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`)

PPred is the matrix of the prediction of the covariance of the robot and the observed landmarks and the interaction of each other. It is made with the following matrices:

- PPredvv, which is the predicted covariance of the robot and its dimention is 3x3.
- PPredvm, which is the predicted covariance between the robot and each landmark and its dimension is 3x2l which l is the number of observed landmarks.
- PPredmm, which is the predicted covariance of each landmark and its dimension is 2lx2l, with l being the number of observed landmarks.

Thus, PPred has the dimension of (3+2l)x(3+2l).

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.

```
[jHxv,jHxf] = GetObsJacs(xVehicle,xFeature)
jH=zeros(2,nStates-1+3);
jH(:,1:3)=jHxv;
jH(:,FeatureIndex:FeatureIndex+1)=jHxf;
```

The output of the GetObsJacs gives two matrices with dimension of 2x3 and 2x2 respectively. The first matrix (jHxv) is the Jacobian for the robot and the second matrix (jHxf) is the Jacobian for the landmark that the robot is looking at.

jH has the dimention of 2x(3+2l), being l the number of observed landmarks. This is the Jacobian which have the first part jHxv and the rest is at 0, except in the index where is located the landmarks which the robot is looking at, with jHxf.

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`.

```
e = xRobotTrue-xEst
XErrStore(:,k)=[e(1:2)'*e(1:2);
               e(3)^2];
PXErrStore(k)=det(PEst(1:3,1:3));
if(length(xEst)>3)
    FeatErrStore(:,k)=ErrorsLandmarks(xEst,MappedFeatures,Map);
    PFeatDetStore(:,k)=DeterminantsLandmarks(PEst,MappedFeatures);
end

function error = ErrorsLandmarks(EstLand,LandObs,Map)
%ERRORS
%   EstLand = Estimated position of each landmark and the robot
%   LandObs = Vector of landmarks observed
    tam = size(LandObs,1);
    error=NaN*ones(tam,1);
    for i=1:tam
        pos=LandObs(i);
        if(~isnan(pos))
            land=Map(:,i);
            est=EstLand(pos:pos+1);
            e=land-est;
            error(i)=e'*e;
        end
    end
end

function d = DeterminantsLandmarks(Covar,LandObs)
%   Covar = Covariances of each landmark and the robot
%   LandObs = Vector of landmarks observed
    tam = size(LandObs,1);
    d=NaN*ones(tam,1);
    for i=1:tam
        pos=LandObs(i);
        if(~isnan(pos))
            cov = Covar(pos:pos+1,pos:pos+1);
            d(i) = det(cov);
        end
    end
end
```

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.

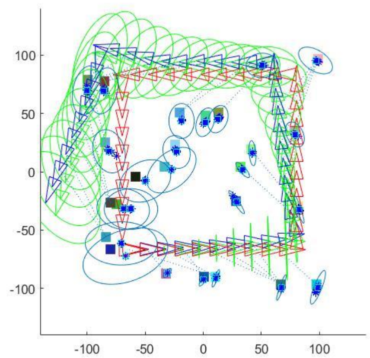
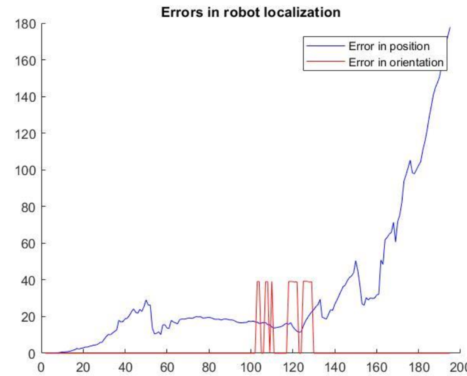
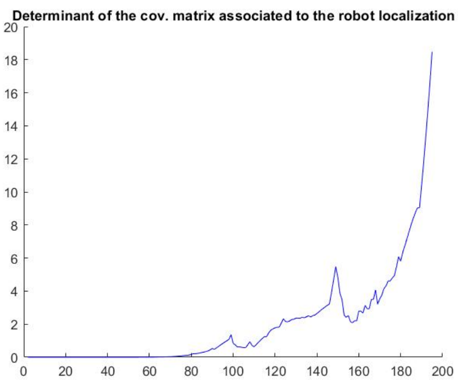


