**The City College of New York**

Department of Electrical Engineering



# G3300 Advanced Mobile ROBOTICS

**Project 1**

# Spring 2017

# Professor: Jizhong Xiao

**Naoki Ishibashi**

1. What is the EKF(the Extended Kalman Filter) and UKF (Unscented Kalman Filter)

The Kalman Filter is a filter that works as a least square error optimizer, and for this to work, it is necessary that the system that you consider inside the filter is linear. In order to make state estimation on nonlinear systems, or parameter estimation, using the Kalman filter, one of the possible approaches is to linearize the system under investigation around its current state and force the filter to use this linearized version of your system as a model. This is the Extended Kalman Filter, or EKF. However, the EKF is not very stable and many times, when it does converge to the "right" solution, it is very slowly. To improve this filter, instead of using linearization to predict the behavior of the system under investigation, some authors started using the Unscented Transformation. Hence, the Kalman Filter with the Unscented transformation is called Unscented Kalman Filter, or UKF. This filter has some advantages when compared to the EKF, because the Unscented transformation somehow describes the nonlinear system better than the linearization, hence this filter converges to the right solution more rapidly. However, as the EKF, this filter may become unstable and results may be biased.

1. EKF

* EKF Function

function [] = ExtendedKalmanFilterLocalization2()

close all;

clear all;

disp('Extended Kalman Filter (EKF) sample program start!!')

time = 0;

endtime = 60; % [sec]

global dt;

dt = 0.1; % [sec]

nSteps = ceil((endtime - time)/dt);

result.time=[];

result.xTrue=[];

result.xd=[];

result.xEst=[];

result.z=[];

result.PEst=[];

result.u=[];

% State Vector [x y yaw v]'

xEst=[0 0 0 0]';

% True State

xTrue=xEst;

% Dead Reckoning

xd=xTrue;

% Observation vector [x y yaw v]'

z=[0 0 0 0]';

% Covariance Matrix for motion

Q=diag([0.1 0.1 toRadian(1) 0.05]).^2;

% Covariance Matrix for observation

R=diag([1.5 1.5 toRadian(3) 0.05]).^2;

% Simulation parameter

global Qsigma

Qsigma=diag([0.1 toRadian(20)]).^2; %[v yawrate]

global Rsigma

Rsigma=diag([1.5 1.5 toRadian(3) 0.05]).^2;%[x y z yaw v]

PEst = eye(4);

tic;

%movcount=0;

% Main loop

for i=1 : nSteps

time = time + dt;

% Input

u=doControl(time);

% Observation

[z,xTrue,xd,u]=Observation(xTrue, xd, u);

% ------ Kalman Filter --------

% Predict

xPred = f(xEst, u);

F=jacobF(xPred, u);

PPred= F\*PEst\*F' + Q;

% Update

H=jacobH(xPred);

y = z - h(xPred);

S = H\*PPred\*H' + R;

K = PPred\*H'\*inv(S);

xEst = xPred + K\*y;

PEst = (eye(size(xEst,1)) - K\*H)\*PPred;

% Simulation Result

result.time=[result.time; time];

result.xTrue=[result.xTrue; xTrue'];

result.xd=[result.xd; xd'];

result.xEst=[result.xEst;xEst'];

result.z=[result.z; z'];

result.PEst=[result.PEst; diag(PEst)'];

result.u=[result.u; u'];

%Animation (remove some flames)

if rem(i,5)==0

%hold off;

plot(result.xTrue(:,1),result.xTrue(:,2),'.b');hold on;

plot(result.z(:,1),result.z(:,2),'.g');hold on;

plot(result.xd(:,1),result.xd(:,2),'.k');hold on;

plot(result.xEst(:,1),result.xEst(:,2),'.r');hold on;

ShowErrorEllipse(xEst,PEst);

axis equal;

grid on;

drawnow;

%movcount=movcount+1;

%mov(movcount) = getframe(gcf); % get an animation frame

end

end

toc

%Save animation

%movie2avi(mov,'movie.avi');

DrawGraph(result);

function ShowErrorEllipse(xEst,PEst)

