**Final Project**

**Classification, Parameter Estimation and Filtering (MAE277)**

Navid Rezazadeh

76848276



Figure 1-Real trajectory of robot VS trajectory without any filtering Vs trajectory filtered using EKF and PF. As it can be seen the result after filtering is alsmot the same as the real trajectory.

****

Figure 2- Log of mean square error of the trajectory without filtering Vs filtering using EKF and PF with respect to real trajectory of the robot.

The code is sent by email. To see the result run the file named as “RUN\_THIS.m” which runs EKF and PF codes and plots the final results.

**EKF:**

clear

clc

load('data\_MAE277\_project.mat');

X\_main=X;

dT = 0.01; % s % Sampling (100 Hz)

time = 0:dT:20; % Time Vector

N = length(time); % Number of time points

% create variables for storing actual states and measurements

nx = 3; % state dimension

nm = 6; % measurement dimension

nw = 2; % process noise dimension

nv = 6; % measurement noise dimension

Z = zeros(nm,N);

sa = 0.5; % m/s^2 % standard deviation for the process noise (acceleration)

sv = 0.01; % rad^2/s^2 % standard deviation for the measurement Noise

Cw = eye(nw)\*sa^2; % process noise covariance (mean is zero)

Cv = eye(nv)\*sv^2; % measurement noise covariance (mean is zero)

% Initialization of pendulum dynamics

X(:,1) = X\_main(:,1); % store actual state evolution in X

X(3,1) = mod(abs(X(3,1)), pi)\*sign(X(3,1)); % adjust X(1,1) in [-pi, pi]

% initial condition; normally distributed

x0 = X(:,1); % with mean x0=[0;0] and

Cx= eye(nx)\*1e-3; % covariance Cx=[sx1^2 0;0 sx2^2]

Hv=eye(6);

% Initialization of Kalman Filter dynamics

% xe denotes the current extended KF state estimate (from update equations)

% and Cex its covariance; xep denotes the current extended state estimate

% from the prediction step and Cxep its covariance;

% xu denotes the current unscented KF state estimate (from update equations)

% and Cux its covariance; xup denotes the current unscented state estimate

% from the prediction step and Cxup its covariance;

Xehat = NaN(nx,N); % store extended KF estimated states in Xehat

msee = NaN(1,N); % store extended KF mse in msee

% intialize filters for first update step

xep=x0; Cxep=Cx;

LM\_position=[5;-2;12;0;20;1];

% Simulation of Pendulum dynamics and KF state estimators

% Filters initialize with UPDATE step!

for n = 1:N

%calculate anticipated measurement

X\_LM=[xep;LM\_position];

Z\_i\_1=[];

for i = 1:3

Z\_i\_1=[Z\_i\_1;LMmeas(X\_LM, i)];

end

%calculating measurment, where measurement is NaN we set to be equal to

%anticipated measurement

z=Zmeas(:,n);

for i = 1:length(z)

if isnan(z(i,1))

z(i,1)=Z\_i\_1(i,1);

end

end

% EKF---------------------------------------------------------------

%calculating jacobian of measurment wrt to states

Hi = [];

for i = 1:3

Hi=[Hi;LMJac(X\_LM, i)];

end

Czep=Hi\*Cxep\*Hi'+Hv\*Cv\*Hv';

Cxzep=Cxep\*Hi';

K = Cxzep\*Czep^-1; % Kalman gain matrix

xe = xep + K\*(z - Z\_i\_1);

xe(3,1) = mod(abs(xe(3,1)), pi)\*sign(xe(3,1)); %check to be in [-pi pi] range

Cxe = Cxep - K\*Czep\*K';

% Store current state estimate and mse

Xehat(:,n) = xe;

msee(n) = trace(Cxe);

% Prediction Step in Extended KF

xep = RobotDyn(xe,dT,Vhat(n),ahat(n)); % nonlinear robot dynamics

% Jacobian of State dynamics wrt w and x

[Fx, Fv, Fa] = RobotJac(xe, dT, Vhat(n), ahat(n));

G=[Fv Fa];

Cxep = Fx\*Cxe\*Fx' + G\*Cw\*G';

end

%

% simulation\_noisy(X,Vhat,ahat)

%

% plot(Xehat(1,:),Xehat(2,:))

**PF:**

clearvars -except Xehat

clc

load('data\_MAE277\_project.mat');

