**HA3 ANALYSIS**

```
close all;
clear all;
clc;
%% Problem 1: Approximations of mean and covariance
% number of samples
N=10000;
%update type
type = 'CKF';
% two prior state_densitiess 1 or 2
state_densities = '1';
switch state_densities
case '1'
X_0_mean = [125 125]';
P_0 = [100 0;0 25];
case '2'
X_0_mean = [-25 125]';
P_0 = [100 0;0 25];
case '3'
X_0_mean = [60 60]';
P_0 = [100 0;0 25];
end
%sensor position
S1 = [0;100]';
S2 = [100;0]';
% Noise covariance
R = diag([0.1*pi/180 0.1*pi/180].^2);
% Measurement model from functions
h = @(x) dualBearingMeasurement(x,S1,S2);
MeasurementSequence =
@(x)genNonLinearMeasurementSequence(x, h, R);
%1a
[y_mu, y_sigma, y_s] = approxGaussianTransform(X_0_mean,
P_0, @(x)genNonLinearMeasurementSequence(x,h,R) , N);
%1c
switch type
case 'UKF'
[SP,W] = sigmaPoints(X_0_mean,P_0,type);
hSP1 = h(SP);
[ye_mu, ye_sigma] = nonLinKFprediction(X_0_mean, P_0, h, R,
type);
case 'CKF'
[SP,W] = sigmaPoints(X_0_mean,P_0,type);
hSP1 = h(SP);
```

```
[ye_mu, ye_sigma] = nonLinKFprediction(X_0_mean, P_0, h, R,
type);
case 'EKF'
[hx, dhx] = h(X_0_mean);
[ye_mu, ye_sigma] = nonLinKFprediction(X_0_mean, P_0, h, R,
type);
end

level=3;
[xy1] = sigmaEllipse2D(y_mu, y_sigma, level,200);
[xy2] = sigmaEllipse2D(ye_mu, ye_sigma, level,200);
figure(1);
scatter(y_s(1,:), y_s(2,:),5,'*r')
hold on
plot(xy1(1,:), xy1(2,:),'Linewidth',2);
scatter(y_mu(1), y_mu(2),25, 'og');
switch type
case 'UKF'
scatter(hSP1(1,:), hSP1(2,:),'om');
case 'CKF'
scatter(hSP1(1,:), hSP1(2,:),'ok');
end
hold on
scatter(ye_mu(1,:), ye_mu(2,:),'oc')
plot(xy2(1,:),xy2(2,:))
xlabel('phi_1');
ylabel('phi_2');
switch type
case 'UKF'
legend('Measurement state densities','Untransformed
ellipse','untransformed mean','sigma points','transformed
mean','transformed ellipse','Location','best')
case 'CKF'
legend('Measurement state densities','Untransformed
ellipse','untransformed mean','sigma points','transformed
mean','transformed ellipse','Location','best')
case 'EKF'
legend('Measurement state densities','Untransformed
ellipse','untransformed mean','transformed
mean','transformed ellipse','Location','best')
end

%% problem 2: Non-linear Kalman
x_0=[0 0 20 0 (5*pi)/180]';
```

```
P_0= diag([100 100 4 (pi/180)^2 (pi/180)^2]);
s1=[-200,100]';
s2=[-200,-100]';
T=1;
N=100;
sigma_v = 1;
sigma_w = (pi/180);
if state_densities == 1
    sigma_phi_1 = (10*pi/180);
else
    sigma_phi_1 = (0.5*pi/180);
end
sigma_phi_2 = (0.5*pi/180);
Q = diag([0, 0, T*sigma_v, 0, T*sigma_w].^2);
R = diag([sigma_phi_1, sigma_phi_2].^2);
% Process and Measurement Model and Transformed Sequence
f=@(x)coordinatedTurnMotion(x,T);
h=@(x)dualBearingMeasurement(x,s1,s2);
X=genNonLinearStateSequence(x_0, P_0, f, Q, N);
Y=genNonLinearMeasurementSequence(X, h, R);
Xm(1,:) = (s2(2)-s1(2)+tan(Y(1,:))*s1(1)-
tan(Y(2,:))*s2(1))./(tan(Y(1,:))- tan(Y(2,:)));
Xm(2,:) = s1(2)+tan(Y(1,:)).*(Xm(1,:)- s1(1));

%2a and 2b
level=3;
for type = {'EKF','UKF','CKF'}
    [xf, Pf, xp, Pp] = nonLinearKalmanFilter(Y, x_0, P_0,
f, Q, h, R, type{1});
    figure();
    clf;
    hold on
    plot(s1(1),s1(2),'*r','Linewidth',2);
    plot(s2(1),s2(2),'or','Linewidth',2);
    plot(X(1,:),X(2,:),'-m');
    plot(xf(1,:),xf(2,:),'-.c');
    plot(Xm(1,:),Xm(2,:),'+g');
    for i= 1:5:length(xf)
        
