Estimation, Information Fusion and Machine Learning - Assignment 5 - Non-Linear State Estimation Exercises

Kalman Prediction Function

```
function [priorMean,priorCovariance] =
 kalmanPrediction(posteriorMean,posteriorCovar,F,Q)
    % Task 3 - Complete this function

    %priorMean = [];
    %priorCovariance = [];
    priorMean = F * posteriorMean;
    priorCovariance = F*posteriorCovar*transpose(F) + Q;
end
```

Measurement Matrix Function

```
function [measMatrix]= MeasurementMatrix(state,radarState)

% Task 4 - Complete this function


xr = radarState(1);
yr = radarState(2);
x1 = state(1);
y1 = state(2);

jacob_matrix = [(x1 - xr)/((x1 - xr)^2 + (y1 - yr)^2)^(1/2) , (y1 - yr)/((x1
 - xr)^2 + (y1 - yr)^2)^(1/2), 0, 0 ; (-(y1-yr)/((x1-xr)^2+(y1-yr)^2)), ((x1-
xr)/((x1-xr)^2+(y1-yr)^2)), 0, 0];


measMatrix = jacob_matrix;
```

Extended Kalman Filter Update Function

```
function [xPosterior,PPosterior]= EkfUpdate(xPrior,PPrior,z,R,radarState)
% Task 5 - Complete this function

H = MeasurementMatrix(xPrior,radarState);

xr = radarState(1);
yr = radarState(2);
```

```
x1 = xPrior(1);
y1 = xPrior(2);
h = [((x1-xr)^2 + (y1-yr)^2)^0.5 ; atan2((y1-yr),(x1-xr))];

S = H * PPrior * transpose(H) + R ;
W = PPrior * transpose(H) / S;

xPosterior = xPrior + W*(z-h);
PPosterior = PPrior - W*S*transpose(W);
```

Unscented Kalman Filter Update Function

```
function [xPosterior,PPosterior]= UkfUpdate(xPrior,PPrior,z,R,radarState)
% Task 6 - Complete this function

na = length(xPrior);
sample_size = 2*length(xPrior) + 1;
K = 0;
xr = radarState(1);
yr = radarState(2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a_ = xPrior;

samples = [];
sample_weight = [];

mat_sqrt = real(sqrtm((na+K)*PPrior))';
hk = zeros(2,sample_size);

%for i = 0
samples = [samples, a_];
sample_weight = [sample_weight, K/(na+K)];
hk(:,1) = [((samples(1,1)-radarState(1))^2 + (samples(2,1)-
radarState(2))^2)^0.5 ; atan2( samples(2,1)-radarState(2) , samples(1,1)-
radarState(1) ) ];

for i = 1:na
    samples = [samples, a_ + mat_sqrt(:,i)];
    sample_weight = [sample_weight, 1/(2*(na+K))];
    hk(:,i+1) = [((samples(1,i+1)-radarState(1))^2 + (samples(2,i+1)-
radarState(2))^2)^0.5 ; atan2( samples(2,i+1)-radarState(2) , samples(1,i+1)-
radarState(1) ) ];
end
for i = (na+1):(2*na)
   samples = [samples, a_ - mat_sqrt(:,i-na)];
   sample_weight = [sample_weight, 1/(2*(na+K))];
   hk(:,i+1) = [((samples(1,i+1)-radarState(1))^2 + (samples(2,i+1)-
radarState(2))^2)^0.5 ; atan2( samples(2,i+1)-radarState(2) , samples(1,i+1)-
radarState(1) ) ];
end
```

```
sample_weight = sample_weight/sum(sample_weight);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
z_k = sum(sample_weight .* hk ,2);
Pxz = (sample_weight.*(samples - xPrior))*(hk-z_k)';
Pzz = (sample_weight.*(hk - z_k))*(hk - z_k)';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S = R + Pzz;
K = Pxz / S;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xPosterior = xPrior + K * (z-z_k);
PPosterior = PPrior - K * S * transpose(K);
```

