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):

$$\Sigma_s = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\varphi^2 \end{bmatrix}$$

```
function [z,landmark] = getRandomObservationFromPose(x,Map,Q)
    pos = randi(size(Map,2));
    landmark=Map(:,pos);
    z=getRangeAndBearing(x,landmark,Q);
end

function z = getRangeAndBearing(x,landmark,Q)
    d=pdist2(landmark(1:2)',x(1:2)');
    xi=landmark(1); yi=landmark(2);
    angle=atan2(yi-x(2),xi-x(1))-x(3);
    z=[d;angle];
    if nargin == 3
       z=z+sqrt(Q)*randn(2,1);
    end
z(2)=AngleWrap(z(2));
end
```

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`):

$$\Sigma_{u_t} = \begin{bmatrix} \sigma_{\Delta x}^2 & 0 & 0 \\ 0 & \sigma_{\Delta y}^2 & 0 \\ 0 & 0 & \sigma_{\Delta \theta}^2 \end{bmatrix}$$

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.*

```
x = tcomp(x,u);
noise = sqrt(R)*randn(3,1);
noisy_u = u+noise;
xTrue = tcomp(xTrue,noisy_u);
[z,landmark] = getRandomObservationFromPose(xTrue,Map,Q);
x0=xTrue(1);y0=xTrue(2);
x1=landmark(1); y1=landmark(2);
line([x0, x1],[y0, y1],'LineStyle','--');
```

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.

$$\nabla h = \frac{\partial h}{\partial\{x, y, \theta\}} = \begin{bmatrix} -\frac{x_i - x}{d} & -\frac{y_i - y}{d} & 0 \\ \frac{y_i - y}{d^2} & -\frac{x_i - x}{d^2} & -1 \end{bmatrix}_{2x3}$$

```
function jH = GetObsJac(xPred, Landmark, Map)
```

```
diff=Landmark-xPred(1:2);
d=pdist2(Landmark',xPred(1:2)');
jH=[-diff(1)/d    -diff(2)/d       0;
      diff(2)/d^2 -diff(1)/d^2    -1];
```

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.

Extended_Kalman_filter (μ_{t-1}, Σ_{t-1}, u_t, z_t):

Prediction:

1. $\bar{\mu}_t = g(\mu_{t-1}, u_t) = \mu_{t-1} \oplus u_t$
2. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

Correction:

1. $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
2. $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
3. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

Return μ_t, Σ_t

Jacobians

$$G_t = \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}$$

$$H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$$

Please, notice that R_t is the covariance of the motion u_t in the coordinate system of the predicted pose ($\bar{x}_t$), then *(Note: J_2 is our popular Jacobian for the motion command, you could also use J_1)*:

$$R_t = J_2 \Sigma_{u_t} J_2^t \qquad with \;\; J_2 = \frac{\partial g(\mu_{t-1}, u_t)}{\partial u_t}$$

```
% Prediction
jG=J1(xEst,u);
j2=J2(xEst,u);
Rt=j2*R*j2';
PredU = tcomp(xEst,u);
PredS = jG*sEst*jG'+Rt;
xEst=PredU;
sEst=PredS;
% Correction (You need to compute the gain k and the innovation z-
z_p)
jH=getObsJac(PredU,landmark);
Kt = PredS*jH'/(jH*PredS*jH'+Q);
hu=getRangeAndBearing(PredU,landmark);
xEst = PredU + Kt*(z-hu);
sEst = (eye(3)-Kt*jH)*PredS;
```

