**Robotics**

**Exercise 7.1. EKF SLAM**

The exercise’s appendix includes a code implementing a SLAM algorithm based on the Extended Kalman Filter, but it’s incomplete at some points. Employing and extending it, answer the following questions:

**1. –** What represents PPred and why is it build in that way in the code? Which are its dimensions? and those of the matrices used to build it? (PPredvv, PPredvm and PPredmm)

**2. –** Build the state Jacobian jH used in the Kalman filter during the update step when a previously perceived landmark is seen again. Employ the output of the GetObsJacs function. Analyze both its size and its content throughout the SLAM simulation.

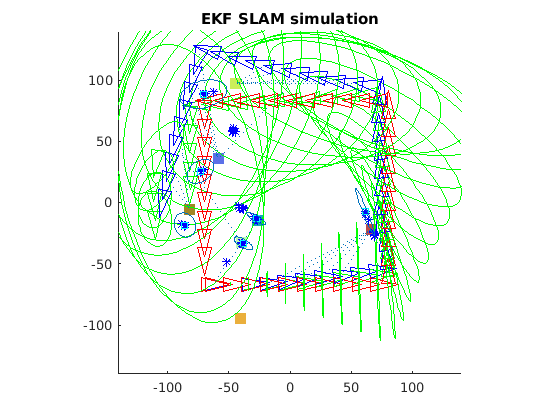
**3. –** Store in each iteration the determinant of the matrices of covariance of the robot pose and the localization of each feature and plot them. Do the same with the error of the localization of the pose and the features. Use the variables PFeatDetStore, FeatErrStore, PXErrStore and XErrStore.

**3.1 –** Now the program is complete. Run it a few times and describe the obtained results. Explain the meaning of each element appearing in the figure resulting from the execution of the SLAM algorithm. (See a running example in the next page).

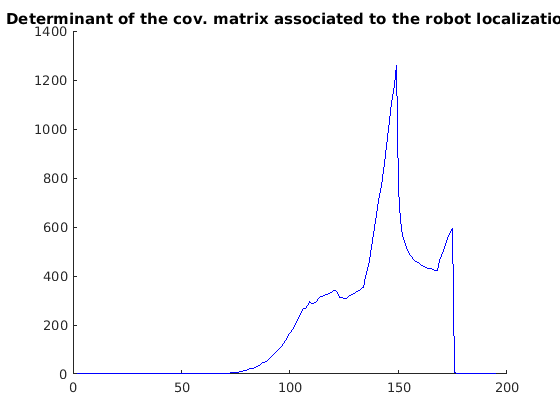
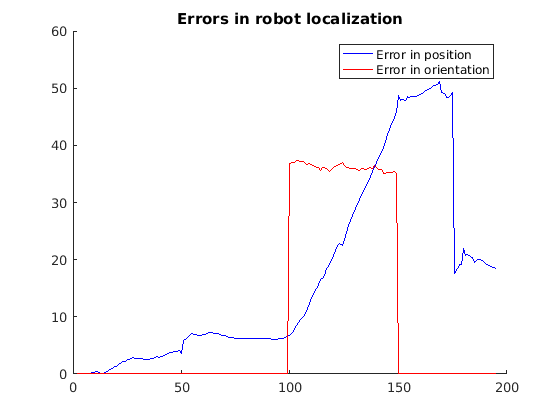
**3.2 –** Play with different numbers of landmarks and discuss the results.

**4. –** **Optional**. The provided code only employs a feature per iteration. Change the code to consider all the features within the FOV of the robot during the update step of the EKF.

**Example execution results:**

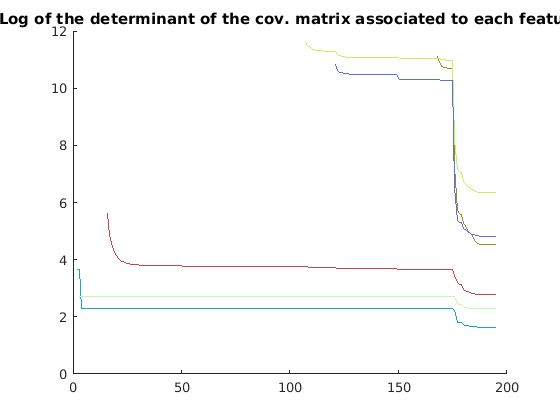


Robotics course



Robotics course

Robotics course



Robotics course

Robotics course

**Appendix: Exercise’s code to analyse/extend**

function EKF\_SLAM

clear all;

close all;

% Map configuration

nFeatures = 15;

MapSize = 200;

[Map,colors] = CreateMap(MapSize,nFeatures);

%how often shall we draw?

