ISTANBUL TECHNICAL UNIVERSITY

PROBABILISTIC METHODS IN ROBOTICS

HOMEWORK III

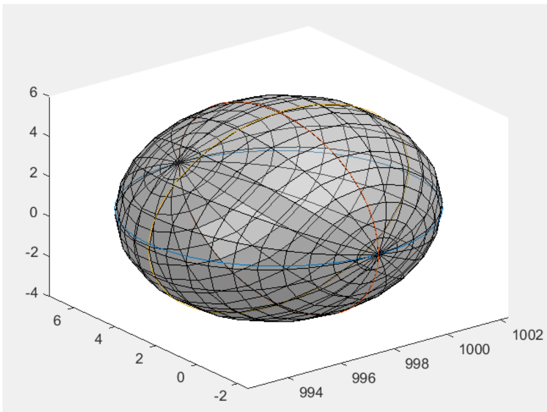
HAKAN AŞIK

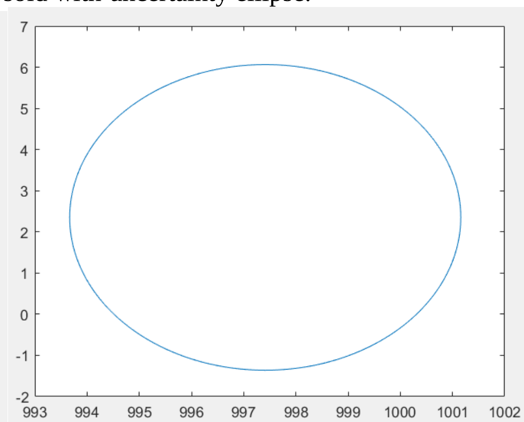
518182008

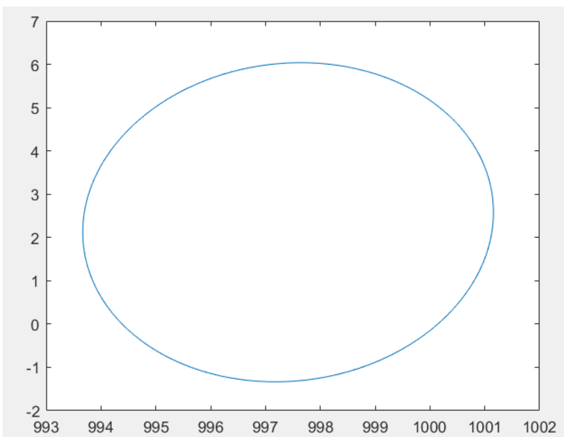
20.10.2019

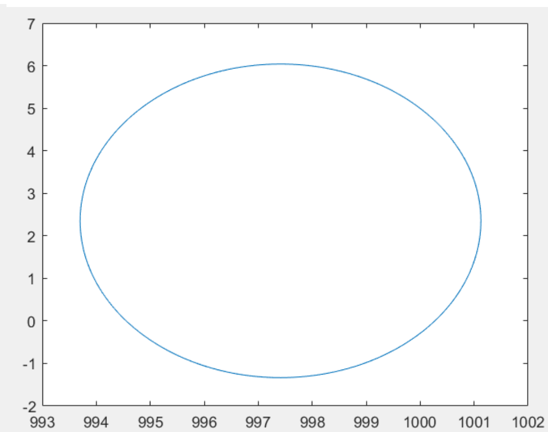
(c) Draw the uncertainty ellipse of the Gaussian and compare it with your intuitive solution.

Since the dimension is 3, resulting figure is an ellipsoid with uncertainty ellipse.