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

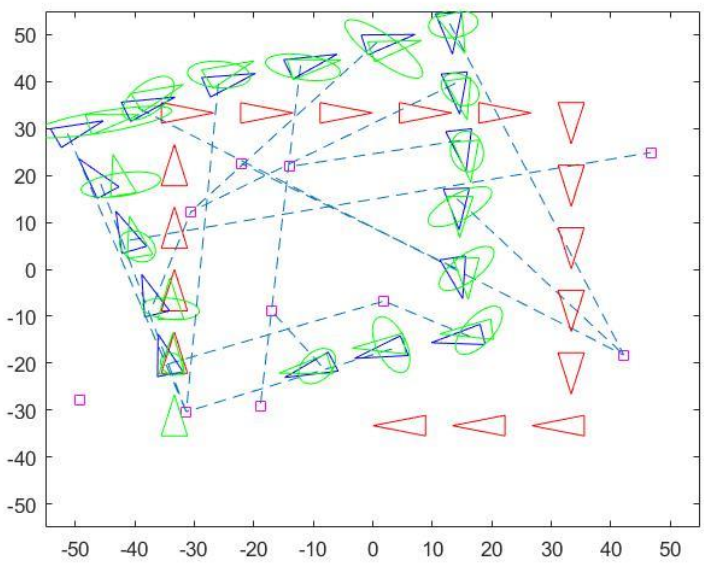

Image 1. EKF localization for one landmark random

As we see in the image, the covariance of the estimated robot changes in each iteration. It decreases when the robot takes the measure of a different landmark which it did take before.

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 r_l and a limited orientation $\pm\alpha$ 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.

```
MapInFov = getLandmarksInsideFOV(xTrue,Map,fov,max_range);
if size(MapInFov,1)~=0
      [z,landmark]=getRandomObservationFromPose(xTrue,MapInFov,Q);
      x0=xTrue(1);y0=xTrue(2);
      x1=landmark(1); y1=landmark(2);
      line([x0, x1],[y0, y1],'LineStyle','--');
end
[...]
if size(MapInFov,1)~=0
% Correction (You need to compute the gain k and the innovation z-
z_p)
      jH=getObsJac(PredU,landmark);
      Kt = PredS*jH'/(jH*PredS*jH'+Q);
      hu=getRangeAndBearing(PredU,landmark);
      xEst = PredU + Kt*(z-hu);
      sEst = (eye(3)-Kt*jH)*PredS;
end
```

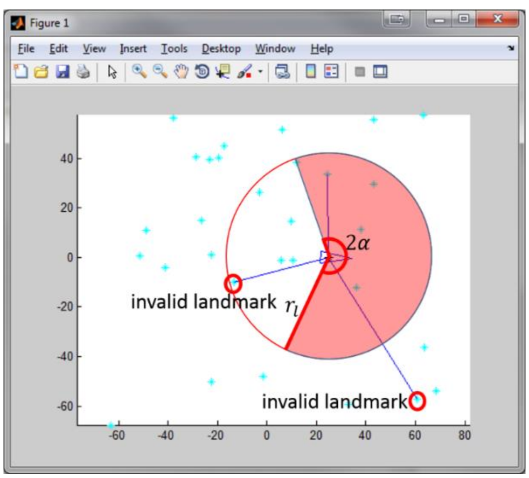

Image 2. Valid landmarks for the FOV algorithm

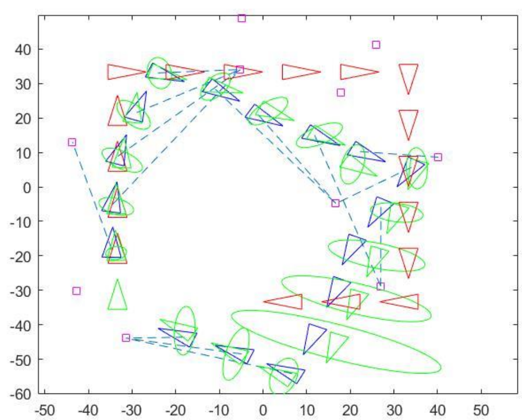

Image 3. EKF localization for one landmark random in the FOV

As we see in the image 3, when the robot does not capture any landmark, the estimated robot moves like the ideal robot and its covariance increase like when the robot moves until it capture a landmark. Since there, the covariance decreases because now it can localize the robot. It happens because in the prediction phase we do a convolution of the gaussians (sum of gaussian) and it cannot conduit to the correction phase.

