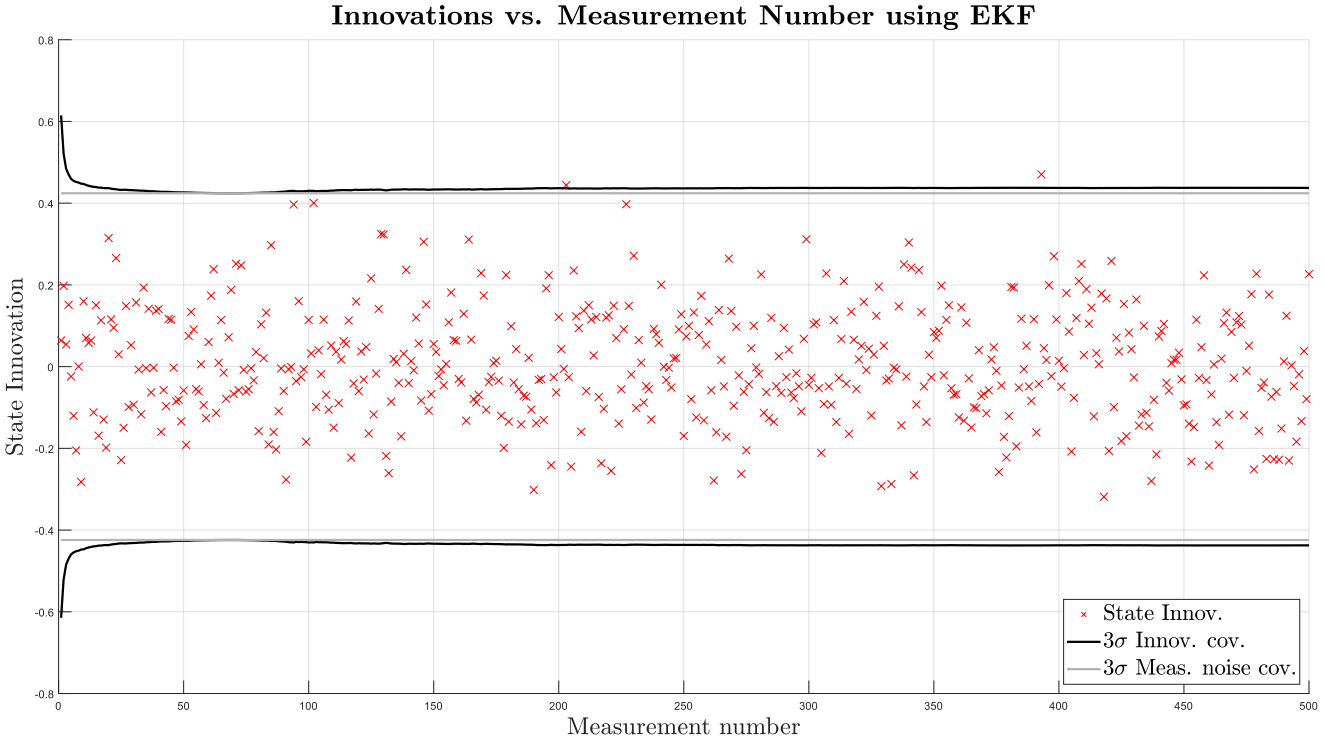

Figure 2: State innovations versus Measurement Number using EKF.

Problem 2

In Figure 3, the state estimation error of the bootstrap particle filter (BPF) without resampling versus measurement number is plotted. The 3σ interval of state covariance is also plotted. Only the posteriors of the state error and covariance are plotted. Again, the state error remains within the 3σ interval for the entirety of the process. In contrast to the EKF, the covariance is much more jagged. This makes sense because as can be seen in Figure 4, the number of effective particles ($\hat{N}_{\text{eff}}$) drops off very quickly. The result of this is that the state estimate is captured with a much smaller number of points, causing the total covariance to be highly variable for each step.