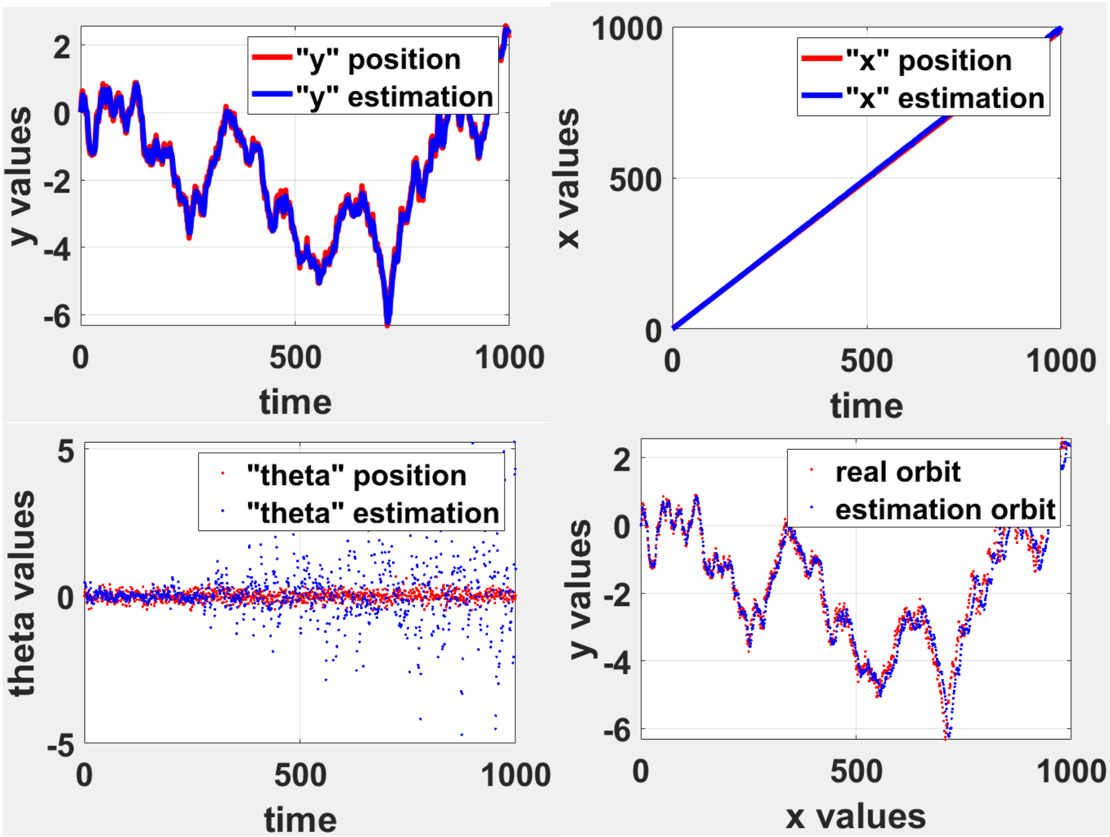








(d) Now incorporate a measurement. Our measurement shall be a noisy projection of the x-coordinate of the robot, with covariance $Q = 0.01$. Specify the measurement model. Now apply the measurement both to your intuitive posterior, and formally to the EKF estimate using the standard EKF machinery. Give the exact result of the EKF, and compare it with the result of your intuitive analysis.



As can be seen from the figures, estimations are well following the intuitive analysis.

UKF_main m file

```
clear all;
clc;

xPresent_post  =[0; 0; 0];
covPresent_post=[0.01   0      0 ;...
                  0    0.01   0; ...
                  0     0     10000];

xbarPast_post = xPresent_post;
covPast_post= covPresent_post;

u_t=[];

time=0:1:1000;
Nsamples=length(time);
Xmsaved=[];
Xhsaved=[];

for t = 1:Nsamples

  [xm, ym, theta] = f_GetPosUK(t);                % real
value
  [xh, yh, thetah,xbarPresent_post,CovPresent_post] =
UnKalFilt_hkn(covPast_post,xbarPast_post, u_t, [xm; ym;
theta]) % kalman result

  covPast_post= CovPresent_post
  xbarPast_post = xbarPresent_post
  Xmsaved(t,:) = [xm, ym, theta];
  Xhsaved(t,:) = [xh, yh, thetah];

end
% Elipsoid
Pxyt=CovPresent_post;
figure (5)
h=f_error_ellipse_drawing(Pxyt,xbarPresent_post)
% x-y
Pxy=[CovPresent_post(1,1) CovPresent_post(1,2);
     CovPresent_post(2,1) CovPresent_post(2,2)];
figure (6)
h=f_error_ellipse_drawing(Pxy,xbarPresent_post)
% x-t
Pxt=[CovPresent_post(1,1) CovPresent_post(1,3);
```

```
     CovPresent_post(3,1) CovPresent_post(3,3)];
figure (7)
h=f_error_ellipse_drawing(Pxt,xbarPresent_post)
% y-t
Pyt=[CovPresent_post(2,2) CovPresent_post(2,3);
     CovPresent_post(3,2) CovPresent_post(3,3)];
figure (8)
h=f_error_ellipse_drawing(Pyt,xbarPresent_post)
%
figure (1)
plot(time,Xmsaved(:,1),'r','linewidth',4)    % x value
(real)
hold on
plot(time,Xhsaved(:,1),'b','linewidth',4)  % x value
(estimation)
xlabel('time', 'FontSize', 24);
ylabel('x values', 'FontSize', 24);
legend('"x" position','"x" estimation')
set(gca,'FontSize',24,'fontWeight','bold')
grid
%
figure (2)
plot(time,Xmsaved(:,2),'r','linewidth',4)    % y value
(real)
hold on
plot(time,Xhsaved(:,2),'b','linewidth',4)  % y value
(estimation)
xlabel('time', 'FontSize', 24);
ylabel('y values', 'FontSize', 24);
legend('"y" position','"y" estimation')
set(gca,'FontSize',24,'fontWeight','bold')
grid
%
figure (3)
plot(time,Xmsaved(:,3),'r.','linewidth',4)   % theta
value (real)
hold on
plot(time,Xhsaved(:,3),'b.','linewidth',4)   % theta
value (estimation)
xlabel('time', 'FontSize', 24);
ylabel('theta values', 'FontSize', 24);
legend('"theta" position','"theta" estimation')
set(gca,'FontSize',24,'fontWeight','bold')
grid
```

```
%
figure (4)
plot(Xmsaved(:,1),Xmsaved(:,2),'r.','linewidth',4)    %
real values in x,y direction
hold on
plot(Xhsaved(:,1),Xhsaved(:,2),'b.','linewidth',4)%
estimation values in x,y direction
xlabel('x values', 'FontSize', 24);
ylabel('y values', 'FontSize', 24);
legend('real orbit','estimation orbit')
set(gca,'FontSize',24,'fontWeight','bold')
grid
```