DrawEveryNFrames = 5;

mode = 'one\_landmark\_in\_fov';

%mode = 'landmarks\_in\_fov';

% Robot base characterization

SigmaX = 0.01; % Standard deviation in the x axis

SigmaY = 0.01; % Standard deviation in the y axis

SigmaTheta = 1.5\*pi/180; % Bearing standar deviation

R = diag([SigmaX^2 SigmaY^2 SigmaTheta^2]); % Cov matrix

% Covariances for our very bad&expensive sensor (in the system <d,theta>)

Sigma\_r = 1.1;

Sigma\_theta = 5\*pi/180;

Q = diag([Sigma\_r,Sigma\_theta]).^2;

fov = pi\*2/3;

max\_range = 100;

xRobot = [-MapSize/3 -MapSize/3 0]';

xRobotTrue = xRobot;

%initial conditions - no map:

xEst = xRobot;

PEst = zeros(3,3);

MappedFeatures = NaN\*zeros(nFeatures,2);

% Drawings

figure(1); hold on; grid off; axis equal;

axis([-MapSize/2-40 MapSize/2+40 -MapSize/2-40 MapSize/2+40]);

for i\_feat=1:nFeatures

plot(Map(1,i\_feat),Map(2,i\_feat),'s','Color',colors(i\_feat,:), ...

'MarkerFaceColor',colors(i\_feat,:),'MarkerSize',10); ,:));

end

set(gcf,'doublebuffer','on');

hObsLine = line([0,0],[0,0]);

set(hObsLine,'linestyle',':');

DrawRobot(xEst(1:3),'g');

DrawRobot(xRobotTrue,'b');

DrawRobot(xRobot,'r');

hFOV = drawFOV(xRobotTrue,fov,max\_range,'b');

PlotEllipse(xEst(1:3),PEst(1:3,1:3),5,'g');

pause;

delete(hFOV);

u = [3;0;0];

% Number of steps

nSteps = 195;

turning = 50;

% Storage:

PFeatDetStore = NaN\*zeros(nFeatures,nSteps);

FeatErrStore = NaN\*zeros(nFeatures,nSteps);

PXErrStore = NaN\*zeros(nSteps,1);

XErrStore = NaN\*zeros(2,nSteps); % error in position and angle

for k = 2:nSteps

%

% Move the robot with a control action u

%

u(3) = 0;

if (mod(k,turning)==0) u(3)=pi/2;end

xRobot = tcomp(xRobot,u); % New pose without noise

noise = sqrt(R)\*randn(3,1); % Generate noise

noisy\_u = tcomp(u,noise); % Apply noise to the control action

xRobotTrue = tcomp(xRobotTrue,noisy\_u);

% Useful vbles

xVehicle = xEst(1:3);

xMap = xEst(4:end);

%

% Prediction step

%

xVehiclePred = tcomp(xVehicle,u);

PPredvv = J1(xVehicle,u)\* PEst(1:3,1:3) \*J1(xVehicle,u)' + J2(xVehicle,u)\* R \* J2(xVehicle,u)';

PPredvm = J1(xVehicle,u)\*PEst(1:3,4:end);

PPredmm = PEst(4:end,4:end);

xPred = [xVehiclePred;xMap];

PPred = [PPredvv PPredvm;

PPredvm' PPredmm];

% Get new observation/s

if strcmp(mode,'one\_landmark\_in\_fov')

% Get a random observations within the fov of the sensor

[MapInFov,iFeatures] = getObservationsInsideFOV(xRobotTrue,Map,fov,max\_range);

if ~isempty(MapInFov)