%Calculate error dispersion and display

Pxy=PEst(1:2,1:2); % get a covariance of x,y

[eigvec, eigval]=eig(Pxy); %calculate a Eigenvector and Eigenvalue

% search a index which has a bigger Eigenvalue

if eigval(1,1)>=eigval(2,2)

bigind=1;

smallind=2;

else

bigind=2;

smallind=1;

end

chi=9.21; % Square distribution value of error ellipse　99%

%draw ellipses

t=0:10:360;

a=sqrt(eigval(bigind,bigind)\*chi);

b=sqrt(eigval(smallind,smallind)\*chi);

x=[a\*cosd(t);

b\*sind(t)];

%calculate a angle in error ellipse

angle = atan2(eigvec(bigind,2),eigvec(bigind,1));

if(angle < 0)

angle = angle + 2\*pi;

end

% calculate a rotation in error ellipse

R=[cos(angle) sin(angle);

-sin(angle) cos(angle)];

x=R\*x;

plot(x(1,:)+xEst(1),x(2,:)+xEst(2))

function x = f(x, u)

% Motion Model

global dt;

F = [1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 0];

B = [

dt\*cos(x(3)) 0

dt\*sin(x(3)) 0

0 dt

1 0];

x= F\*x+B\*u;

function jF = jacobF(x, u)

% Jacobian of Motion Model

global dt;

jF=[

1 0 0 0

0 1 0 0

-dt\*u(1)\*sin(x(3)) dt\*u(1)\*cos(x(3)) 1 0

dt\*cos(x(3)) dt\*sin(x(3)) 0 1];

function z = h(x)

%Observation Model

H = [1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1 ];

z=H\*x;

function jH = jacobH(x)

%Jacobian of Observation Model

jH =[1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1];

function u = doControl(time)

%Calc Input Parameter

T=10; % [sec]

% [V yawrate]

V=1.0; % [m/s]

yawrate = 5; % [deg/s]

u =[ V\*(1-exp(-time/T)) toRadian(yawrate)\*(1-exp(-time/T))]';

function [z, x, xd, u] = Observation(x, xd, u)

%Calc Observation from noise prameter

global Qsigma;

global Rsigma;

x=f(x, u);% Ground Truth

u=u+Qsigma\*randn(2,1);%add Process Noise

xd=f(xd, u);% Dead Reckoning

z=h(x+Rsigma\*randn(4,1));%Simulate Observation

function []=DrawGraph(result)

%Plot Result

figure(1);

x=[ result.xTrue(:,1:2) result.xEst(:,1:2) result.z(:,1:2)];

set(gca, 'fontsize', 16, 'fontname', 'times');

plot(x(:,5), x(:,6),'.g','linewidth', 4); hold on;

plot(x(:,1), x(:,2),'-.b','linewidth', 4); hold on;

plot(x(:,3), x(:,4),'r','linewidth', 4); hold on;

plot(result.xd(:,1), result.xd(:,2),'--k','linewidth', 4); hold on;

title('EKF Localization Result', 'fontsize', 16, 'fontname', 'times');

xlabel('X (m)', 'fontsize', 16, 'fontname', 'times');

ylabel('Y (m)', 'fontsize', 16, 'fontname', 'times');

legend('Ground Truth','GPS','Dead Reckoning','EKF','Error Ellipse');

grid on;

axis equal;

function angle=Pi2Pi(angle)

%modify a angle from -pi to pi

angle = mod(angle, 2\*pi);

i = find(angle>pi);

angle(i) = angle(i) - 2\*pi;

i = find(angle<-pi);

angle(i) = angle(i) + 2\*pi;

function radian = toRadian(degree)

% degree to radian

radian = degree/180\*pi;

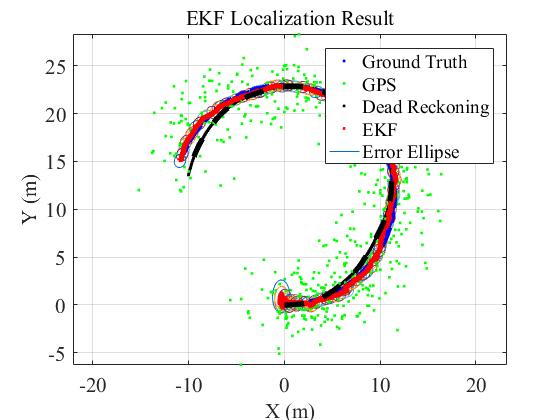
function degree = toDegree(radian)

