**Robotics**

**Exercise 5.2 EKF Localization (Range-Bearing)**

In this exercise we are going to implement the EKF localization algorithm using a map of landmarks and a sensor providing range and bearing measurements from the robot pose to such landmarks. You can use the attached code to ease the programming task.

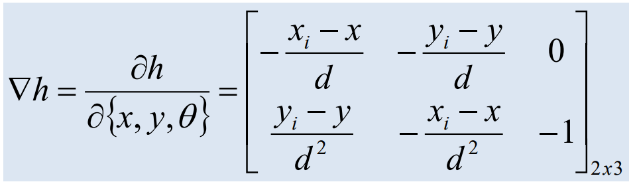
**1.- Getting an observation to a random landmark**. Using the information provided by the CreateMap function in the code, implement your own function named getRandomObservationFromPose that, given the robot pose, randomly selects a landmark and returns an observation from the range-bearing sensor using the getRangeAndBearing function (that you have also to implement). *Hint: use the randi() function.*

**2.- Adding uncertainty to the sensor model.** Modify the previous functions to also consider the uncertainty in the sensor measurements defined by the matrix (Q in the code):

**3.- Simulating the robot motion.** In the exercise 3.1 we commanded a mobile robot to follow a squared trajectory. Add random noise to each motion command (noisy\_u)based on the following matrix, and update the true robot pose (xTrue):

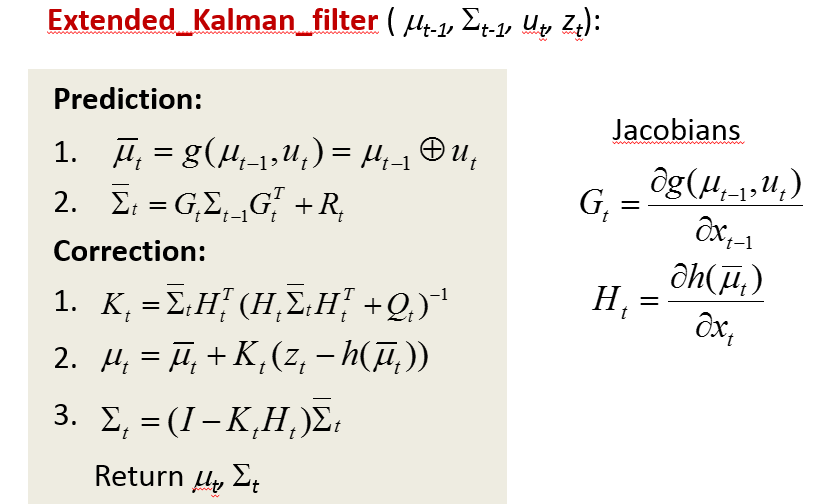
Simulate that in each iteration the robot gathers an observation from the sensor (to a random landmark from the map). Draws a line from the robot to the landmark. *Hint: you can use line([x0, x1],[y0, y1]); for that.*

**4.- Fixing the robot pose according to the map.** Given that the position of the landmarks in the map is known, we can use this information in a Kalman filter, in our case an EKF. For that we need to implement the Jacobians of the observation model. Implement a function that, given the predicted pose in the first step of the Kalman filter, the selected landmark and the map, returns such Jacobian.



function jH = GetObsJac(xPred, Landmark, Map)

**5.- EKF filter.** Employing the previously coded functions, implement the EKF filter (both prediction and correction steps) and show the estimated pose and its uncertainty.



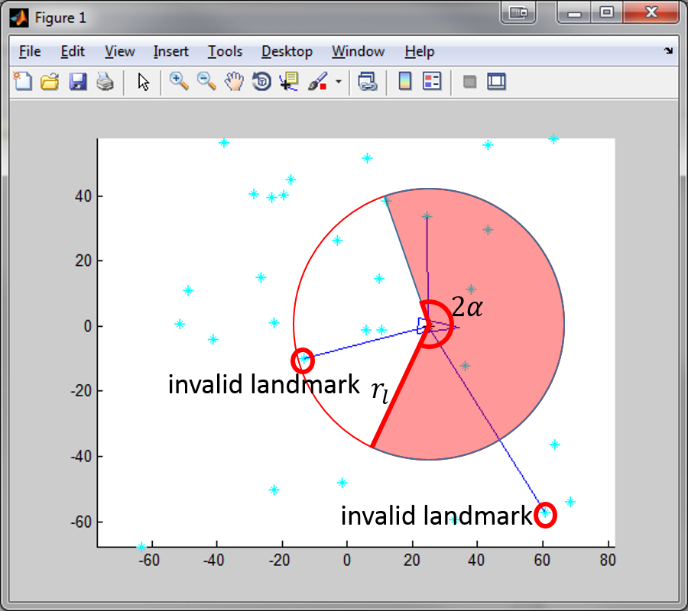
Please, notice that *Rt* is the covariance of the motion *ut* in the coordinate system of the predicted pose (), then *(Note: J2 is our popular Jacobian for the motion command, you could also use J1)*:

****

The figure below shown an example of the execution of the EKF localization algorithm with the code implemented until this point.