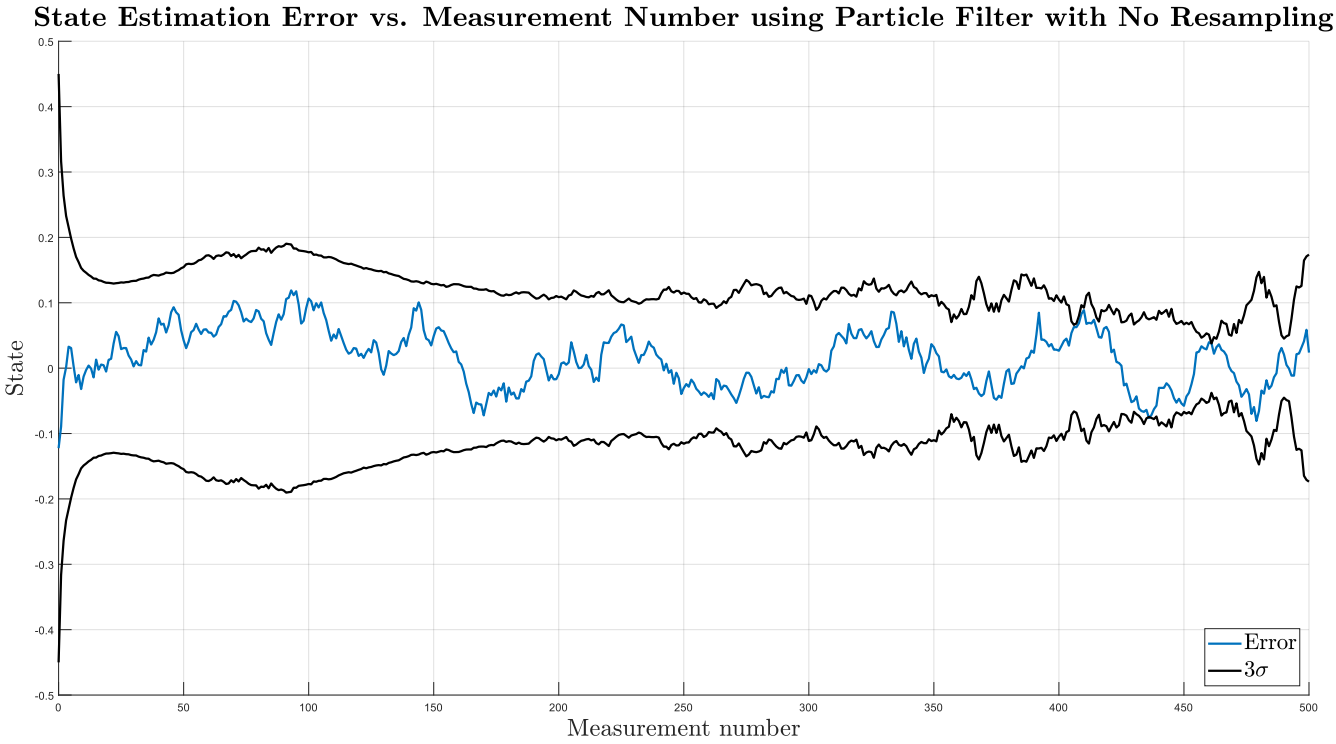

Figure 3: State Posterior Estimation Error versus Measurement Number using BPF with no Resampling.

As mentioned above, in Figure 4, $\hat{N}_{\text{eff}}$ is reduced by over half nearly instantly, and continues to decrease as more measurements are processed. This demonstrates the degeneracy problem of a BPF without resampling. Using the first few measurements, the filter becomes very confident in a sliver of particles, after which it is impossible for the filter to recover without making the process noise extremely high. In this particular example, the BPF without resampling works fairly well nonetheless.