% radian to degree

degree = radian/pi\*180;

Contact GitHub API Training Shop Blog About

* Result



1. UKF

* UKF Function

function [] = UnscentedKalmanFilterLocalization2()

close all;

clear all;

disp('Unscented Kalman Filter (UKF) sample program start!!')

time = 0;

endtime = 60; % [sec]

global dt;

dt = 0.1; % [sec]

nSteps = ceil((endtime - time)/dt);

result.time=[];

result.xTrue=[];

result.xd=[];

result.xEst=[];

result.z=[];

result.PEst=[];

result.u=[];

% State Vector [x y yaw v]'

xEst=[0 0 0 0]';

% True State

xTrue=xEst;

% Dead Reckoning

xd=xTrue;

% Observation vector [x y yaw v]'

z=[0 0 0 0]';

% Covariance Matrix for predict

Q=diag([0.1 0.1 toRadian(1) 0.05]).^2;

% Covariance Matrix for observation

R=diag([1.5 1.5 toRadian(3) 0.05]).^2;

% Simulation parameter

global Qsigma

Qsigma=diag([0.1 toRadian(20)]).^2;

global Rsigma

Rsigma=diag([1.5 1.5 toRadian(3) 0.05]).^2;

% UKF Parameter

alpha=0.001;

beta =2;

kappa=0;

n=length(xEst);%size of state vector

lamda=alpha^2\*(n+kappa)-n;

%calculate weights

wm=[lamda/(lamda+n)];

wc=[(lamda/(lamda+n))+(1-alpha^2+beta)];

for i=1:2\*n

wm=[wm 1/(2\*(n+lamda))];

wc=[wc 1/(2\*(n+lamda))];

end

gamma=sqrt(n+lamda);

PEst = eye(4);

%movcount=0;

tic;

% Main loop

for i=1 : nSteps

time = time + dt;

% Input

u=doControl(time);

% Observation

[z,xTrue,xd,u]=Observation(xTrue, xd, u);

% ------ Unscented Kalman Filter --------

% Predict

sigma=GenerateSigmaPoints(xEst,PEst,gamma);

sigma=PredictMotion(sigma,u);

xPred=(wm\*sigma')';

PPred=CalcSimgaPointsCovariance(xPred,sigma,wc,Q);

% Update

y = z - h(xPred);

sigma=GenerateSigmaPoints(xPred,PPred,gamma);

zSigma=PredictObservation(sigma);

zb=(wm\*sigma')';

St=CalcSimgaPointsCovariance(zb,zSigma,wc,R);

Pxz=CalcPxz(sigma,xPred,zSigma,zb,wc);

K=Pxz\*inv(St);

xEst = xPred + K\*y;

PEst=PPred-K\*St\*K';

% Simulation Result

result.time=[result.time; time];

result.xTrue=[result.xTrue; xTrue'];

result.xd=[result.xd; xd'];

result.xEst=[result.xEst;xEst'];

result.z=[result.z; z'];

result.PEst=[result.PEst; diag(PEst)'];

result.u=[result.u; u'];

%Animation (remove some flames)

if rem(i,5)==0

plot(xTrue(1),xTrue(2),'.b');hold on;

plot(z(1),z(2),'.g');hold on;

plot(xd(1),xd(2),'.k');hold on;

plot(xEst(1),xEst(2),'.r');hold on;

ShowErrorEllipse(xEst,PEst);

axis equal;

grid on;

drawnow;

%movcount=movcount+1;

%mov(movcount) = getframe(gcf);% get an animation frame

end

end

toc

%save an animation

%movie2avi(mov,'movie.avi');

DrawGraph(result);

function ShowErrorEllipse(xEst,PEst)