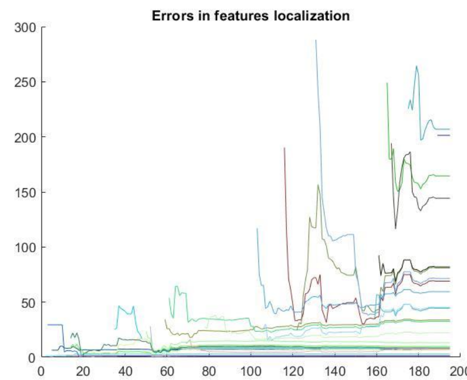
Image 1. Execution of SLAM for one random landmark



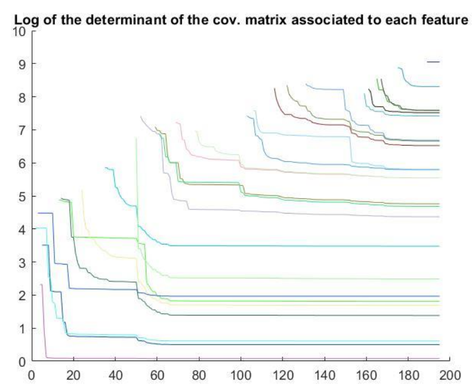
Plot 1. Error in the robot localization and orientation



Plot 2. Covariance of the robot



Plot 3. Error in the localization of each landmark



Plot 4. Covariance of each landmark

In Plot 1, we see the error in the localization and orientation of the robot. It increases when the sensor does not capture any landmark, or it captures a landmark and is the first time when it captures. And it decreases when the sensor captures a landmark which has been already observed. Furthermore, the orientation only depends on how well is calculated where is the robot think it is.
In Plot 2, we see the value of the determinant of the covariance of the robot. Mostly, it has the same behavior as the error.

In Plot 3, we see the error in the localizations of the landmarks. They start with high values, and later they decrease until a point where oscillate trying to put the estimated landmark as close as the real one is.
In Plot 4, it shows the determinants of the covariance matrices of the landmarks, which all of them decreases when the sensor captures them.

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

- For 1 landmark:

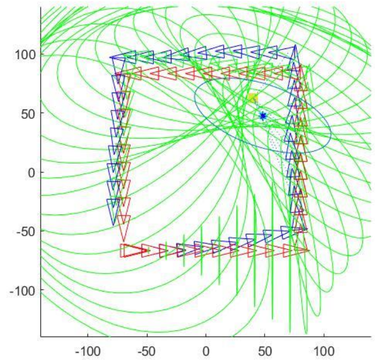
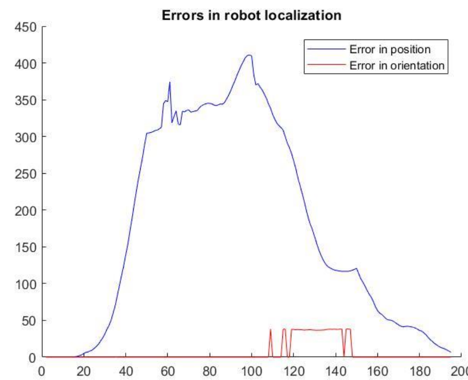
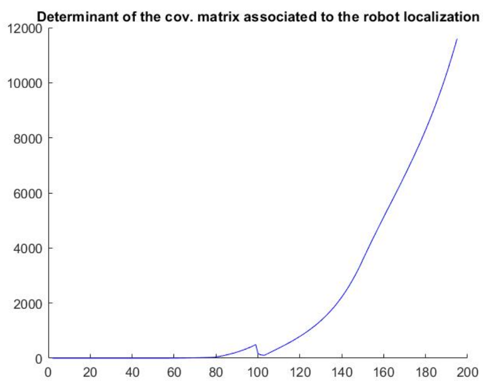


