UNIVERSIDAD DE MÁLAGA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

# Robotics

## Exercise 6.1. Mapping EKF

In this exercise we are going to **build a map** consisting of **landmarks** (or features) using an algorithm based on **EKF** and *a range-bearing* sensor, provided in the exercise's appendix.

For your convenience, it is included here the slide illustrating how the algorithm performs once the sensor takes a measurement to a landmark. Two cases are possible: the landmark is observed for first time, or the landmark was already present in the map. *Note: xEst is a vector with the coordinates of all landmarks, while pEst stores their associated uncertainty.*

New observation $[x,y]^T$

**New observation: No update step**

- State vector extended: $Xest = [-, -, \dots, -, \blacksquare]^T$
- Covariance matrix extended:

$$Pest = \begin{bmatrix} \begin{bmatrix} - & - \\ - & - \end{bmatrix} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \begin{bmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{bmatrix}_{2x2} \end{bmatrix}$$

Observation covariance projected to the global system: $J*Qest*J^T$

$$Qest = \Sigma_{r\theta} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix}$$

No correlation between the landmarks because the pose is known

**Observed landmark already in the map: do update for that landmark**

Observation Jacobian $$JH = \begin{bmatrix} 0 & 0 \cdots & \underbrace{[JHxf]_{2x2}}_{} & \cdots & 0 & 0 \\ 0 & 0 & & & 0 & 0 \end{bmatrix}$$

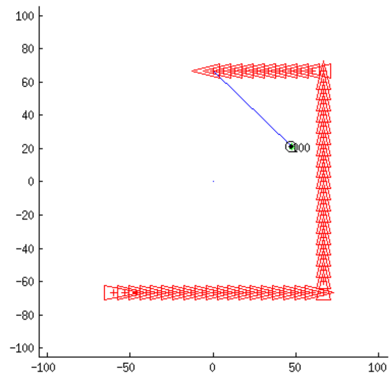
Jacobian of the landmark that has been observed, $0_{2x2}$ for the rest

In this case we make the assumption that **the robot's pose is known** (without uncertainty) and we want to estimate the *pdf* of the location of a number of landmarks that are present in the robot's surroundings, utilizing for that a **noisy sensor**.

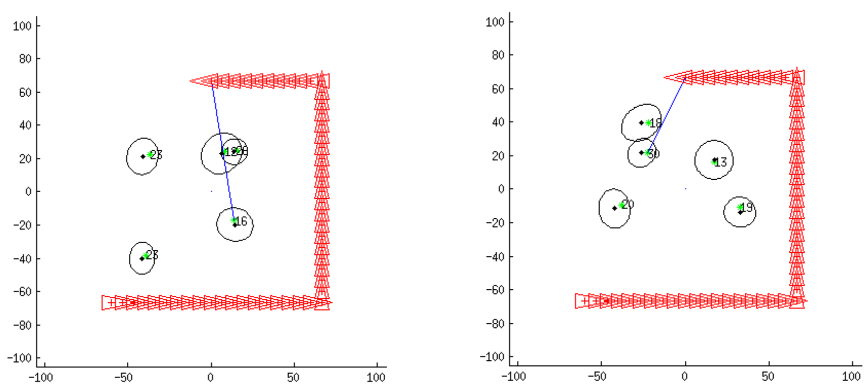
**Exercise goals:**

**1. – Complete the code.** The algorithm has gaps at some key places, so your first goal is to fill them with the appropriate code. For that, first review the code that is written and understand what is going on. Concretely, your mission is to implement the Jacobians computation, as well as some stuff related to the measurements.

**2. - Consider that only a landmark exists.** Set the variable *nFeatures* to 1. Execute the program and show the content of the vector of states *xEst* and the covariance matriz *Pest* each 5 iterations. What dimensions do they have?




**3. - Repeat the last point employing 5 *landmarks*.** Explain why and how the content of the variables *xEst* and *Pest* has change. Show also, each 5 iterations, the content of the jacobian of the observation (*jH*). What structure does the matrix of covariances have? Is there any kind of correlation among the observations of different *landmarks*?




**4. - Results of the mapping process.** Modify the code to store the determinant of the covariance matrix of each landmark and the error in the fitting along the algorithm iterations. Plot it for the case of 5 *landmarks*.