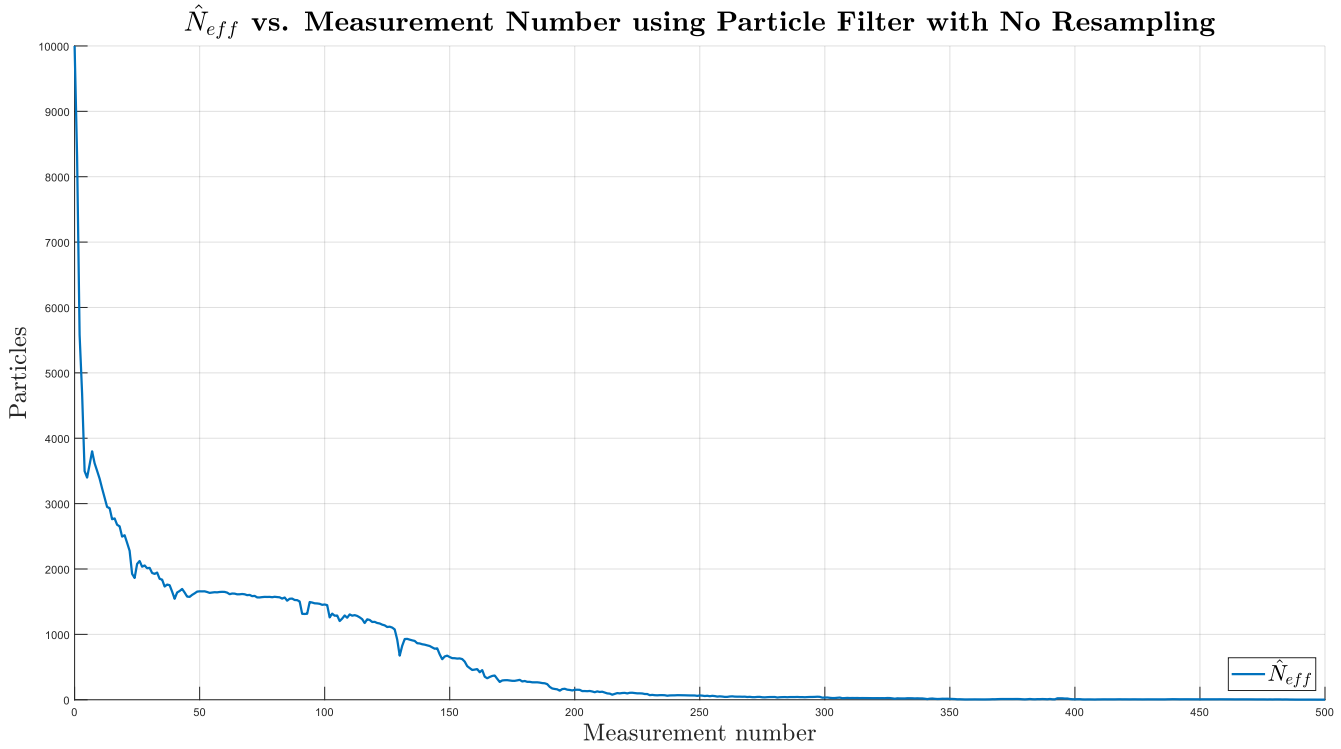

Figure 4: Number of effective particles versus Measurement Number using BPF with no Resampling.

Problem 3

In Figure 5, the state estimation error of the bootstrap particle filter (BPF) with basic resampling versus measurement number is plotted. The 3σ interval of state covariance is also plotted. Only the posterior state error and covariance is plotted. Again, the state error remains within the 3σ interval for the entirety of the process. In contrast to the EKF, the covariance is still more jagged, but is much smoother than for the BPF without resampling. Again, this makes intuitive sense. When basic resampling occurs, the number of effective points is ramped up, causing the covariance to be represented by more particles, leading to a smoother curve. Indeed, using Figure 6, it can be seen that the most "jagged" region of the covariance occurs between approximately measurement $k = 25$, and $k = 125$, when the number of particles slowly drops to slightly above the resampling threshold.

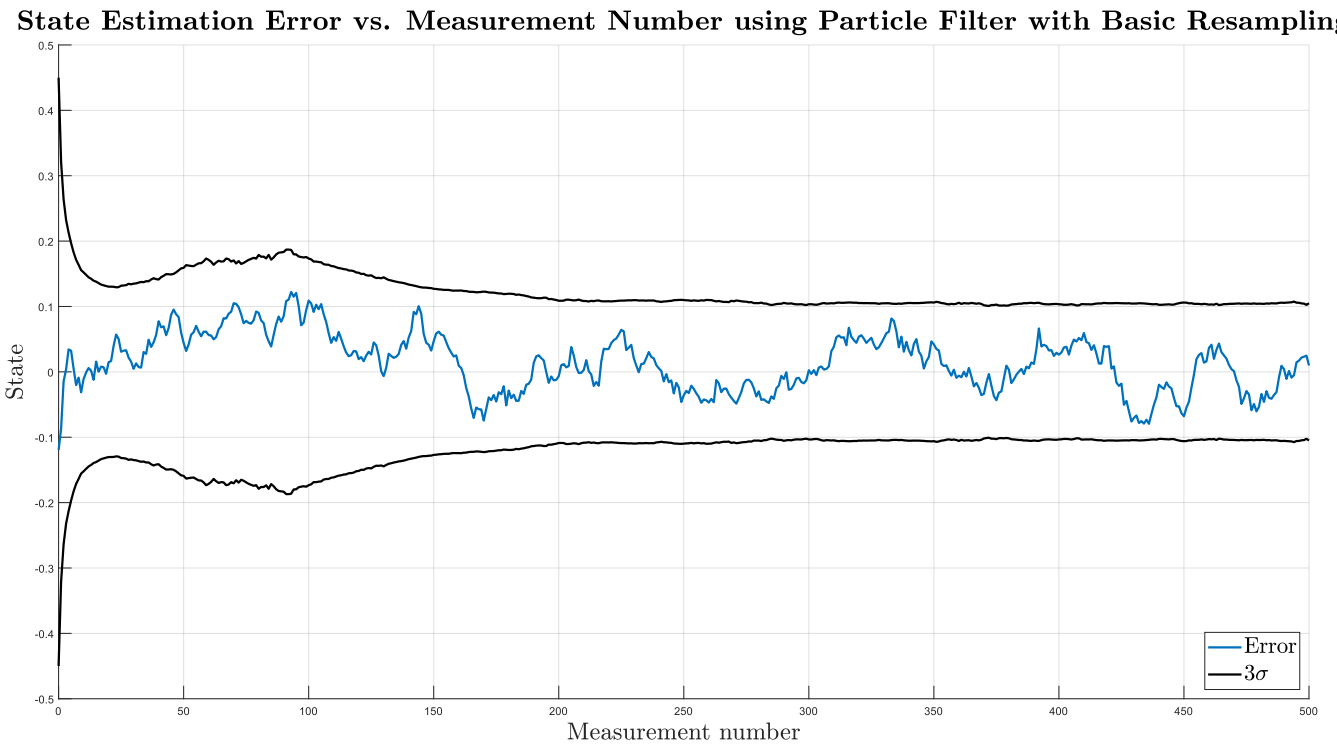

Figure 5: State Posterior Estimation Error versus Measurement Number using BPF with Basic Resampling.