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).

```
MapInFov = getLandmarksInsideFOV(xTrue,Map,fov,max_range);
if size(MapInFov,1)~=0
      n=size(MapInFov,2);
      z=zeros(2*n,1);
      k=1;
      for i=1:n
            landmark=MapInFov(:,i);
            zn=getRangeAndBearing(xTrue,landmark);
            z(k)=zn(1);
            z(k+1)=zn(2);
            k=k+2;
            x0=xTrue(1);y0=xTrue(2);
            x1=landmark(1); y1=landmark(2);
            line([x0, x1],[y0, y1],'LineStyle','--');
      end
end
[...]
if size(MapInFov,1)~=0
% Correction (You need to compute the gain k and the innovation z-
z_p)
      if strcmp(mode,'landmarks_in_fov')
            jH=getObsJac(PredU,[],MapInFov);
            Kt=PredS*jH'/(jH*PredS*jH'+diag(repmat(diag(Q),n,1)));
            n=size(MapInFov,2);
            hu=zeros(2*n,1);
            k=1;
            for i=1:n
                  hun=getRangeAndBearing(PredU,MapInFov(:,i));
                  hu(k)=hun(1);
                  hu(k+1)=hun(2);
                  k=k+2;
            end
      else
             jH=getObsJac(PredU,landmark);
             Kt = PredS*jH'/(jH*PredS*jH'+Q);
             hu=getRangeAndBearing(PredU,landmark);
     end
     xEst = PredU + Kt*(z-hu);
     sEst = (eye(3)-Kt*jH)*PredS;
end
function jH = getObsJac(xPred,Landmark, Map)
         if nargin == 3
            n=size(Map,2);
            jH=zeros(2*n,3);
            k=1;
            for i=1:n
                jh=getObsJac(xPred,Map(:,i));
                jH(k:k+1,:)=jh;
                k=k+2;
```

```
            end
        else
          [...]
        end
end
```

The figure below shows an example of the execution of EKF using information from all the landmarks within the FOV:

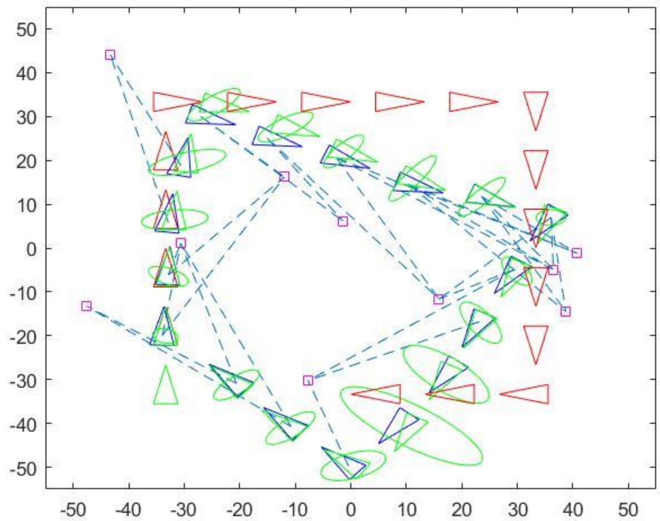

Image 4. EKF localization for landmarks in the FOV

In this case, the algorithm function like the previous one. However, since it captures all the landmark in his FOV view, the estimated robot covariance is smaller than the previous one because in the correction phase we do the multiplication of all the gaussians. And since we have a lot of gaussians, the covariance will be much smaller than usual. Despite of this advantage, when it does not capture any landmark, it suffers the same problem as the last one.

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 = u+noise;              % Apply noise to the control action
    xTrue = tcomp(xTrue,noisy_u);   % New noisy pose (real robot pose)