[xy3]=sigmaEllipse2D(xf(1:2,i),Pf(1:2,1:2,i),level,50);
        plot(xy3(1,:),xy3(2,:),'--b');
    end
    xlabel('x');
    ylabel('y');
```

```
    legend('Sensor 1 Pos','Sensor 2 Pos','True
State','Filtered State','Measured State','3sigma
region','Interpreter','Latex','Location','best');
    hold off
end

% Problem 2.c: Plot Histograms of Estimation Error
MC = 100;
est_err = cell(1,3);
type = {'EKF','UKF','CKF'};
for imc = 1:MC
    X = genNonLinearStateSequence(x_0, P_0, f, Q, N);
    Y = genNonLinearMeasurementSequence(X, h, R);
     for itype = 1:numel(type)

        %  Kalman filter
        [xfM,PfM,xpM,PpM] =
nonLinearKalmanFilter(Y,x_0,P_0,f,Q,h,R,type{itype});
        est_err{1,itype}(1:2,end+1:end+length(xf)) =
X(1:2,2:end) - xfM(1:2,:);

    end
end

 MCcount = 100;
 close all;
 loc = {'x','y'};
     figure(2);
     for itype = 1:numel(type)
        for iloc = 1:numel(loc)
            subplot(2,3, itype + (iloc-1)*numel(type) );
            hold on;
%
             histo = est_err{1,itype}(iloc,:);
             errmu  = mean(histo);
             errsig = std(histo);

            histogram( histo, MCcount
,'Normalization','pdf');
            level=3;
            N2=100;
            x = linspace(errmu-level*sqrt(errsig^2),
errmu+level*sqrt(errsig^2), N2);
            y = normpdf(x, errmu, sqrt(errsig^2));
```

```
            plot(x,y, 'LineWidth',2 );

        end
    end

%% Problem 3: Tuning non-linear filters
% Sampling period
T = 0.1;
% Length of time sequence
K = 600;
% Allocate memory
omega = zeros(1,K+1);
% Turn rate
omega(150:450) = -pi/301/T;
% Initial state
x0 = [0 0 20 0 omega(1)]';
% Allocate memory
X = zeros(length(x0),K+1);
X(:,1) = x0;
% Create true track
for i=2:K+1
% Simulate
X(:,i) = coordinatedTurnMotion(X(:,i-1),T);
% Set turn-rate
X(5,i) = omega(i);
end
% Prior
x_0 = [0 0 0 0 0]';
P_0 = diag([10 10 10 5*pi/180 pi/180].^2);
% Sensor positions
s_1 =[300 -100]';
s_2 =[300 -300]';
gamma=[0 0;0 0;1 0;0 0;0 1];
Q=gamma*diag([200*1 200*pi/180].^2)*gamma';
% measurement variance
R = 0.05*diag([pi/180 pi/180].^2);
% generate measurement sequence
h = @(x) dualBearingMeasurement(x,s_1,s_2);
Y = genNonLinearMeasurementSequence(X,h,R);
% Motion model
motionModel=@(x)coordinatedTurnMotion(x,T);
[xf,Pf,xp,Pp]=nonLinearKalmanFilter(Y,x_0,P_0,motionModel,Q
,h,R,'CKF');
%unfiltered position
```

```
xmeas =(s_2(2)-s_1(2)+tan(Y(1,:))*s_1(1)-
tan(Y(2,:))*s_2(1))./(tan(Y(1,:))-tan(Y(2,:)));
ymeas =s_1(2)+tan(Y(1,:)).*(xmeas(1,:) - s_1(1));
figure(1);
grid on;
hold on
axis equal;
plot(X(1,:),X(2,:),'m');
plot(xf(1,:),xf(2,:),'r');
scatter(s_1(1),s_1(2),100,'o');
scatter(s_2(1),s_2(2),200,'o');
axis manual
plot(xmeas,ymeas,'*');
for i=1:15:length(xf)
variance_xy = sigmaEllipse2D(xf(1:2,i),Pf(1:2,1:2,i),3,50);
plot(variance_xy(1,:),variance_xy(2,:))
end
xlabel('pos x');
ylabel('pos y');
legend('true state','filtered position','sensor1
potion','sensor2 potion','Measurements')
% plot position error
err=X(1:2,2:end) - xf(1:2,:);
figure(2);
grid on;
hold on;
plot((1:K)*T,err(1,:),(1:K)*T,err(2,:))
```

**coordinatedTurnMotion:**

```
function [fx, Fx] = coordinatedTurnMotion(x, T)
%COORDINATEDTURNMOTION calculates the predicted state using
a coordinated
%turn motion model, and also calculated the motion model
Jacobian
%
%Input:
%   x           [5 x 1] state vector