## Appendix: Exercise's code to modify

```
function EKFMappingRob
clear all;
close all;

%%global variables
global Map;global nSteps;
global Reading; global ObservedTimes;

%mode = 'step_by_step';
mode = 'visualize_process';
%mode = 'non_stop';

% Num features/landmarks considered within the map
nFeatures = 5;

% Generation of the map
MapSize = 100;
Map = MapSize*rand(2,nFeatures)-MapSize/2;

% Covariances for our very bad&expensive sensor (in the system <d,theta>)
Sigma_r = 8.0;
Sigma_theta = 7*pi/180;
Q = diag([Sigma_r,Sigma_theta]).^2;

% Initial robot pose
xVehicleTrue = [-MapSize/1.5;-MapSize/1.5;0]; % We know the exact robot pose at
any moment

%initial conditions - no map:
xEst = [];
PEst = []; %Covariance marix of the landmark position (2 rows per landmark)
QEst = 1.0*Q; %Covariance matrix of the measurement

% MappedFeatures relates the index of a feature from the true map and its
% place within the state (which depends on when it was observed).
MappedFeatures = NaN*zeros(nFeatures,1);

% storing the number of times a features has been seen
% also store the handler to the graphical info shown
ObservedTimes = zeros(nFeatures,2);

% Initial graphics - plot true map
figure(1); hold on; grid off;
plot(Map(1,:),Map(2,:),'g*');hold on;
axis([-MapSize-5 MapSize+5 -MapSize-5 MapSize+5]);
axis equal;
set(gcf,'doublebuffer','on'); %gcf: current figure handle
hObsLine = line([0,0],[0,0]);
set(hObsLine,'linestyle',':');

% Loop configuration
nSteps  = 100; % Number of motions
turning = 40; % Number of motions before turning (square path)

% Control action
u=zeros(3,1);
u(1)=(2*MapSize/1.5)/turning;
u(2)=0;

% Start the loop!
for k = 1:nSteps

    %
    % Move the robot
    %
```

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

xVehicleTrue = tcomp(xVehicleTrue,u); % Perfectly known robot pose

% We assume that the map is static (the state transition model of
xPred = xEst;
PPred = PEst;

%
% Observe a randomn feature
%

[z,iFeature] = getRandomObservationFromPose(xVehicleTrue,Map,Q);

% Update the "observedtimes" for the feature and plot the reading
ObservedTimes(iFeature)=ObservedTimes(iFeature)+1;
PlotNumberOfReadings(xVehicleTrue,iFeature,Map);

 % Have we seen this feature before?
 if( ~isnan(MappedFeatures(iFeature)) ) %Yes, it is already in the map

        %
        % Predict observation
        %

        % Find out where it is in state vector
        FeatureIndex = MappedFeatures(iFeature);

        % xFeature is the current estimation of the position of the
        % landmard "FeatureIndex"
        xFeature = xPred(FeatureIndex:FeatureIndex+1);

        % Predicts the observation
        zPred = 000000000; % Hint: use getRangeAndBearing function

        % Get observation Jacobians
        jHxf = GetObsJacs(xVehicleTrue,xFeature);

        % Fill in state jacobian
        % (the jacobian is zero except for the observed landmark)
        jH = 000000000;

        %
        % Kalman update
        %

        Innov = z-zPred; % Innovation
        Innov(2) = AngleWrap(Innov(2));

        S = jH*PPred*jH'+QEst;
        K = PPred*jH'*inv(S); % Gain
        xEst = xPred+ K*Innov;
        PEst = PPred-K*S*K';

        %ensure P remains symmetric
        PEst = 0.5*(PEst+PEst');

 else % No in the current map (state)

        % This is a new feature, so add it to the map
        nStates = length(xEst); %dimension 2x#landmarks_in_map

        % The observation is in the local frame of the robot, it has to
        % be translated to the global frame
        xFeature = 000000000;
```

```
            % Add it to the current state
            xEst = [xEst;xFeature]; %Each new feature two new rows

            % Compute the jacobian
            jGz = GetNewFeatureJacs(xVehicleTrue,z); %Dimension 2x2

            M = [eye(nStates), zeros(nStates,2);% note we don't use jacobian
w.r.t vehicle since the pose doesn’t have uncertainty
                zeros(2,nStates)  , jGz];