In Figure 6 the number of effective particles ($\hat{N}_{\text{eff}}$) versus measurement number is plotted. As expected, the plot is very similar to Figure 4 until the first resampling procedure is executed. At approximately k = 130, the number of effective particles dips below 1000, triggering the first resampling procedure. The number of effective particles then spikes due to the resampling, and the process of particle degeneracy followed by resampling continues. Throughout the process, basic resampling occurs six times.

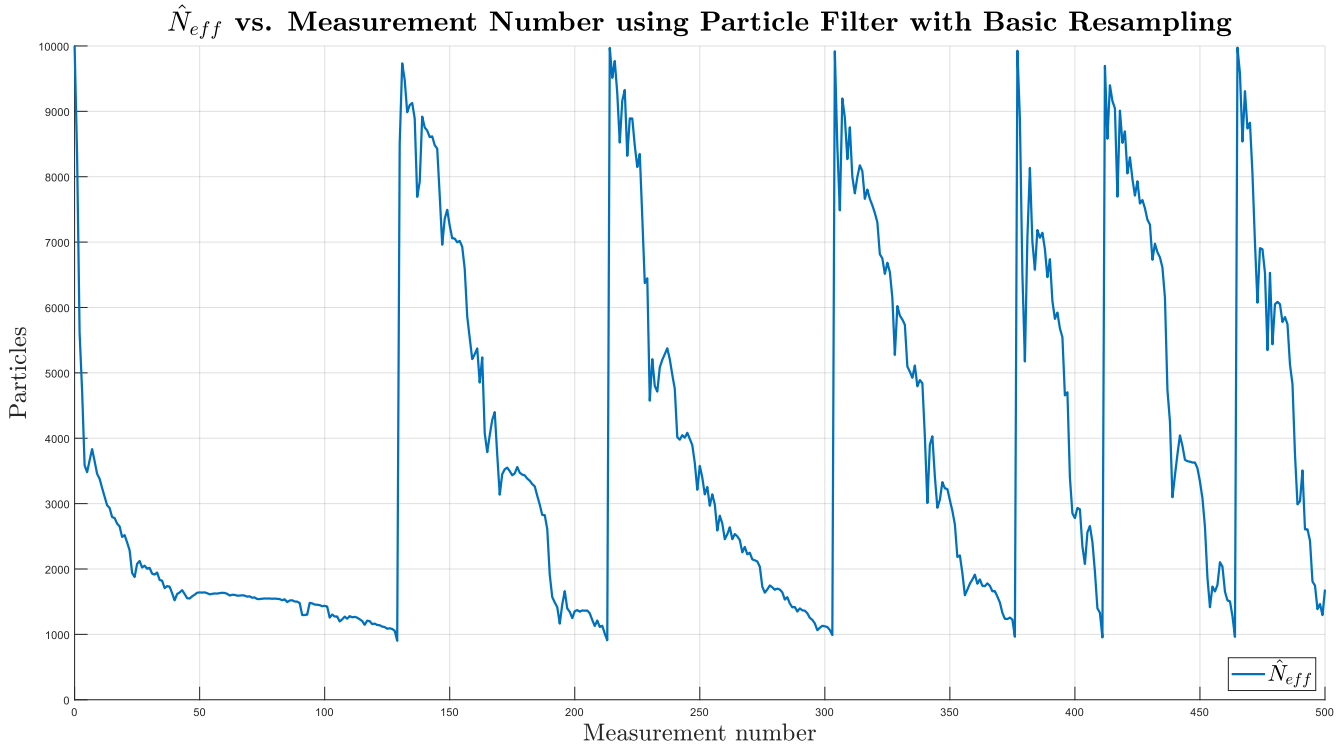

Figure 6: Number of effective particles versus Measurement Number using BPF with Basic Resampling.

Problem 4

In Table 1, the results of each filter are compared by using the root-mean-square of the error (RMSE), and the mean absolute error (MAE) using the posterior estimation errors. Here, BPF1 refers to the BPF without resampling, and BPF2 refers to the BPF with a basic resampling procedure. The results of all three filters are both visually quite similar, and are also quite similar using RMSE and MAE. From this single run, BPF2 does slightly better than the EKF, with a lower RMSE and MAE. Both the EKF and BPF2 do better than the BPF1. This is not unexpected, as the BPF1 experiences particle degeneracy early on in the estimation process and does not recover which means it attempts to estimate the state using only a handful of points. A simple analysis of computation time was done as well, demonstrating the computational efficiency that the EKF provides. While particle filters require no linearization and can produce good state estimates, they are extremely costly in their most basic form.