%   T           [1 x 1] Sampling time
%
%Output:
%   fx          [5 x 1] motion model evaluated at state x
%   Fx          [5 x 5] motion model Jacobian evaluated at
state x
```

```
%
% NOTE: the motion model assumes that the state vector x
consist of the
% following states:
%   px          X-position
%   py          Y-position
%   v           velocity
%   phi         heading
%   omega       turn-rate

px = x(1);
py = x(2);
v = x(3);
phi = x(4);
omega = x(5);

% Your code for the motion model here
fx = [                      % Motion Model
      px + T*v*cos(phi)
      py + T*v*sin(phi)
      v
      phi + T*omega
      omega
     ];

%Check if the Jacobian is requested by the calling function
if nargout > 1
    % Your code for the motion model Jacobian here
    Fx = [
          1,0,T*cos(phi),-T*v*sin(phi),0;
          0,1,T*sin(phi),T*v*cos(phi),0;
          0,0,1,0,0;
          0,0,0,1,T;
          0,0,0,0,1;
         ];
end
end
```

**dualBearingMeasurement:**

```
function [hx, Hx] = dualBearingMeasurement(x, s1, s2)
%DUOBEARINGMEASUREMENT calculates the bearings from two
sensors, located in
%s1 and s2, to the position given by the state vector x.
Also returns the
```

```
%Jacobian of the model at x.
%
%Input:
%   x            [n x 1] State vector, the two first element
are 2D position
%   s1           [2 x 1] Sensor position (2D) for sensor 1
%   s2           [2 x 1] Sensor position (2D) for sensor 2
%
%Output:
%   hx           [2 x 1] measurement vector
%   Hx           [2 x n] measurement model Jacobian
%
% NOTE: the measurement model assumes that in the state
vector x, the first
% two states are X-position and Y-position.

% Your code here
hx = zeros(2,size(x,2));
Hx = zeros(2,size(x,1));
hx(1:2,:)=[
     atan2(x(2,:)-s1(2),x(1,:)-s1(1));
     atan2(x(2,:)-s2(2),x(1,:)-s2(1));
     ];
Hx(1:2,1:2)=[
     -(x(2)-s1(2))/( (x(1)-s1(1))^2 + (x(2)-s1(2))^2
),(x(1)-s1(1))/( (x(1)-s1(1))^2 + (x(2)-s1(2))^2);
     -(x(2)-s2(2))/( (x(1)-s2(1))^2 + (x(2)-s2(2))^2
),(x(1)-s2(1))/( (x(1)-s2(1))^2 + (x(2)-s2(2))^2);
     ];
end
```

**genNonLinearStateSequence**

```
function X = genNonLinearStateSequence(x_0, P_0, f, Q, N)
%GENNONLINEARSTATESEQUENCE generates an N+1-long sequence
of states using a
%    Gaussian prior and a nonlinear Gaussian process model
%
%Input:
%   x_0          [n x 1] Prior mean
%   P_0          [n x n] Prior covariance
%   f            Motion model function handle
%                [fx,Fx]=f(x)
```

```
%                Takes as input x (state),
%                Returns fx and Fx, motion model and
Jacobian evaluated at x
%                All other model parameters, such as sample
time T,
%                must be included in the function
%   Q            [n x n] Process noise covariance
%   N            [1 x 1] Number of states to generate
%
%Output:
%   X            [n x N+1] State vector sequence
%

% Your code here
X = zeros(length(x_0),N+1);
a = (mvnrnd(zeros(size(Q,1),1),Q,N))';
X(:,1) = (mvnrnd(x_0,P_0))';
for i = 2:N+1
    [fx,~]=f(X(:,i-1));
    X(:,i)=fx + a(:,i-1);
end
end
```

**genNonLinearMeasurementSequence:**

```
function Y = genNonLinearMeasurementSequence(X, h, R)
%GENNONLINEARMEASUREMENTSEQUENCE generates ovservations of
the states
% sequence X using a non-linear measurement model.