    % Get sensor observation/s
    if strcmp(mode,'one_landmark')
      [z,landmark] = getRandomObservationFromPose(xTrue,Map,Q);
      x0=xTrue(1);y0=xTrue(2);
      x1=landmark(1); y1=landmark(2);
      line([x0, x1],[y0, y1],'LineStyle','--');
    elseif strcmp(mode,'one_landmark_in_fov')
      MapInFov = getLandmarksInsideFOV(xTrue,Map,fov,max_range);
      if size(MapInFov,1)~=0
        [z,landmark] = getRandomObservationFromPose(xTrue,MapInFov,Q);
        x0=xTrue(1);y0=xTrue(2);
        x1=landmark(1); y1=landmark(2);
        line([x0, x1],[y0, y1],'LineStyle','--');
      end
    elseif strcmp(mode,'landmarks_in_fov')
      MapInFov = getLandmarksInsideFOV(xTrue,Map,fov,max_range);
        if size(MapInFov,1)~=0
            n=size(MapInFov,2);
            z=zeros(2*n,1);
            k=1;
            for i=1:n
                landmark=MapInFov(:,i);
                zn=getRangeAndBearing(xTrue,landmark);
                z(k)=zn(1);
                z(k+1)=zn(2);
                k=k+2;
                x0=xTrue(1);y0=xTrue(2);
                x1=landmark(1); y1=landmark(2);
                line([x0, x1],[y0, y1],'LineStyle','--');
            end
        end
    end

    %
    % EKF Localization
    %

    % Prediction
    jG=J1(xEst,u);
    j2=J2(xEst,u);
    Rt=j2*R*j2';
    PredU = tcomp(xEst,u);
    PredS = jG*sEst*jG'+Rt;
    xEst=PredU;
    sEst=PredS;

    if size(MapInFov,1)~=0
% Correction (You need to compute the gain k and the innovation z-z_p)
        if strcmp(mode,'landmarks_in_fov')
            jH=getObsJac(PredU,[],MapInFov);
            Kt = PredS*jH'/(jH*PredS*jH'+diag(repmat(diag(Q),n,1)));
            n=size(MapInFov,2);
            hu=zeros(2*n,1);
            k=1;
            for i=1:n
```

```
                hun=getRangeAndBearing(PredU,MapInFov(:,i));
                hu(k)=hun(1);
                hu(k+1)=hun(2);
                k=k+2;
            end
        else
            jH=getObsJac(PredU,landmark);
            Kt = PredS*jH'/(jH*PredS*jH'+Q);
            hu=getRangeAndBearing(PredU,landmark);
        end
        xEst = PredU + Kt*(z-hu);
        sEst = (eye(3)-Kt*jH)*PredS;
    end

    % 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)
    pos = randi(size(Map,2));
    landmark=Map(:,pos);
    z=getRangeAndBearing(x,landmark,Q);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function z = getRangeAndBearing(x,landmark,Q)

    d=pdist2(landmark(1:2)',x(1:2)');
    xi=landmark(1); yi=landmark(2);
    angle=atan2(yi-x(2),xi-x(1))-x(3);
    z=[d;angle];
    if nargin == 3 % Add noise
       z=z+sqrt(Q)*randn(2,1);
    end

% utilize AngleWrap to ensure that the measurement angle is correct
z(2)=AngleWrap(z(2));

end

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


function jH = getObsJac(xPred,Landmark, Map)
       if nargin == 3
            n=size(Map,2);
            jH=zeros(2*n,3);
            k=1;
            for i=1:n
                jh=getObsJac(xPred,Map(:,i));
                jH(k:k+1,:)=jh;
                k=k+2;
            end
         else
            diff=Landmark-xPred(1:2);
            d=pdist2(Landmark',xPred(1:2)');
            jH=[-diff(1)/d   -diff(2)/d      0;
                 diff(2)/d^2 -diff(1)/d^2   -1];
         end

end

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

function MapInFov = getLandmarksInsideFOV(x,Map,fov,max_range)
    cont=1;
    MapInFov=[];
    alpha = fov/2;
    min_angle=x(3)-alpha;
    max_angle=x(3)+alpha;
    nLandmark=length(Map);
    for i=1:nLandmark
        landmark=Map(:,i);
        z=getRangeAndBearing(x,landmark);
        angle=z(2)+x(3);
        dist=z(1);
        if dist<=max_range && angle<=max_angle && angle>=min_angle
            MapInFov(:,cont)=landmark;
            cont=cont+1;
        end
    end

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
```