Main Function

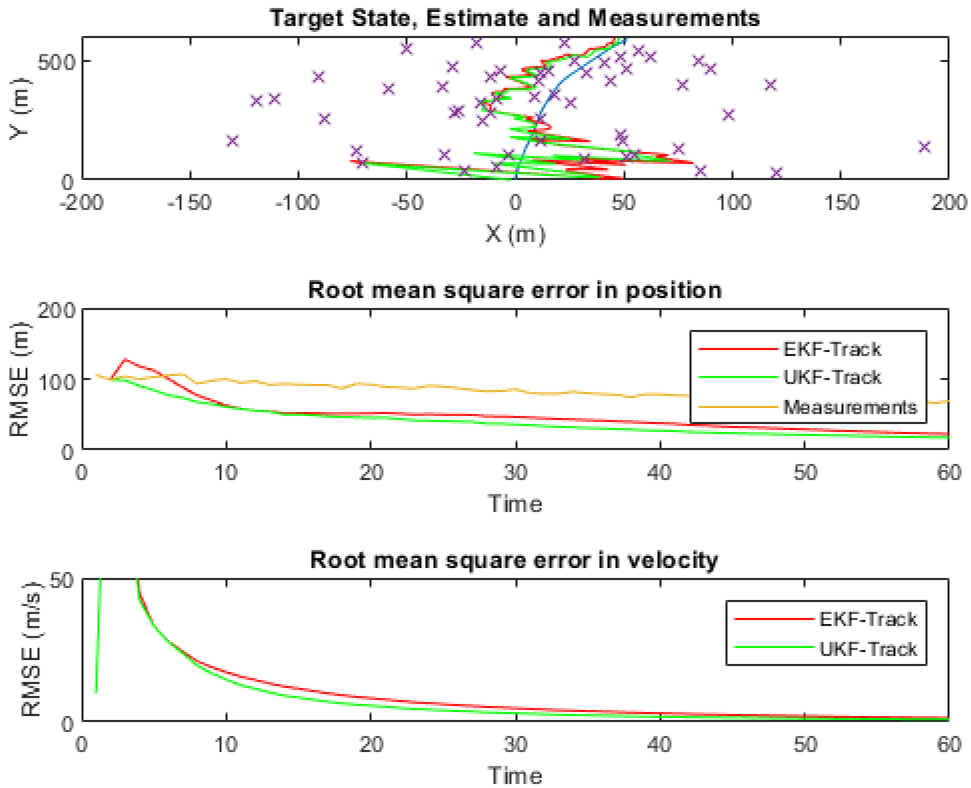
```
close all
clearvars

% Set the number of Monte Carlo runs
numMonteCarlo = 1000;

delT = 1; % Time step
% Task 1 - Complete the state transition matrix
F = [1 0 delT 0 ; 0 1 0 delT ; 0 0 1 0 ; 0 0 0 1]; % State transition matrix
proNoise = 0.01; % Process noise
% Task 1 - Complete the state transition noise matrix
Q = proNoise*[ ...
    (delT^3)/3 0 (delT^2)/2 0; ...
    0 (delT^3)/3 0 (delT^2)/2; ...
    (delT^2)/2 0 delT 0; ...
    0 (delT^2)/2 0 delT];  % State transition noise matrix

sigmaRange = 5; % Measurement error standard deviation in range
Theta = 6
sigmaTheta = Theta*pi/180; % Measurement error standard deviation in angle
% Task 2 - Complete the measurement error covariance
R = [sigmaRange^2 0 ; 0 sigmaTheta^2]; % Measurement error covariance

radarState = [600 800];
radarState2 = [600 800];

% Preallocate arrays for holding RMSE data
ekfRmseTrackPos = zeros(1,60);
ekfRmseTrackVel = zeros(1,60);
ukfRmseTrackPos = zeros(1,60);
ukfRmseTrackVel = zeros(1,60);
rmseMeasPos = zeros(1,60);

% Loop through the Monte Carlo runs
for j = 1:numMonteCarlo
% Generate a random target trajectory according to the model
targetState = zeros(4,60);
```

```
targetState(:,1) = [0 0 0 10]; % Starting state

targetState2 = zeros(4,60);
targetState2(:,1) = [0 0 0 10];



for i = 2:60
% Perform a random walk according to motion model
targetState(:,i) = F*targetState(:,i-1)+mvnrnd([0 0 0 0]',Q)';
targetState2(:,i) = F*targetState2(:,i-1)+mvnrnd([0 0 0 0]',Q)';

end

% Arrays for logging data from a single run
measurements = zeros(2,60); % Log of measurements in polar space
measurementsCart = zeros(2,60); % Log of the measurements in Cartesian space

measurements2 = zeros(2,60); % Log of measurements in polar space
measurementsCart2 = zeros(2,60); % Log of the measurements in Cartesian space

ekfEstimate = zeros(4,60); % Log of EKF estimate
ukfEstimate = zeros(4,60); % Log of UKF estimate

% Loop for each time step
for i = 1:60
% Generate a random measurement
z = GenerateMeasurement(targetState(:,i),R,radarState);
z2 = GenerateMeasurement(targetState(:,i),R,radarState2);

% Store the measurement
measurements(:,i) = z;
[measX,measY] = pol2cart(z(2),z(1)); % Convert measurement into Cartesian
 space
measurementsCart(:,i) = [radarState(1)+measX;radarState(2)+measY]; % Store
 measurement

measurements2(:,i) = z2;
[measX2,measY2] = pol2cart(z2(2),z2(1)); % Convert measurement into Cartesian
 space
measurementsCart2(:,i) = [radarState2(1)+measX2;radarState2(2)+measY2]; %
 Store measurement

if i == 1
    % Store the first time step to be used later in two point
    % initialisation
    ekfEstimate(:,i) = [radarState(1)+measX; radarState(2)+measY;0;0];
    ukfEstimate(:,i) = [radarState(1)+measX; radarState(2)+measY;0;0];
elseif i == 2
    % Perform two point initialisation
    ekfMean = [radarState(1)+measX;radarState(2)+measY;radarState(1)+measX-
ekfEstimate(1,i-1);radarState(2)+measY-ekfEstimate(2,i-1)];
    ukfMean = [radarState(1)+measX;radarState(2)+measY;radarState(1)+measX-
ukfEstimate(1,i-1);radarState(2)+measY-ukfEstimate(2,i-1)];
```

```