Robot Pose m file

```
function [xm, ym, theta] =f_GetPosUK(t)
%
persistent Posxm Posym

if isempty(Posxm)
    Posxm=0;
    Posym=0;
end
d=1;
%% Generate values from a normal distribution with
specified mean vector and covariance matrix.
mu = [0 0 0];
sigma = [0.01 0 0; 0 0.01 0; 0 0 10000];
R = chol(sigma);
zp = repmat(mu,t,1) + randn(t,3)*R;
xn=zp(t,1);
yn=zp(t,2);
theta=zp(t,3)*pi/1800

xm=Posxm+d*cos(theta);
ym=Posym+d*sin(theta);

Posxm=xm;         % true position
Posym=ym;         % true position
```

Unscented Kalman Filter function m file

```
function [xh,yh,thetah,xbarPresent_post,
CovPresent_post] =
UnKalFilt_hkn3(covPast_post,xbarPast_post, u_t, z_t)

%% %%PARAMETER DEFINITION
n=3;
m=3;
dt=0.01;
z = [z_t(1); z_t(2); z_t(3)] ;  % 3x1
theta=z_t(3)

alpha=1; % 0<alpha<1
kappa=1; % kappa=>0
lambda= alpha^2*(kappa+n)-n;
beta=2;

H = [1 0 0;
       0 1 0;
        0 0 1];
Q_t = 1.0*eye(3);
R_t = 1.0*eye(3);

%% %%PREDICTION STEP
% 1. Generate Sigma points
%%2. Propagate each sigma-point through prediction
% nx(2*n+1)--> 3x7
[XSigmaPresent_prior]= SigPointGen(n, lambda,
xbarPast_post, covPast_post, theta, dt)

% 3. Compute Mean and Covariance matrix
% Weight for computing the Mean
Wsigma_mean(1) = lambda/(n+lambda)
% Weight for computing the Covariance
Wsigma_cov(1) = ( lambda/(n+lambda) ) +1- alpha^2 +beta

for i=1:(2*n)
Wsigma_mean(i+1) = 1/( 2*(n+lambda) ) % 1<k<2n  In
total- [1x(2*n+1)] : 1x7

Wsigma_cov(i+1) = 1/( 2*(n+lambda) ) % 1<k<2n   In
total- [1x(2*n+1)] : 1x7
end
```

```
[xbarPresent_prior, CovPresent_prior]= UnsTrans(
XSigmaPresent_prior, Wsigma_mean, Wsigma_cov, R_t)
%%%Thus we get 3x1 xbarPresent_prior and 3x3
CovPresent_prior
%% %% CORRECTION STEP
HSigmaPresent_prior=zeros(n,2*n+1)
HSigmaPresent_prior(:,1)= xbarPresent_prior

MSRpresent =chol( (n+lambda)*CovPresent_prior )  %
chol: To Calculate square root of error covariance
% Calculate mean of estimated output
for i=1:n
HSigmaPresent_prior(:,i+1)= xbarPresent_prior + (
MSRpresent(i,:)'  )
HSigmaPresent_prior(:,n+i+1)= xbarPresent_prior - (
MSRpresent(i,:)'  )
end

% 5. Propogate Sigma points to obtain prediction of the
observation

Zet_t_k=zeros(m, 2*n+1);
for k=1:2*n+1
    Zet_t_k(:,k)=hx(HSigmaPresent_prior(:,k),theta) %
Zet_t_k is the transformed sigma points
end

% 6. & 7. Compute estimated observation and covariance
matrix
%
[zbar_t, S_t]= UnsTrans( Zet_t_k, Wsigma_mean,
Wsigma_cov, Q_t)

% 8. Compute Cross covariance matrix