Table 1: Filter Comparison

Filter Variation	**RMSE**	**MAE**	**Computation Time [sec]**
EKF	.0455	.0373	0.0218
BPF1	.0461	.0381	1.5416
BPF2	.0453	.0372	2.018

A Matlab Code

```
%% AERO 626 Homework 8
%
%   Texas A&M University
%   Aerospace Engineering
%   van Wijk, David

close all; clear; clc;

plot_flag = false;

% Seed to reproduce results
% rng(100)

xaxis_sz = 22; yaxis_sz = 22; legend_sz = 20;
data = load('data_HW08.mat');

%% Part 1 - EKF Implementation

tic

Pww  = .01^2;
Pvv  = .02;
mx0  = 1.5;
Pxx0 = .15^2;
numPts = length(data.xk);

% Propagate the truth
x0 = data.x0;
x_truth = data.xk';

opts = odeset('AbsTol',1e-9,'RelTol',1e-9);

% NOTATION:
%   tkm1   = t_{k-1}         time at the (k-1)th time
%   mxkm1  = m_{x,k-1}^{+}   posterior mean at the (k-1)th time
%   Pxxkm1 = P_{xx,k-1}^{+}  posterior covariance at the (k-1)th time
%   tk     = t_{k}           time at the kth time
%   mxkm   = m_{x,k}^{-}     prior mean at the kth time
%   Pxxkm  = P_{xx,k}^{-}    prior covariance at the kth time
%   mxkp   = m_{x,k}^{+}     posterior mean at the kth time
%   Pxxkp  = P_{xx,k}^{+}    posterior covariance at the kth time
%   Pzzkm  = P_{zz,k}^{-}    innovation covariance at the kth time
%   Pvvk   = P_{vv,k}        measurement noise covariance at the kth time

% declare storage space for saving state estimation error information
xcount  = 0;
xstore  = nan(1,2*numPts-1);
kxstore = nan(1,2*numPts-1);
mxstore = nan(1,2*numPts-1);
exstore = nan(1,2*numPts-1);
sxstore = nan(1,2*numPts-1);
```

```
% declare storage space
zcount  = 0;
zstore  = nan(1,numPts);
kzstore = nan(1,numPts);
mzstore = nan(1,numPts);
ezstore = nan(1,numPts);
dzstore = nan(1,numPts);
dxstore = nan(1,numPts);
szstore = nan(1,numPts);
svstore = nan(1,numPts);

% store initial data
xcount            = xcount + 1;
kxstore(:,xcount) = 0;
xstore(:,xcount)  = x0;
mxstore(:,xcount) = mx0;
sxstore(:,xcount) = sqrt(diag(Pxx0));
exstore(:,xcount) = data.x0 - mx0;

% measurement noise
Hv = 1;

% process noise
Fw = 1;

% initialize time, mean, and covariance for the EKF
mxkm1  = mx0;
Pxxkm1 = Pxx0;

% loop over the number of data points
for k = 1:numPts
    zk   = data.zk(k);     % current measurement to process
    Pvvk = Pvv;            % measurement noise covariance

    % unpack the truth -- this cannot be used in the filter, only for
       analysis
    xk = x_truth(k,:);       % true state

    % propagate the mean and covariance
    mxkm = recursivePropSingle(mxkm1,0);
    Fx = stateJacobianMean(mxkm1);
    Pxxkm = Fx*Pxxkm1*Fx' + Fw*Pww*Fw';

    % store a priori state information for analysis
    xcount            = xcount + 1;
    xstore(:,xcount)  = xk;
    kxstore(:,xcount) = k;
    mxstore(:,xcount) = mxkm;
    exstore(:,xcount) = xk - mxkm;
    sxstore(:,xcount) = sqrt(diag(Pxxkm));

    % compute the estimated measurement
```

```
    mzkm = measurementFunSingle(mxkm,0);

    % compute the measurement Jacobian
    Hxk = measurementJacobianMean(mxkm);

    % update the mean and covariance
    Pxzkm = Pxxkm*Hxk';
    Pzzkm = Hxk*Pxxkm*Hxk' + Hv*Pvvk*Hv';
    Kk = Pxzkm/Pzzkm;
    mxkp = mxkm + Kk*(zk - mzkm);
    Pxxkp = Pxxkm - Pxzkm*Kk' - Kk*(Pxzkm)' + Kk*(Pzzkm)*Kk';

    % store a posteriori state information for analysis
    xcount            = xcount + 1;
    xstore(:,xcount)  = xk;
    kxstore(:,xcount) = k;
    mxstore(:,xcount) = mxkp;
    exstore(:,xcount) = xk - mxkp;
    sxstore(:,xcount) = sqrt(diag(Pxxkp));

    % store measurement information for analysis
    zcount            = zcount + 1;
    zstore(:,zcount)  = zk;
    kzstore(:,zcount) = k;
    mzstore(:,zcount) = mzkm;
    ezstore(:,zcount) = zk - mzkm;
    dzstore(:,zcount) = (zk - mzkm)'*(Pzzkm\(zk - mzkm));
    dxstore(:,zcount) = (xk - mxkp)'*(Pxxkp\(xk - mxkp));
    szstore(:,zcount) = sqrt(diag(Pzzkm));