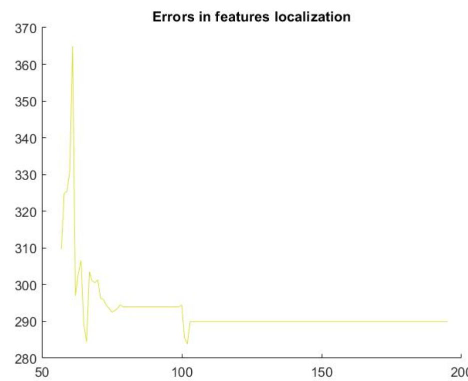
Image 2. Execution of SLAM



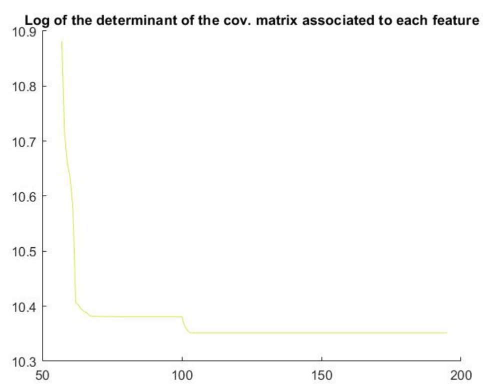
Plot 5. Error in the robot localization and orientation



Plot 6. Covariance of the robot



Plot 7. Error in the localization of the landmark



Plot 8. Covariance of the landmark

As we see in the plots, for a single landmark it is more difficult to localize the robot and the landmark. The determinant of the covariance matrix of the robot increases until it found the landmark, from there decrease until it does not capture the landmark. However, the robot does not know the existence of the landmark until the sensor capture it, from there the algorithms try to localize the landmark. The error in the localization fluctuate until the sensor does not capture the landmark, and the determinant decrease as always.

- For 10 landmarks:

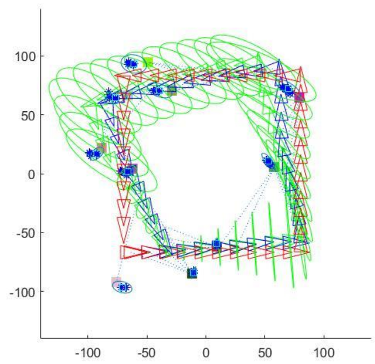
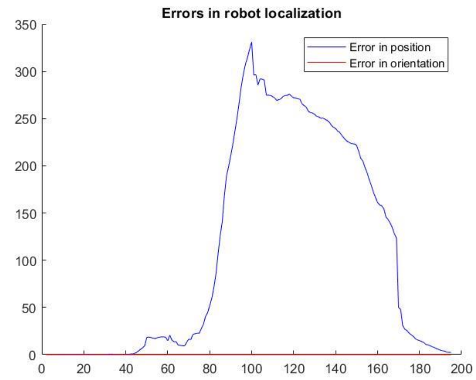
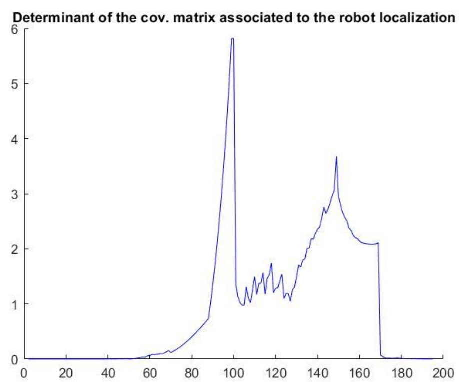