%
%Input:
%   X            [n x N+1] State vector sequence
%   h            Measurement model function handle
%   h            Measurement model function handle
%                [hx,Hx]=h(x)
%                Takes as input x (state)
%                Returns hx and Hx, measurement model and
Jacobian evaluated at x
%   R            [m x m] Measurement noise covariance
%
%Output:
%   Y            [m x N] Measurement sequence
%

% Your code here
```

```
Y = zeros(length(R),size(X,2)-1);
a = (mvnrnd(zeros(length(R),1),R,size(X,2)-1))';
for i = 1:(size(X,2)-1)
    [hx,~] = h(X(:,i+1));
    Y(:,i) = hx+a(:,i);
end
end
```

**sigmaPoints:**

```
function [SP,W] = sigmaPoints(x, P, type)
% SIGMAPOINTS computes sigma points, either using unscented
transform or
% using cubature.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%
%Output:
%   SP          [n x 2n+1] UKF, [n x 2n] CKF. Matrix with
sigma points
%   W           [1 x 2n+1] UKF, [1 x 2n] UKF. Vector with
sigma point weights
%
n = length(x);
    switch type
        case 'UKF'

            SP = zeros(n,2*n+1);
            SP(:,1) = x;
            Sqrt_P = sqrtm(P);
            W0 = 1-n/3;
            for i = 1:size(P,2)
                SP(:,i+1) = x + sqrt((n)/(1-
W0))*Sqrt_P(:,i);
                SP(:,i+n+1) = x - sqrt((n)/(1-
W0))*Sqrt_P(:,i);
            end
            W = [W0, ((1-W0)/(2*n))*ones(1,2*n)];

        case 'CKF'

            SP = zeros(n,2*n);
```

```
            Sqrt_P = sqrtm(P);
            for i = 1:size(P,2)
                SP(:,i) = x + sqrt(n)*Sqrt_P(:,i);
                SP(:,i+n) = x - sqrt(n)*Sqrt_P(:,i);
            end
            W = ((1)/(2*n))*ones(1,2*n);

        otherwise
            error('Incorrect type of sigma point')
    end

end
```

**nonLinKFprediction:**

```
function [x, P] = nonLinKFprediction(x, P, f, Q, type)
%NONLINKFPREDICTION calculates mean and covariance of
predicted state
%   density using a non-linear Gaussian model.
%
%Input:
%   x               [n x 1] Prior mean
%   P               [n x n] Prior covariance
%   f               Motion model function handle
%                   [fx,Fx]=f(x)
%                   Takes as input x (state),
%                   Returns fx and Fx, motion model and
Jacobian evaluated at x
%                   All other model parameters, such as sample
time T,
%                   must be included in the function
%   Q               [n x n] Process noise covariance
%   type            String that specifies the type of non-
linear filter
%
%Output:
%   x               [n x 1] predicted state mean
%   P               [n x n] predicted state covariance
%
n=length(x)
[fx,Fx]=f(x);
    switch type
        case 'EKF'
```

```
            x=fx;
            P=Fx*P*Fx'+Q;

        case 'UKF'

            [SP,W] = sigmaPoints(x, P, 'UKF');
            for i=0:2*n
            [fx,Fx]=f(SP(:,i+1));
            x(:,i+1)=fx*W(:,i+1);
            end
            x=sum(x,2);
            for i=0:2*n
            [fx,Fx]=f(SP(:,i+1));
            P(:,:,i+1)=(fx-x)*(fx-x)'*W(:,i+1);
            end
            P=sum(P,3)+Q;

            % Make sure the covariance matrix is semi-
definite
            if min(eig(P))<=0
                [v,e] = eig(P, 'vector');
                e(e<0) = 1e-4;
                P = v*diag(e)/v;
            end

        case 'CKF'

            [SP,W] = sigmaPoints(x, P, 'CKF');
             for i=1:2*n
             [fx,Fx]=f(SP(:,i));
             x(:,i)=fx*W(:,i);
             end
             x=sum(x,2);
             for i=1:2*n
             [fx,Fx]=f(SP(:,i));
             P(:,:,i)=(fx-x)*(fx-x)'*W(:,i);
             end
             P=sum(P,3)+Q;

        otherwise
            error('Incorrect type of non-linear Kalman
filter')
    end
```

```
end
```

**nonLinKFupdate:**

```
function [x, P] = nonLinKFupdate(x, P, y, h, R, type)
%NONLINKFUPDATE calculates mean and covariance of predicted
state
%   density using a non-linear Gaussian model.
%
%Input:
%   x               [n x 1] Prior mean
%   P               [n x n] Prior covariance
%   y               [m x 1] measurement vector
%   h               Measurement model function handle
%                   [hx,Hx]=h(x)
%                   Takes as input x (state),
%                   Returns hx and Hx, measurement model and
Jacobian evaluated at x
%                   Function must include all model parameters
for the particular model,
%                   such as sensor position for some models.