% calculate a Eigenvector and Eigenvalue

Pxy=PEst(1:2,1:2); % get a covariance of x,y

[eigvec, eigval]=eig(Pxy);% calculate a Eigenvector and Eigenvalue

% search a index which has a bigger Eigenvalue

if eigval(1,1)>=eigval(2,2)

bigind=1;

smallind=2;

else

bigind=2;

smallind=1;

end

chi=9.21; % Square distribution value of error ellipse　99%

% draw ellipses

t=0:10:360;

a=sqrt(eigval(bigind,bigind)\*chi);

b=sqrt(eigval(smallind,smallind)\*chi);

x=[a\*cosd(t);

b\*sind(t)];

% calculate a angle in error ellipse

angle = atan2(eigvec(bigind,2),eigvec(bigind,1));

if(angle < 0)

angle = angle + 2\*pi;

end

% calculate a rotation in error ellipse

R=[cos(angle) sin(angle);

-sin(angle) cos(angle)];

x=R\*x;

plot(x(1,:)+xEst(1),x(2,:)+xEst(2))

function sigma=PredictMotion(sigma,u)

% Sigma Points predition with motion model

for i=1:length(sigma(1,:))

sigma(:,i)=f(sigma(:,i),u);

end

function sigma=PredictObservation(sigma)

% Sigma Points predition with observation model

for i=1:length(sigma(1,:))

sigma(:,i)=h(sigma(:,i));

end

function P=CalcSimgaPointsCovariance(xPred,sigma,wc,N)

nSigma=length(sigma(1,:));

d=sigma-repmat(xPred,1,nSigma);

P=N;

for i=1:nSigma

P=P+wc(i)\*d(:,i)\*d(:,i)';

end

function P=CalcPxz(sigma,xPred,zSigma,zb,wc)

nSigma=length(sigma(1,:));

dx=sigma-repmat(xPred,1,nSigma);

dz=zSigma-repmat(zb,1,nSigma);

P=zeros(length(sigma(:,1)));

for i=1:nSigma

P=P+wc(i)\*dx(:,i)\*dz(:,i)';

end

function sigma=GenerateSigmaPoints(xEst,PEst,gamma)

sigma=xEst;

Psqrt=sqrtm(PEst);

n=length(xEst);

%Positive direction

for ip=1:n

sigma=[sigma xEst+gamma\*Psqrt(:,ip)];

end

%Negative direction

for in=1:n

sigma=[sigma xEst-gamma\*Psqrt(:,in)];

end

function x = f(x, u)

% Motion Model

global dt;

F = [1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 0];

B = [

dt\*cos(x(3)) 0

dt\*sin(x(3)) 0

0 dt

1 0];

x= F\*x+B\*u;

function z = h(x)

%Observation Model

H = [1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1 ];

z=H\*x;

function u = doControl(time)

%Calc Input Parameter

T=10; % [sec]

% [V yawrate]

V=1.0; % [m/s]

yawrate = 5; % [deg/s]

u =[ V\*(1-exp(-time/T)) toRadian(yawrate)\*(1-exp(-time/T))]';

%Calc Observation from noise prameter

function [z, x, xd, u] = Observation(x, xd, u)

global Qsigma;

global Rsigma;

x=f(x, u);% Ground Truth

u=u+Qsigma\*randn(2,1);%add Process Noise

xd=f(xd, u);% Dead Reckoning

z=h(x+Rsigma\*randn(4,1));%Simulate Observation

function []=DrawGraph(result)

%Plot Result

figure(1);

x=[ result.xTrue(:,1:2) result.xEst(:,1:2) result.z(:,1:2)];

set(gca, 'fontsize', 16, 'fontname', 'times');

plot(x(:,5), x(:,6),'.g','linewidth', 4); hold on;

plot(x(:,1), x(:,2),'-.b','linewidth', 4); hold on;

plot(x(:,3), x(:,4),'r','linewidth', 4); hold on;

plot(result.xd(:,1), result.xd(:,2),'--k','linewidth', 4); hold on;

title('UKF Localization Result', 'fontsize', 16, 'fontname', 'times');

xlabel('X (m)', 'fontsize', 16, 'fontname', 'times');

ylabel('Y (m)', 'fontsize', 16, 'fontname', 'times');

legend('Ground Truth','GPS','Dead Reckoning','UKF');

grid on;

axis equal;

function radian = toRadian(degree)

% degree to radian

radian = degree/180\*pi;

function degree = toDegree(radian)

% radian to degree

degree = radian/pi\*180;

* Result