CrossCovar=zeros(n,m);
for i=1:2*n+1
CrossCovar= CrossCovar+ Wsigma_cov(i) * (
XSigmaPresent_prior(:,i) - xbarPresent_prior ) * (
XSigmaPresent_prior(:,i) - zbar_t)'
end


% 9. Compute Kalman gain
K_t= CrossCovar *inv(S_t) % [nxn]:3x3
```

```
% K = PxyCrossCovar/PyyCrossCovar

% 10. Correction of the mean
% xbarPresent_post = xbarPresent_prior + K_t * (z_t -
zbar_t) % [nx1]:3x1
xbarPresent_post = xbarPresent_prior + K_t * (z_t -
xbarPresent_prior)
% xbarPresent_post22 = xbarPresent_prior + K * (z_t -
xbarPresent_prior)

% 11. Update the Covariance matrix
% CovPresent_post = CovPresent_prior - K_t * S_t *
(K_t)'
CovPresent_post = CovPresent_prior - K_t * H *
CovPresent_prior
% CovPresent_post22 = CovPresent_prior - K * H *
CovPresent_prior

Pe=CovPresent_post
xe=xbarPresent_post
xh=xbarPresent_post(1); yh=xbarPresent_post(2);
thetah=xbarPresent_post(3)

end
%% ----------------

function [XSigmaPresent_prior]= SigPointGen(n, lambda,
xbarPast_post, covPast_post, theta, dt)
% 1. Generate Sigma points
XSigmaPast_post=zeros(n,2*n+1)
XSigmaPast_post(:,1)= xbarPast_post
MSR =chol( (n+lambda)*covPast_post )  % chol: To
Calculate square root of error covariance
    for i=1:n
    XSigmaPast_post(:,i+1)  = xbarPast_post + (
MSR(i,:)' )
    XSigmaPast_post(:,n+i+1)= xbarPast_post - (
MSR(i,:)' )
    end
%%2. Propagate each sigma-point through prediction
%XSigmaPresent_prior=eval( g(XSigmaPast_post)) %
nx(2*n+1)--> 3x7
XSigmaPresent_prior=zeros(n, 2*n+1);
    for k=1:2*n+1
```

```
XSigmaPresent_prior(:,k)=fx(XSigmaPast_post(:,k),theta,
dt)  % XSigmaPresent_prior is the transformed sigma
points
    end
end

function xp=fx(x,theta,dt)
Ak = [1 0  cosd(theta)/theta;
        0 1  sind(theta)/theta;
        0 0  1];
A=eye(3)+dt*Ak;
 xp=A*x;
end
%
function  zpk=hx(x,theta)
zpk(1,1)=x(1)%sin(theta);
zpk(2,1)=x(2)%cos(theta);
zpk(3,1)=x(3);
end

function [xmean, CovMat]= UnsTrans( Sigmapoints, Wmean,
Wcov, NoiseCov)
%[xbarPresent_prior, CovPresent_prior]= UnsTrans(
XSigmaPresent_prior, Wsigma_mean, Wsigma_cov, R_t)

[n,~]=size(Sigmapoints)
% Calculate mean of predicted state
xmean=0;
for i=1:(2*n+1)
xmean= xmean + Wmean(i) * Sigmapoints(:,i) %
[nx(2*n+1)] :3x7 * ([1x(2*n+1)] : 1x7)' = [nx1] :3x1
end
% Calculate covariance of predicted state
CovMat=0;
for i=1:(2*n+1)
CovMat= CovMat + Wcov(i)* ( Sigmapoints(:,i) - xmean )
* ( Sigmapoints(:,i) - xmean )' + NoiseCov % 1 * (
[nx1]: 3x1 - 3x1) * ( [nx1]:3x1 -3x1)' + [n,n]:3x3
end

end
```