    ekfEstimate(:,i) = ekfMean;
    ukfEstimate(:,i) = ukfMean;
    % Covariance initialisation
    T = [
        cos(measurements(2,i)) -measurements(1,i)*sin(measurements(2,i));
        sin(measurements(2,i)) measurements(1,i)*cos(measurements(2,i));
        ];
    Rcart = T*R*T'; % Measurement error in Cartesian frame
    ekfCovar = [Rcart(1,1) Rcart(1,2) 0 0; Rcart(2,1) Rcart(2,2) 0 0; 0 0
 2*Rcart(1,1) 0; 0 0 0 2*Rcart(2,2)];
    ukfCovar = [Rcart(1,1) Rcart(1,2) 0 0; Rcart(2,1) Rcart(2,2) 0 0; 0 0
 2*Rcart(1,1) 0; 0 0 0 2*Rcart(2,2)];
elseif i > 2
    % EKF Prediction
    [ekfPriorMean,ekfPriorCovar] = kalmanPrediction(ekfMean,ekfCovar,F,Q);
    % UKF Prediction
    [ukfPriorMean,ukfPriorCovar] = kalmanPrediction(ukfMean,ukfCovar,F,Q);
    % Extended Kalman update step
    [ekfMean, ekfCovar] =
 EkfUpdate(ekfPriorMean,ekfPriorCovar,z,R,radarState);
    [ekfMean2, ekfCovar2] = EkfUpdate(ekfMean,ekfCovar,z2,R,radarState2);
    ekfEstimate(:,i) = ekfMean2;
    % Extended Kalman update step
    [ukfMean, ukfCovar] =
 UkfUpdate(ukfPriorMean,ukfPriorCovar,z,R,radarState);
    [ukfMean2, ukfCovar2] = UkfUpdate(ukfMean,ukfCovar,z2,R,radarState2);
    ukfEstimate(:,i) = ukfMean2;
end
end
% Log EKF RMSE
ekfRmseTrackPos = ekfRmseTrackPos + (ekfEstimate(1,:)-targetState(1,:)).^2 +
 (ekfEstimate(2,:)-targetState(2,:)).^2;
ekfRmseTrackVel = ekfRmseTrackVel + (ekfEstimate(3,:)-targetState(3,:)).^2 +
 (ekfEstimate(4,:)-targetState(4,:)).^2;
% Log UKF RMSE
ukfRmseTrackPos = ukfRmseTrackPos + (ukfEstimate(1,:)-targetState(1,:)).^2 +
 (ukfEstimate(2,:)-targetState(2,:)).^2;
ukfRmseTrackVel = ukfRmseTrackVel + (ukfEstimate(3,:)-targetState(3,:)).^2 +
 (ukfEstimate(4,:)-targetState(4,:)).^2;
% Log measurement RMSE
rmseMeasPos = rmseMeasPos + (measurementsCart(1,:)-targetState(1,:)).^2 +
 (measurementsCart(2,:)-targetState(2,:)).^2;
% Make a plot of a single run for only the single run
if j==1
%figure
t = tiledlayout(3,1);
nexttile
plot(targetState(1,:),targetState(2,:))
hold on
plot(ekfEstimate(1,:),ekfEstimate(2,:),'r')
plot(ukfEstimate(1,:),ukfEstimate(2,:),'g')
plot(measurementsCart(1,:),measurementsCart(2,:),'x')
ylim([0 600])
xlim([-200 200])
```

```
xlabel('X (m)')
ylabel('Y (m)')
title('Target State, Estimate and Measurements')
end
end

% Calculate the RMSE
ekfRmseTrackPos = sqrt(ekfRmseTrackPos/numMonteCarlo);
ukfRmseTrackPos = sqrt(ukfRmseTrackPos/numMonteCarlo);
rmseMeasPos = sqrt(rmseMeasPos/numMonteCarlo);
ekfRmseTrackVel = sqrt(ekfRmseTrackVel/numMonteCarlo);
ukfRmseTrackVel = sqrt(ukfRmseTrackVel/numMonteCarlo);

% Plot the RMSE in position
%figure
nexttile
plot(1:60,ekfRmseTrackPos(1:60),'r')
hold on
plot(1:60,ukfRmseTrackPos(1:60),'g')
plot(1:60,rmseMeasPos(1:60))
ylim([0 200])
xlabel('Time')
ylabel('RMSE (m)')
title('Root mean square error in position')
legend('EKF-Track','UKF-Track','Measurements')

% Plot the RMSE in velocity
%figure
nexttile
plot(1:60,ekfRmseTrackVel(1:60),'r')
hold on
plot(1:60,ukfRmseTrackVel(1:60),'g')
ylim([0 50])
xlabel('Time')
ylabel('RMSE (m/s)')
title('Root mean square error in velocity')
legend('EKF-Track','UKF-Track')
```

```
Theta =

     6
```



Published with MATLAB® R2021b