%   R               [m x m] Measurement noise covariance
%   type            String that specifies the type of non-
linear filter
%
%Output:
%   x               [n x 1] updated state mean
%   P               [n x n] updated state covariance
%
[hx,Hx]=h(x);
n=length(x);
m=length(y);
    switch type
        case 'EKF'

             S=Hx*P*Hx'+R;
             K=P*Hx'*S^-1;
             x=x+K*(y-hx);
             P=P-K*S*K';

        case 'UKF'

             [SP,W]=sigmaPoints(x,P,'UKF');
             for i=0:2*n
```

```
            ycap(:,i+1)=h(SP(:,i+1))*W(i+1);
            end
            ycap=sum(ycap,2);
            for i=0:2*n
            Pcap(:,:,i+1)=(SP(:,i+1)-x)*(h(SP(:,i+1))-
ycap)'*W(i+1);
            end
            Pcap=sum(Pcap,3);
            for i=0:2*n
            Scap(:,:,i+1)=(h(SP(:,i+1))-
ycap)*(h(SP(:,i+1))-ycap)'*W(i+1);
            end
            Scap=sum(Scap,3)+R;
            x=x+Pcap*inv(Scap)*(y-ycap);
            P=P-Pcap*inv(Scap)*Pcap';

           % Make sure the covariance matrix is semi-
definite
           if min(eig(P))<=0
               [v,e] = eig(P,'vector');
               e(e<0) = 1e-4;
               P = v*diag(e)/v;
           end

        case 'CKF'

            [SP,W]=sigmaPoints(x,P,'CKF');
            for i=1:2*n
            ycap(:,i)=h(SP(:,i))*W(i);
            end
            ycap=sum(ycap,2);
            for i=1:2*n
            Pcap(:,:,i)=(SP(:,i)-x)*(h(SP(:,i))-
ycap)'*W(i);
            end
            Pcap=sum(Pcap,3);
            for i=1:2*n
            Scap(:,:,i)=(h(SP(:,i))-ycap)*(h(SP(:,i))-
ycap)'*W(i);
            end
            Scap=sum(Scap,3)+R;
            x=x+Pcap*Scap^-1*(y-ycap);
            P=P-Pcap*Scap^-1*Pcap';
```

```
        otherwise
            error('Incorrect type of non-linear Kalman
filter')
    end

end
```

**nonLinearKalmanFilter:**

```
function [xf, Pf, xp, Pp] = nonLinearKalmanFilter(Y, x_0,
P_0, f, Q, h, R, type)
%NONLINEARKALMANFILTER Filters measurement sequence Y using
a
% non-linear Kalman filter.
%
%Input:
%   Y           [m x N] Measurement sequence for times
1,...,N
%   x_0         [n x 1] Prior mean for time 0
%   P_0         [n x n] Prior covariance
%   f                   Motion model function handle
%                       [fx,Fx]=f(x)
%                       Takes as input x (state)
%                       Returns fx and Fx, motion model and
Jacobian evaluated at x
%   Q           [n x n] Process noise covariance
%   h                   Measurement model function handle
%                       [hx,Hx]=h(x,T)
%                       Takes as input x (state),
%                       Returns hx and Hx, measurement
model and Jacobian evaluated at x
%   R           [m x m] Measurement noise covariance
%
%Output:
%   xf          [n x N]     Filtered estimates for times
1,...,N
%   Pf          [n x n x N] Filter error convariance
%   xp          [n x N]     Predicted estimates for times
1,...,N
%   Pp          [n x n x N] Filter error convariance
%

% Your code here. If you have good code for the Kalman
filter, you should re-use it here as
% much as possible.
```

```
N = size(Y,2);
%% Data allocation
xp = zeros(length(x_0),N);
Pp = zeros(length(x_0),length(x_0),N);
xf = zeros(length(x_0),N+1);
Pf = zeros(length(x_0),length(x_0),N+1);
%% Filter Implementation
xf(:,1)=x_0;
Pf(:,:,1)=P_0;
for i = 1:N
    [xp(:,i), Pp(:,:,i)] = nonLinKFprediction(xf(:,i),
Pf(:,:,i), f, Q, type);
    [xf(:,i+1), Pf(:,:,i+1)] = nonLinKFupdate(xp(:,i),
Pp(:,:,i), Y(:,i), h, R, type);
end
xf = xf(:,2:end);
Pf = Pf(:,:,2:end);
end
```