[z,iFeature] = getRandomObservationFromPose(xRobotTrue,MapInFov,Q,iFeatures);

else

z = [];

end

elseif strcmp(mode,'landmarks\_in\_fov')

%

% Point 4

%

000000000;

end

%

% Update step

%

if(~isempty(z))

%have we seen this feature before?

if( ~isnan(MappedFeatures(iFeature,1)))

%predict observation: find out where it is in state vector

FeatureIndex = MappedFeatures(iFeature,1);

xFeature = xPred(FeatureIndex:FeatureIndex+1);

zPred = getRangeAndBearing(xVehiclePred,xFeature);

% get observation Jacobians

[jHxv,jHxf] = GetObsJacs(xVehicle,xFeature);

% Fill in state jacobian

%

% Point 2, Build jH from JHxv and jHxf

%

jH = 000000000;

% Do Kalman update:

Innov = z-zPred;

Innov(2) = AngleWrap(Innov(2));

S = jH\*PPred\*jH'+Q;

W = PPred\*jH'\*inv(S);

xEst = xPred+ W\*Innov;

PEst = PPred-W\*S\*W';

%ensure P remains symmetric

PEst = 0.5\*(PEst+PEst');

else

% this is a new feature add it to the map....

nStates = length(xEst);

xFeature = xVehiclePred(1:2) + [z(1)\*cos(z(2)+xVehiclePred(3));z(1)\*sin(z(2)+xVehiclePred(3))];

xEst = [xPred;xFeature]; %augmenting state vector

[jGxv, jGz] = GetNewFeatureJacs(xVehicle,z);

M = [eye(nStates), zeros(nStates,2);% note we don't use jacobian w.r.t vehicle

jGxv zeros(2,nStates-3) , jGz];

PEst = M\*blkdiag(PEst,Q)\*M';

%remember this feature as being mapped we store its ID and position in the state vector

MappedFeatures(iFeature,:) = [length(xEst)-1, length(xEst)];

end;

else

xEst = xPred;

PEst = PPred;

end;

%

% Point 3, Robot pose and features localization errors and determinants

%

000000000;

000000000;

000000000;

000000000;

% Drawings

if(mod(k-2,DrawEveryNFrames)==0)

% Robot estimated, real, and ideal poses, fov and uncertainty

DrawRobot(xEst(1:3),'g');

DrawRobot(xRobotTrue,'b');

DrawRobot(xRobot,'r');

hFOV = drawFOV(xRobotTrue,fov,max\_range,'b');

PlotEllipse(xEst(1:3),PEst(1:3,1:3),5,'g');

% A line to the observed feature

if(~isnan(z))

hLine = line([xRobotTrue(1),Map(1,iFeature)],[xRobotTrue(2),Map(2,iFeature)]);

set(hLine,'linestyle',':');

end;

% The uncertainty of each perceived landmark

n = length(xEst);

nF = (n-3)/2;

hEllipses = [];

for i = 1:nF

iF = 3+2\*i-1;

plot(xEst(iF),xEst(iF+1),'b\*');

hEllipse = PlotEllipse(xEst(iF:iF+1),PEst(iF:iF+1,iF:iF+1),3);

hEllipses = [hEllipses hEllipse];

end

drawnow; % flush pending drawings events

pause;

% Clean a bit

delete(hFOV);

for i=1:size(hEllipses,2)

delete(hEllipses(i));

end

end

end

% Draw the final estimated positions and uncertainties of the features

n = length(xEst);

nF = (n-3)/2;

for i = 1:nF

iF = 3+2\*i-1;

plot(xEst(iF),xEst(iF+1),'cs');

PlotEllipse(xEst(iF:iF+1),PEst(iF:iF+1,iF:iF+1),3);

end;

%

% Draw erros and determinants of the location of the robot and the

% featuers

%

figure(2); hold on;

title('Errors in robot localization');

plot(XErrStore(1,:),'b');

plot(XErrStore(2,:),'r');

legend('Error in position','Error in orientation')

figure(3); hold on;

title('Determinant of the cov. matrix associated to the robot localization');

xs = 1:nSteps;

plot(PXErrStore(:),'b');

figure(4); hold on;

title('Errors in features localization');

figure(5); hold on;

title('Log of the determinant of the cov. matrix associated to each feature');

for i=1:nFeatures

figure(5)

h = plot(log(PFeatDetStore(i,:)));

set(h,'Color',colors(i,:));

figure(4)

h = plot(FeatErrStore(i,:));

set(h,'Color',colors(i,:));

end

%-------------------------------------------------------------------------%

function [Map,colors] = CreateMap(MapSize,nFeatures)