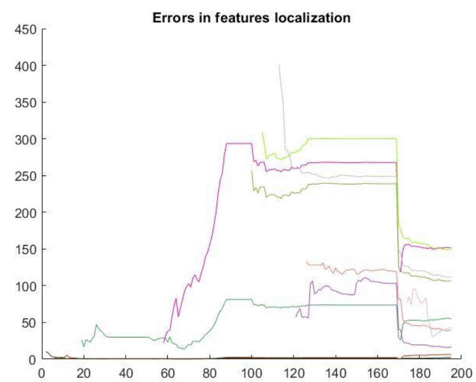
Image 3. Execution of SLAM



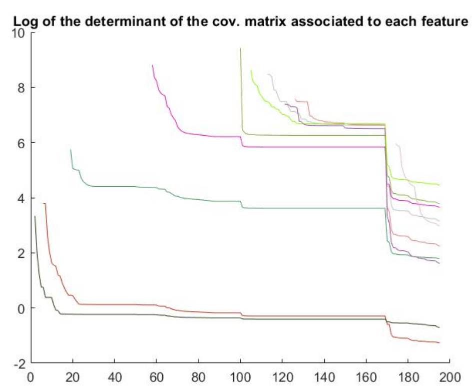
Plot 9. Error in the robot localization and orientation



Plot 10. Covariance of the robot



Plot 11. Error in the localization of the landmark



Plot 12. Covariance of each landmark

In this case, the robot can be localized better than the before one. Also, the determinant is little until the robot do not observe any landmark. For the landmark ones, it has the same results as the other cases. Furthermore, all the values are better than the last case, but worse then the original one (3.1).

- For 50 landmarks:

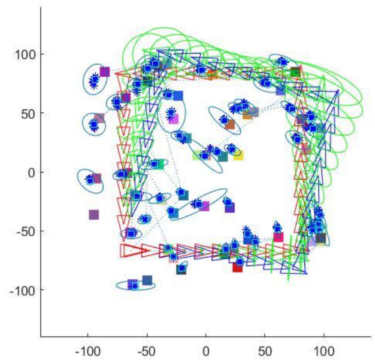
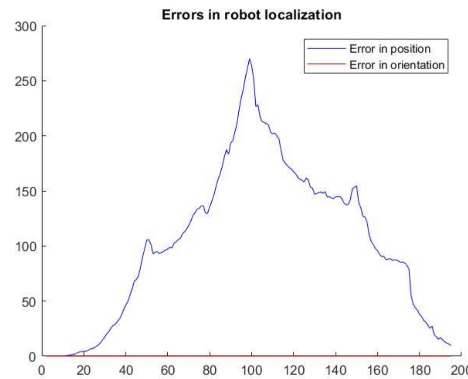
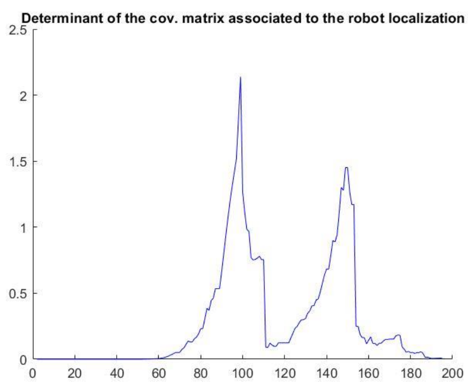


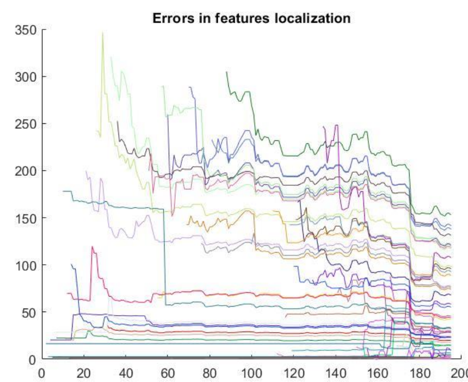
Image 4. Execution of SLAM



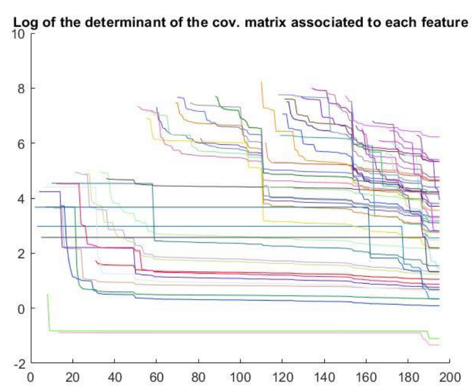
Plot 13. Error in the robot localization and orientation



Plot 14. Covariance of the robot



Plot 15. Error in the localization of the landmark



Plot 16. Covariance of each landmark

In this case, all have the same behavior, but it is even more precise where the robot is, and its determinant is much litter then the other cases. For the landmarks, they also have a better result than any other case.