    svstore(:,zcount) = sqrt(diag(Pvvk));

    % cycle the time, mean, and covariance for the next step of the EKF
    mxkm1  = mxkp;
    Pxxkm1 = Pxxkp;
end

toc

% dxstoreEKF = dxstore;
% rmsEKF     = sqrt((sum(ezstore(1:end-1).^2))/length(ezstore(1:end-1)));
MAE_EKF  = mean(abs(exstore));
RMSE_EKF = rms(exstore);

if plot_flag
    % Plot Innovations
    plotPart1_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
       legend_sz) %#ok<*UNRCH>

    % Plot Estimation Error
    plotPart1_EstimationError(exstore,sxstore,numPts,xaxis_sz,yaxis_sz,
       legend_sz)
end
```

```
%% Part 2 - Bootstrap Particle Filter (no resampling)

tic

% declare storage space for saving state estimation error information
xcount    = 0;
xstore    = nan(1,numPts+1);
kxstore   = nan(1,numPts+1);
mxstore   = nan(1,numPts+1);
exstore   = nan(1,numPts+1);
sxstore   = nan(1,numPts+1);
Neffstore = nan(1,numPts+1);

N = 10000;

% sample N particles from Gaussian (mx0, Pxx0)
[particles_m, weights_m]   = sampleInitialParticles(mx0,Pxx0,N);
[mxk0] = calcWMeanParticles(particles_m,weights_m,N);

% calc Neff
[Neff]  = calcNeff(weights_m);

% store initial data
xcount              = xcount + 1;
kxstore(:,xcount)   = 0;
xstore(:,xcount)    = x0;
mxstore(:,xcount)   = mxk0;
sxstore(:,xcount)   = sqrt(diag(Pxx0));
exstore(:,xcount)   = data.x0 - mxk0;
Neffstore(:,xcount) = Neff;

% process all measurements
for k = 1:numPts
    zk   = data.zk(k);     % current measurement to process

    % unpack the truth -- this cannot be used in the filter, only for
       analysis
    xk = x_truth(k,:);       % true state

    % propagate particles
    [particles_p] = propParticles(particles_m,Pww);

    % calculate the weights based on true measurements vs. expected
    [weights_p] = calcWeights(particles_p,weights_m,zk,Pvv);

    [Neff] = calcNeff(weights_p);

    [mxkp] = calcWMeanParticles(particles_p,weights_p,N);

    % posterior storing
    xcount            = xcount + 1;
    xstore(:,xcount)  = xk;
    kxstore(:,xcount) = k;
```

```
    mxstore(:,xcount) = mxkp;
    exstore(:,xcount) = xk - mxkp;
    sxstore(:,xcount) = sqrt(sum(weights_p.*(particles_p-mxkp).^2));
    Neffstore(:,xcount) = Neff;

    % initialize for next step
    particles_m = particles_p;
    weights_m   = weights_p;

end

toc

MAE_PF1  = mean(abs(exstore));
RMSE_PF1 = rms(exstore);

if plot_flag
    plotPart2_EstimationError(exstore,sxstore,numPts,xaxis_sz,yaxis_sz,
       legend_sz)
    plotPart2_Neff(Neffstore,numPts,xaxis_sz,yaxis_sz,legend_sz)
end

%% Part 3 - Bootstrap Particle Filter (with resampling)

tic

% declare storage space for saving state estimation error information
xcount    = 0;
xstore    = nan(1,numPts+1);
kxstore   = nan(1,numPts+1);
mxstore   = nan(1,numPts+1);
exstore   = nan(1,numPts+1);
sxstore   = nan(1,numPts+1);
Neffstore = nan(1,numPts+1);

N = 10000;

% sample N particles from Gaussian (mx0, Pxx0)
[particles_m, weights_m]   = sampleInitialParticles(mx0,Pxx0,N);
[mxk0] = calcWMeanParticles(particles_m,weights_m,N);

% calc Neff
[Neff]  = calcNeff(weights_m);
Nthresh = .1*N;

% store initial data
xcount              = xcount + 1;
kxstore(:,xcount)   = 0;
xstore(:,xcount)    = x0;
mxstore(:,xcount)   = mxk0;
sxstore(:,xcount)   = sqrt(diag(Pxx0));
exstore(:,xcount)   = data.x0 - mxk0;
Neffstore(:,xcount) = Neff;
```

```
resampling_counter  = 0;

% process all measurements
for k = 1:numPts
    zk   = data.zk(k);     % current measurement to process

    % unpack the truth -- this cannot be used in the filter, only for
       analysis
    xk = x_truth(k,:);       % true state

    % propagate particles
    [particles_p] = propParticles(particles_m,Pww);

    % calculate the weights based on true measurements vs. expected
    [weights_p] = calcWeights(particles_p,weights_m,zk,Pvv);