Map = zeros(2,nFeatures);

colors = zeros(nFeatures,3);

for i\_feat = 1:nFeatures

Map(:,i\_feat) = MapSize\*rand(2,1)-MapSize/2;

colors(i\_feat,:) = [rand rand rand];

end

%-------------------------------------------------------------------------%

function [jHxv,jHxf] = GetObsJacs(xPred, xFeature)

jHxv = zeros(2,3);jHxf = zeros(2,2);

Delta = (xFeature-xPred(1:2));

r = norm(Delta);

jHxv(1,1) = -Delta(1) / r;

jHxv(1,2) = -Delta(2) / r;

jHxv(2,1) = Delta(2) / (r^2);

jHxv(2,2) = -Delta(1) / (r^2);

jHxv(2,3) = -1;

jHxf(1:2,1:2) = -jHxv(1:2,1:2);

%-------------------------------------------------------------------------%

function [jGx,jGz] = GetNewFeatureJacs(Xv, z)

theta = Xv(3,1);

r = z(1);

bearing = z(2);

jGx = [ 1 0 -r\*sin(theta + bearing);

0 1 r\*cos(theta + bearing)];

jGz = [ cos(theta + bearing) -r\*sin(theta + bearing);

sin(theta + bearing) r\*cos(theta + bearing)];

%-------------------------------------------------------------------------%

function h = drawFOV(x,fov,max\_range,c)

if nargin < 4; c = 'b'; end

alpha = fov/2;

angles = -alpha:0.01:alpha;

nAngles = size(angles,2);

arc\_points = zeros(2,nAngles);

for i=1:nAngles

arc\_points(1,i) = max\_range\*cos(angles(i));

arc\_points(2,i) = max\_range\*sin(angles(i));

aux\_point = tcomp(x,[arc\_points(1,i);arc\_points(2,i);1]);

arc\_points(:,i) = aux\_point(1:2);

end

h = plot([x(1) arc\_points(1,:) x(1)],[x(2) arc\_points(2,:) x(2)],c);

%-------------------------------------------------------------------------%

function [MapInFov,iFeatures] = getObservationsInsideFOV(x,Map,fov,max\_range)

nLandmarks = size(Map,2);

MapInFov = [];

iFeatures = [];

z = zeros(2,1);

for i\_landmark = 1:nLandmarks

Delta\_x = Map(1,i\_landmark) - x(1);

Delta\_y = Map(2,i\_landmark) - x(2);

z(1) = norm([Delta\_x Delta\_y]); % Range

z(2) = atan2(Delta\_y,Delta\_x) - x(3); % Bearing

z(2) = AngleWrap(z(2));

if (z(2) < fov/2) && (z(2) > -fov/2) && (z(1) < max\_range)

MapInFov = [MapInFov Map(:,i\_landmark)];

iFeatures = [iFeatures i\_landmark];

end

end

%-------------------------------------------------------------------------%

function [z,iFeature] = getRandomObservationFromPose(x,Map,Q,iFeatures)

nLandmarks = size(Map,2);

iFeature = randi(nLandmarks);

landmark = Map(:,iFeature);

z = getRangeAndBearing(x,landmark,Q);

if nargin == 4

iFeature = iFeatures(iFeature);

end

%-------------------------------------------------------------------------%

function z = getRangeAndBearing(x,landmark,Q)

Delta\_x = landmark(1,:) - x(1);

Delta\_y = landmark(2,:) - x(2);

z(1,:) = sqrt(Delta\_x.^2 + Delta\_y.^2); % Range

z(2,:) = atan2(Delta\_y,Delta\_x) - x(3); % Bearing

if nargin == 3

z = z + sqrt(Q)\*rand(2,1); % Adding noise

end

z(2,:) = AngleWrap(z(2,:));