And thus, comparing all the cases, if we have more landmarks, we have better values. However, it is more time and space consuming.

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.

[…]

```
elseif strcmp(mode,'landmarks_in_fov')
    [MapInFov,iFeatures]=getObservationsInsideFOV(xRobotTrue,Map,fov,max_range);
        if ~isempty(MapInFov)
            for i=1:size(MapInFov,2)
                landmark = MapInFov(:,i);
                z(:,i) = getRangeAndBearing(xRobotTrue,landmark,Q);
            end
        else
            z=[];
        end
    end
if(~isempty(z))
        if strcmp(mode,'landmarks_in_fov')
            tam = size(MapInFov,2);
        else
            tam=1;
        end
        for i=1:tam
            if strcmp(mode,'landmarks_in_fov')
                iFeature=iFeatures(i);
            end
            if( ~isnan(MappedFeatures(iFeature,1)))

               […]

                if i~=1
                    PPred=PEst;
                    xPred = xEst;
                end

                […]
            else
                […]
                if i~=1
                    xPred = xEst;
                end
                […]

            end;
        end
    else
        xEst = xPred;
        PEst = PPred;
    end;
[…]
if(~isnan(z))
for i=1:tam
if strcmp(mode,'landmarks_in_fov')
   iFeature=iFeatures(i);
end
hLine=line([xRobotTrue(1),Map(1,iFeature)],[xRobotTrue(2),Map(2,iFeature)]);
set(hLine,'linestyle',':');
end;
end
```

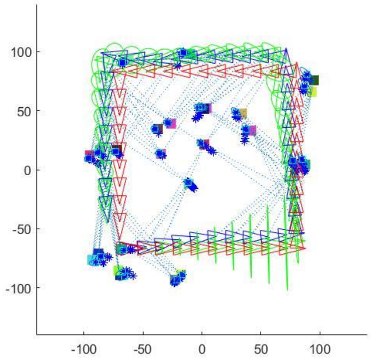
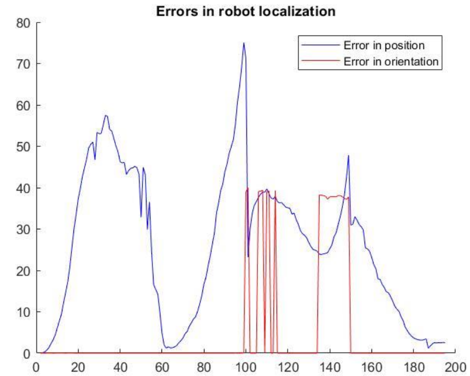
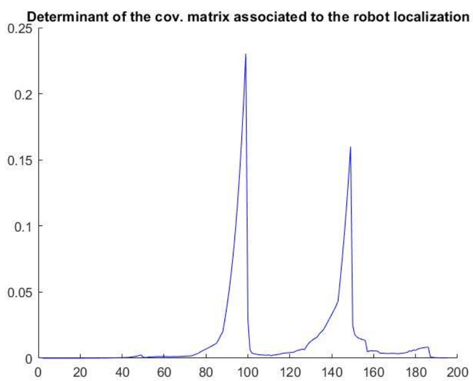


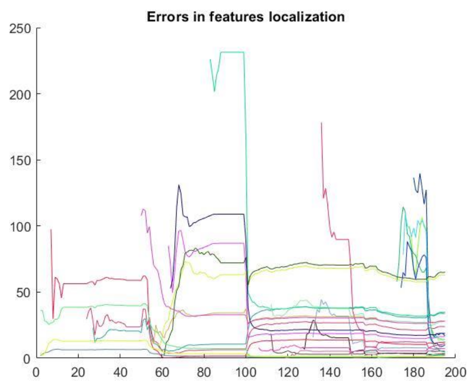
Image 5. Execution of SLAM for all captured landmark