    [Neff] = calcNeff(weights_p);
    if Neff < Nthresh
        [particles_p,weights_p] = basicResampling(particles_p,weights_p,N
           );
        resampling_counter = resampling_counter + 1;
    end

    [mxkp] = calcWMeanParticles(particles_p,weights_p,N);

    % posterior storing
    xcount             = xcount + 1;
    xstore(:,xcount)   = xk;
    kxstore(:,xcount) = k;
    mxstore(:,xcount) = mxkp;
    exstore(:,xcount) = xk - mxkp;
    sxstore(:,xcount) = sqrt(sum(weights_p.*(particles_p-mxkp).^2));
    Neffstore(:,xcount) = Neff;

    % initialize for next step
    particles_m = particles_p;
    weights_m    = weights_p;

end

toc

MAE_PF2  = mean(abs(exstore));
RMSE_PF2 = rms(exstore);

disp(' ')
disp(['Resampling was performed ', num2str(resampling_counter), ' times'
   ])

if plot_flag
    plotPart3_EstimationError(exstore,sxstore,numPts,xaxis_sz,yaxis_sz,
       legend_sz)
    plotPart3_Neff(Neffstore,numPts,xaxis_sz,yaxis_sz,legend_sz)
```

```
end

%% Part 4 - Filter Comparison

disp('EKF: RMSE  |  MAE')
disp([RMSE_EKF MAE_EKF])

disp('PF1: RMSE  |  MAE')
disp([RMSE_PF1 MAE_PF1])

disp('PF2: RMSE  |  MAE')
disp([RMSE_PF2 MAE_PF2])

%% Dynamics Functions

function [Fx] = stateJacobianMean(xkminus1)
Fx = 1 - .01*cos(xkminus1);
end

function [Hx] = measurementJacobianMean(xk)
Hx = cos(2*xk);
end

function [zk] = measurementFunSingle(xk,Pvv)
% Generate measurment given xk

vk = randn*sqrt(Pvv);
zk = .5*sin(2*xk) + vk;

end

function [xk] = recursivePropSingle(x0,Pww)
% Recursively propagate the state

xkminus1 = x0;
wkminus1 = randn*sqrt(Pww);
xk = xkminus1 - .01*sin(xkminus1) + wkminus1;

end

%% Particle Filter Functions

function [particles,weights] = sampleInitialParticles(mx0,Pxx0,N)

particles = zeros(1,N);
for i = 1:N
    x_i = mx0 + randn*sqrt(Pxx0);
    particles(:,i) = x_i;
end
weights = ones(1,N)/N;

end
```

```
function [particles_p] = propParticles(particles_m,Pww)

N = length(particles_m);
particles_p = zeros(size(particles_m));
for i = 1:N
    particles_p(:,i) = recursivePropSingle(particles_m(:,i),Pww);
end

end

function [weights_p] = calcWeights(particles_p,weights_m,zk,Pvv)

N = length(weights_m);
weights_p = zeros(size(weights_m));
for i = 1:N
    xk        = particles_p(:,i);
    zk_tilde  = measurementFunSingle(xk,0);
    wkm1      = weights_m(:,i);
    pw        = abs(2*pi*Pvv)^(-.5)*exp(-.5*(zk - zk_tilde)'*Pvv^-1*(zk -
       zk_tilde));
    % pw        = normpdf(zk - zk_tilde,0,sqrt(Pvv));
    weights_p(:,i) = wkm1*pw;
end
weights_p = weights_p/sum(weights_p);

end

function [mxk] = calcWMeanParticles(particles,weights,N)

mxk = 0;
for i = 1:N
    xk = particles(:,i);
    wk = weights(:,i);
    mxk = mxk + xk*wk;
end

end

function [Neff] = calcNeff(weights)

Neff = 1/sum(weights.^2);

end

function [particles,weights] = basicResampling(particles_k,weights_k,N)

c = cumsum(weights_k);
particles = zeros(size(particles_k));
for i = 1:N
    u = rand;
    j = find(u <= c,1,'first');
    particles(:,i) = particles_k(:,j);
end
```

```
weights = ones(1,N)/N;

end

%% Plotting Functions

function plotPart1_Innovations(ezstore,szstore,svstore,xaxis_sz,yaxis_sz,
    legend_sz)
measx = 1:length(ezstore);
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];
ez_scatter_opts = {100,'x','r'};

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{Innovations vs. Measurement Number using EKF}','Fontsize'
    ,25,'interpreter','latex')
a1 = scatter(measx,ezstore(1,:),ez_scatter_opts{:});
b1 = plot(measx,std_plot*szstore(1,:),'-','Color','k','LineWidth',2,'
    MarkerSize',20);
b2 = plot(measx,std_plot*svstore(1,:),'-','Color',[.7 .7 .7],'LineWidth'
    ,2,'MarkerSize',20);
plot(measx,-std_plot*szstore(1,:),'-','Color','k','LineWidth',2,'
    MarkerSize',20);
plot(measx,-std_plot*svstore(1,:),'-','Color',[.7 .7 .7],'LineWidth',2,'