X\_main=X;

dT = 0.01; % s % Sampling (100 Hz)

time = 0:dT:20; % Time Vector

N = length(time); % Number of time points

% create variables for storing actual states and measurements

nx = 3; % state dimension

nm = 6; % measurement dimension

nw = 2; % process noise dimension

nv = 6; % measurement noise dimension

Z = zeros(nm,N);

sa = 0.5; % m/s^2 % standard deviation for the process noise (acceleration)

sv = 0.01; % rad^2/s^2 % standard deviation for the measurement Noise

Cw = eye(nw)\*sa^2; % process noise covariance (mean is zero)

Cv = eye(nv)\*sv^2; % measurement noise covariance (mean is zero)

% Initialization of pendulum dynamics

X(:,1) = X\_main(:,1); % store actual state evolution in X

X(3,1) = mod(abs(X(3,1)), pi)\*sign(X(3,1)); % adjust X(1,1) in [-pi, pi]

% initial condition; normally distributed

x0 = X(:,1); % with mean x0=[0;0] and

Cx= eye(nx)\*1e-3; % covariance Cx=[sx1^2 0;0 sx2^2]

Hv=eye(6);

% Initialization of Kalman Filter dynamics

% xe denotes the current extended KF state estimate (from update equations)

% and Cex its covariance; xep denotes the current extended state estimate

% from the prediction step and Cxep its covariance;

% xu denotes the current unscented KF state estimate (from update equations)

% and Cux its covariance; xup denotes the current unscented state estimate

% from the prediction step and Cxup its covariance;

Xphat = NaN(nx,N); % store extended KF estimated states in Xehat

msee = NaN(1,N); % store extended KF mse in msee

% intialize filters for first update step

xep=x0; Cxep=Cx;

LM\_position=[5;-2;12;0;20;1];

% Simulation of Pendulum dynamics and KF state estimators

% Filters initialize with UPDATE step!

M = 500; % number of particles

Mlow = 400; % threshold of effective particles to perform resampling

% set Mlow = 0 to do resampling at each step

% Initialize PF particles

Xpt = mvnrnd(x0',Cx,M)'; % particles are stored in columns of Xpt

Wpt = ones(1,M)\*(1/M); % initialize importance weights

Xtemp = zeros(nx,M); % temp storage for resamped particles

Mtemp = zeros(1,M); % temp storage for resampling indices

for n = 1:N

z\_temp=Zmeas(:,n);

z\_availabe=[];

Cv\_temp=[];

z=[];

for i = 1:length(z\_temp)/2

if ~isnan(z\_temp(2\*i,1))

z\_availabe=[z\_availabe;i];

z=[z;z\_temp(2\*i-1:2\*i,1)];

end

end

Cv\_temp=Cv(1:2\*length(z\_availabe),1:2\*length(z\_availabe));

if ~isempty(z\_availabe)

% PF---------------------------------------------------------------

for k=1:M

%calculate anticipated measurement

X\_LM=[Xpt(:,k);LM\_position];

Z\_i\_1=[];

for i = 1:length(z\_availabe)

Z\_i\_1=[Z\_i\_1;LMmeas(X\_LM, z\_availabe(i))];

end

%calculating measurment, where measurement is NaN we set to be equal to

%anticipated measurement

Wpt(1,k) = Wpt(1,k)\* mvnpdf(z,Z\_i\_1,Cv\_temp);

end

Wpt = Wpt/sum(Wpt);

% Resample if necessary

Meff = 1/norm(Wpt); % effective number of particles

if Meff < Mlow

Wtemp = cumsum(Wpt); % cumulative sum of weights

for m=1:M

m1 = find(rand() < Wtemp,1); % select sample

if m1>M, keyboard,end

Mtemp(m) = m1; % store index of selected particle

Xtemp(:,m) = Xpt(:,m1);

end

% Muq = length(unique(Mtemp)); % # unique particles after resampling

Xpt = Xtemp; % resampled particles

% reset weights

Wpt = ones(1,M)\*(1/M);

end

end

Xphat(:,n) = Xpt\*Wpt'; % average particles after update step and

% resampling to create PF point estimate

% Prediction step: pass particles through

w1=mvnrnd(0,sa^2,M);

w2=mvnrnd(0,sa^2,M);

for j=1:M

Xpt(:,j) = RobotDyn(Xpt(:,j),dT,Vhat(n)+w1(j),ahat(n)+w2(j)); % particles processed through dynamics

end

end

X\_noisy=simulation\_noisy(X,Vhat,ahat);

%

% plot(Xphat(1,:),Xphat(2,:))