****

**6.- Modifying the sensor information.** Sensors exhibit certain physical limitations regarding their field of view and maximum operating distance (max. Range). Modify the code to consider that the sensor can only provide information from a random landmark in a limited range and a limited orientation with respect to the robot pose (implement the getLandmarksInsideFOV function for that). That is the 'one\_landmark\_in\_fov' mode. It could happen that any landmark exists in the field of view of the sensor, so the robot couldn’t gather sensory information in that iteration. Discuss how the uncertainty evolves.



**7.- Adding more information from the sensor**. Usually, sensors do not provide information from only a landmark. Modify the code so in each observation the sensor returns the measurement to *k* landmarks. This implies modifications in the functions for computing the Jacobian of the sensor model (it now has 2\*k rows and 3 columns). The figure below shows an example of the execution of EKF using information from all the landmarks within the FOV:



**Código anexo: Esqueleto de la práctica a rellenar**

function EKFLocalization

clear; close all;

% Map configuration

Size = 50;

NumLandmarks = 10;

Map=CreateMap(NumLandmarks, Size); % Create map of size [Size\*2 Size\*2]

mode = 'one\_landmark';

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

%mode = 'landmarks\_in\_fov';

% Sensor characterization

SigmaR = 1; % Standard deviation of the range

SigmaB = 0.7; % Standard deviation of the bearing

Q = diag([SigmaR^2 SigmaB^2]); % Cov matrix

fov = pi/2; % field of view = 2\*alpha

max\_range = Size; % maximum sensor measurement range

% Robot base characterization

SigmaX = 0.8; % Standard deviation in the x axis

SigmaY = 0.8; % Standard deviation in the y axis

SigmaTheta = 0.1; % Bearing standar deviation

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

% Initialization of poses

x = [-Size+Size/3 -Size+Size/3 pi/2]'; % Ideal robot pose

xTrue = [-Size+Size/3 -Size+Size/3 pi/2]'; % Real robot pose

xEst = [-Size+Size/3 -Size+Size/3 pi/2]'; % Estimated robot pose by EKF

sEst = zeros(3,3); % Uncertainty of estimated robot pose

% Drawings

plot(Map(1,:),Map(2,:),'sc');

axis([-Size-5 Size+5 -Size-5 Size+5]);

hold on;

DrawRobot(x,'r');

DrawRobot(xTrue,'b');

DrawRobot(xEst,'g');

PlotEllipse(xEst,sEst,4,'g');

nSteps = 20; % Number of motions

turning = 5; % Number of motions before turning (square path)

u = [(2\*Size-2\*Size/3)/turning;0;0]; % Control action

pause;

% Let's go!

for k = 1:nSteps-3 % Main loop

u(3) = 0;

if mod(k,turning) == 0 % Turn?

u(3) = -pi/2;

end

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

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

noisy\_u = --------- % Apply noise to the control action

xTrue = --------- % New noisy pose (real robot pose)

% Get sensor observation/s

if strcmp(mode,'one\_landmark')

---------

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

---------

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

---------

end

%

% EKF Localization

%

% Prediction

---------

% Correction (You need to compute the gain k and the innovation z-z\_p)

---------

% Drawings

% Plot the FOV of the robot

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

h = drawFOV(xTrue,fov,max\_range,'g');

end

% Plot Robot pose and uncertainty

DrawRobot(x,'r'); % Ideal Pose (noise free)

DrawRobot(xTrue,'b'); % Real pose (noisy)

DrawRobot(xEst,'g'); % EKF estimation of the pose (motion+obs)

PlotEllipse(xEst,sEst,3,'g'); %Uncertainty of EKF estimation

pause;

%Delete the previous FOV

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

delete(h);

end

end;

end % main

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Map=CreateMap(NumLandmarks, Size)

Map=Size\*2\*rand(2,NumLandmarks)-Size;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

---------

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function z = getRangeAndBearing(x,landmark,Q)

---------

if nargin == 3 % Add noise

---------

end

% utilize AngleWrap to ensure that the measurement angle is correct

---------

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function jH = getObsJac(xPred,Landmark, Map)

---------

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function MapInFov = getLandmarksInsideFOV(x,Map,fov,max\_range)

---------

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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);

end