    MarkerSize',20);
ylabel('State Innovation','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legendtxt = {'State Innov.', [txt  ' Innov. cov.'],[txt ' Meas. noise cov
    .']};
legend([a1 b1 b2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')
end

function plotPart1_EstimationError(exstore,sxstore,numPts,xaxis_sz,
    yaxis_sz,legend_sz)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',2};
std_line_opts = {'-','LineWidth',2,'Color','k'};

measx1 = sort([measx measx]); measx1 = [0 measx1];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{State Estimation Error vs. Measurement Number using EKF}',
    'Fontsize',25,'interpreter','latex')
a1 = plot(measx1,exstore(1,:),err_line_opts{:});
a2 = plot(measx1,std_plot*sxstore(1,:),std_line_opts{:});
plot(measx1,-std_plot*sxstore(1,:),std_line_opts{:})
ylabel('State','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legendtxt = {'Error',txt};
legend([a1 a2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')
```

```
end

function plotPart2_EstimationError(exstore,sxstore,numPts,xaxis_sz,
    yaxis_sz,legend_sz)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',2};
std_line_opts = {'-','LineWidth',2,'Color','k'};

measx1 = [0 measx];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{State Estimation Error vs. Measurement Number using
    Particle Filter with No Resampling}','Fontsize',25,'interpreter','
    latex')
a1 = plot(measx1,exstore(1,:),err_line_opts{:});
a2 = plot(measx1,std_plot*sxstore(1,:),std_line_opts{:});
plot(measx1,-std_plot*sxstore(1,:),std_line_opts{:})
ylabel('State','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legendtxt = {'Error',txt};
legend([a1 a2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
    location','southeast')

end

function plotPart2_Neff(Neffstore,numPts,xaxis_sz,yaxis_sz,legend_sz)
measx = 1:numPts;

err_line_opts = {'-','LineWidth',2};

measx1 = [0 measx];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{$\hat{N}_{eff}$ vs. Measurement Number using Particle
    Filter with No Resampling}','Fontsize',25,'interpreter','latex')
a1 = plot(measx1,Neffstore(1,:),err_line_opts{:});
ylabel('Particles','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legend(a1,'$\hat{N}_{eff}$ ','FontSize',legend_sz,'interpreter','latex','
    location','southeast')

end

function plotPart3_EstimationError(exstore,sxstore,numPts,xaxis_sz,
    yaxis_sz,legend_sz)
measx = 1:numPts;
std_plot = 3; txt = [num2str(std_plot) '$\sigma$'];

err_line_opts = {'-','LineWidth',2};
std_line_opts = {'-','LineWidth',2,'Color','k'};
```

```
measx1 = [0 measx];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{State Estimation Error vs. Measurement Number using
   Particle Filter with Basic Resampling}','Fontsize',25,'interpreter','
   latex')
a1 = plot(measx1,exstore(1,:),err_line_opts{:});
a2 = plot(measx1,std_plot*sxstore(1,:),std_line_opts{:});
plot(measx1,-std_plot*sxstore(1,:),std_line_opts{:})
ylabel('State','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legendtxt = {'Error',txt};
legend([a1 a2],legendtxt,'FontSize',legend_sz,'interpreter','latex','
   location','southeast')

end

function plotPart3_Neff(Neffstore,numPts,xaxis_sz,yaxis_sz,legend_sz)
measx = 1:numPts;

err_line_opts = {'-','LineWidth',2};

measx1 = [0 measx];

figure; grid on; set(gcf, 'WindowState', 'maximized'); hold on;
title('\textbf{$\hat{N}_{eff}$ vs. Measurement Number using Particle
   Filter with Basic Resampling}','Fontsize',25,'interpreter','latex')
a1 = plot(measx1,Neffstore(1,:),err_line_opts{:});
ylabel('Particles','Fontsize',yaxis_sz,'interpreter','latex')
xlabel('Measurement number','Fontsize',xaxis_sz,'interpreter','latex')
legend(a1,'$\hat{N}_{eff}$ ','FontSize',legend_sz,'interpreter','latex','
   location','southeast')

end
```