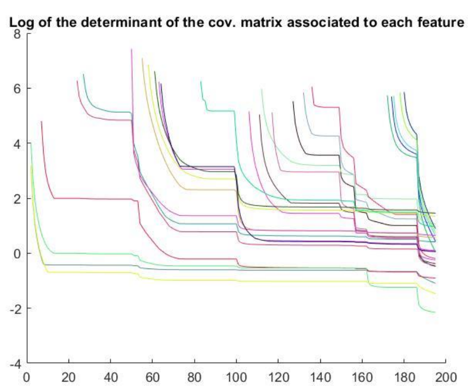
Plot 17. Error in the robot localization and orientation



Plot 18. Covariance of the robot



Plot 19. Error in the localization of the landmark



Plot 20. Covariance of each landmark

Comparing with the other method, this has better values in all their plots. For instance, the determinant for the covariance matrix of the robot is little that it is almost 0, meaning that where can be the robot is nearly s precise and perfect. Furthermore, the error in the localization of the landmarks mostly all decrease and have acceptable error in the end.

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
        %
        [MapInFov,iFeatures] = getObservationsInsideFOV(xRobotTrue,Map,fov,max_range);
        if ~isempty(MapInFov)
            for i=1:size(MapInFov,2)
                landmark = MapInFov(:,i);
                z(:,i) = getRangeAndBearing(xRobotTrue,landmark,Q);
            end
        else
            z=[];
        end
       end

        %
        % Update step
        %

        if(~isempty(z))
            %have we seen this feature before?
          if strcmp(mode,'landmarks_in_fov')
            tam = size(MapInFov,2);
          else
            tam=1;
          end
          for i=1:tam
            if strcmp(mode,'landmarks_in_fov')
                iFeature=iFeatures(i);
```

```
            end

            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=zeros(2,nStates-1+3);
                jH(:,1:3)=jHxv;
                jH(:,FeatureIndex:FeatureIndex+1)=jHxf;


                % Do Kalman update:
                Innov = z-zPred;
                Innov(2) = AngleWrap(Innov(2));

                if i~=1
                    PPred=PEst;
                    xPred = xEst;
                end


                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))];
                if i~=1
                    xPred = xEst;
                end

                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;
             end
        else
            xEst = xPred;
            PEst = PPred;
        end;
```

```
    %
    % Point 3, Robot pose and features localization errors and determinants
    %
    e = xRobotTrue-xEst
    XErrStore(:,k)=[e(1:2)'*e(1:2);
                    e(3)^2];
    PXErrStore(k)=det(PEst(1:3,1:3));
    if(length(xEst)>3)
        FeatErrStore(:,k)=ErrorsLandmarks(xEst,MappedFeatures,Map);
        PFeatDetStore(:,k)=DeterminantsLandmarks(PEst,MappedFeatures)
    end
        % 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))
            for i=1:tam
                if strcmp(mode,'landmarks_in_fov')
                    iFeature=iFeatures(i);
                end
                hLine =
line([xRobotTrue(1),Map(1,iFeature)],[xRobotTrue(2),Map(2,iFeature)]);
                set(hLine,'linestyle',':');
            end;
        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,:));

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

function error = ErrorsLandmarks(EstLand,LandObs,Map)
%ERRORS
%    EstLand = Estimated position of each landmark and the robot
%    LandObs = Vector of landmarks observed
    tam = size(LandObs,1);
    error=NaN*ones(tam,1);
    for i=1:tam
```

```
        pos=LandObs(i);
        if(~isnan(pos))
            land=Map(:,i);
            est=EstLand(pos:pos+1);
            e=land-est;
            error(i)=e'*e;
        end
    end
end

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

function d = DeterminantsLandmarks(Covar,LandObs)
%   Covar = Covariances of each landmark and the robot
%   LandObs = Vector of landmarks observed
    tam = size(LandObs,1);
    d=NaN*ones(tam,1);
    for i=1:tam
        pos=LandObs(i);
        if(~isnan(pos))
            cov = Covar(pos:pos+1,pos:pos+1);
            d(i) = det(cov);
        end
    end
end
```