            PEst = M*blkdiag(PEst,QEst)*M';
            %THis can also be done directly PEst = [PEst,zeros(nStates,2);
            %                                        zeros(2,nStates),
jGz*QEst*jGz']


            %remember this feature as being mapped: we store its ID for the state
vector
            MappedFeatures(iFeature) = length(xEst)-1;  %Always an odd number

        end;

    % Drawings

    pause(0.005);

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

        %xEst
        %PEst

        DrawRobot(xVehicleTrue,'r');%plot(xVehicleTrue(1),xVehicleTrue(2),'r*');
        DoMapGraphics(xEst,PEst,5); % Draw estimated poitns (in black) and
ellipses
        axis([-MapSize-5 MapSize+5 -MapSize-5 MapSize+5]); % Set limits again
        drawnow;
        if strcmp(mode,'step_by_step')
            pause;
        elseif strcmp(mode,'visualize_process')
            pause(0.2);
        elseif strcmp(mode,'non_stop')
            % non stop!
        end
    end;
end;
end

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

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

    iFeatures = size(Map,2);
    iFeature = randi(iFeatures);
    feature = Map(:,iFeature);

    z = getRangeAndBearing(xVehicleTrue,feature,Q);

end

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

function z = getRangeAndBearing(xVehicleTrue,feature,Q)

    Delta_x = feature(1,:) - xVehicleTrue(1);
    Delta_y = feature(2,:) - xVehicleTrue(2);
```

```
    z(1,:) = sqrt(Delta_x.^2 + Delta_y.^2);   % Range
    z(2,:) = atan2(Delta_y,Delta_x) - xVehicleTrue(3);  % Bearing
    z(2,:) = AngleWrap(z(2,:));

    if nargin == 3
        z = z + sqrt(Q)*rand(2,1); % Adding noise
    end

end

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

function [jHxf] = GetObsJacs(xPred, xFeature)

    000000000;

end

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

function [jGz] = GetNewFeatureJacs(Xv, z)

    000000000;

end

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

function PlotNumberOfReadings(xVehicleTrue,iFeature,Map)

    global Reading;
    global ObservedTimes;

    for c=1:length(Reading)
       if (~isnan(Reading(c)))
           delete(Reading(c));
       end;
    end

    Reading=zeros(length(iFeature));

    if (ObservedTimes(iFeature,2)~=0) delete(ObservedTimes(iFeature,2));end;
    ObservedTimes(iFeature,2)=text(Map(1,iFeature)+rand(), ...
        Map(2,iFeature)+rand(),sprintf('%d',ObservedTimes(iFeature,1)));

    for c=1:length(iFeature)
       if (iFeature(c)~=-1)
        Reading(c)=line([xVehicleTrue(1), Map(1,iFeature(c))], ...
                        [xVehicleTrue(2), Map(2,iFeature(c))]);
       else
           Reading(c)=NaN;
       end
    end
end

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

function DoMapGraphics(xMap,PMap,nSigma)

    persistent k;
    persistent handler_ellipse; %%cga animating ellipses
    persistent handler_state; %%cga animating ellipses

    if(isempty(k))
        k = 0;
    end;
    k = k+1;
```

```
    % removing ellipses from the previous iteration
    if isempty(handler_ellipse)
        handler_ellipse=zeros(length(xMap));
    else
        for i=1:length(handler_ellipse)
            if (handler_ellipse(i)~=0)
                delete (handler_ellipse(i));
            end
        end
    end

    % removing state from the previous iteration
    if (isempty(handler_state))
        handler_state=zeros(length(xMap));
    else
        for i=1:length(handler_state)
            if (handler_state(i)~=0)
                delete (handler_state(i));
            end
        end
    end

    handler_ellipse=zeros(length(xMap));
    handler_state=zeros(length(xMap));

    colors = 'kkkk';

    for i = 1:length(xMap)/2
        iL = 2*i-1; iH = 2*i;
        x = xMap(iL:iH);
        P = PMap(iL:iH,iL:iH);
        handler_ellipse(i)= PlotEllipse(x,P,nSigma,'k');
        handler_state(i)= plot(x(1),x(2),'k.');
        c = colors(mod(i,4)+1);
        set(handler_ellipse(i),'color',char(c));
       % plot3(x(1),x(2),k,'r+');
    end
end

%-----------------------------------------------------------------------%
```