23TTP409 Autonomous Vehicles
Coursework 2 - Extended Kalman Filtering (EKF) for Landmark-based Localisation

Gabriel Wendel

Second assignment in module 23TTP409 - Autonomous Vehicles

Aeronautical and Automotive Engineering
Erasmus Exchange Semester at Loughborough University
United Kingdom

Contents

Problem Formulation

The task at hand involves localizing a vehicle equipped with a vision sensor within a given environment containing known landmarks. The vehicle initiates its trajectory from the coordinate (-35, 0) and travels the scene at varying velocities. The vision sensor measures relative bearing angles between the vehicle and the landmarks, with each landmark's location being known. These sensor measurements are associated with their corresponding landmarks. The task is to develop and execute an Extended Kalman Filter algorithm capable of estimating the vehicle's motion parameters, specifically its position and velocity, utilizing the provided sensor data.

1 EKF - Algorithm Derivation

The Extended Kalman Filter is an extension of the Kalman Filter, designed to handle nonlinear systems. It operates on the principles of Bayesian estimation, which involves recursively updating the belief about the state of a dynamic system based on noisy measurements [1]. In this case, the measurements were bearing angle readings for three landmarks with known positions. The vehicle's motion is described by the nonlinear unicycle model:

$$\begin{cases} x_{k+1} &= x_k + v\cos\varphi T_s \\ y_{k+1} &= y_k + v\sin\varphi T_s \\ \varphi_{k+1} &= \varphi_k + \omega_k T_s \end{cases} \tag{1}$$

where T_s is the time step, v is vehicle velocity, φ is the heading angle and ω is the yaw rate. The motion model can be written in matrix form as:

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k \tag{2}$$

where $\mathbf{F}_k$ is the state transition matrix, which defines how the state changes from one time step to the next under the influence of the system dynamics. In this case $\mathbf{F}_k$ is defined as:

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & -v\sin\varphi T_s \\ 0 & 1 & v\cos\varphi T_s \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

As previously mentioned measurements are the bearing angles from the vehicle to each of the landmarks. The measurement model relating state vector to the measurements from the i-th landmark is:

$$z_{k,i} = \tan^{-1}\left(\frac{y_{\text{landmark},i} - y_k}{x_{\text{landmark},i} - x_k}\right) + v_k \tag{4}$$

where v_k is the measurement noise which is assumed to be Gaussian. The basic principle of the EKF-Algorithm can be summarized in three steps; Prediction step, Update step and Iteration. First the process noise covariance matrix, $\mathbf{Q}$, and the measurement noise covariance matrix, $\mathbf{R}$ are initialized. The process noise covariance matrix, $\mathbf{Q}$, represents the covariance of the process noise which accounts for the uncertainties in the system model.

$$\mathbf{Q} = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$$

where σ is the odometry covariance associated with the uncertainties in the respective state variables.

$\mathbf{R}$ is initialized based on the characteristics of the sensors or measurement processes involved. It influences how much trust is placed in the new measurements relative to the predicted state:

$$\mathbf{R} = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix} \tag{5}$$

where σ is the variances of the measurement noise for each sensor. The state prediction is the process of estimating the state of the system at the next time step based on the current state estimate and the system's dynamics, described by the vehicle model, Equation 1. The prediction step can be described as:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_{k-1}\hat{\mathbf{x}}_{k-1} \tag{6}$$

Here, $\hat{\mathbf{x}}_{k|k-1}$ is the predicted state vector at time k given all information up to time $k-1$. The update step corrects the predicted state by incorporating new measurement data. This process adjusts the predicted state and its covariance matrix to reflect the new evidence provided by the measurements. First, we define the measurement model, which relates the state vector to the measurements. For a system where measurements are nonlinear functions of the state, the measurement model is represented by:

$$\mathbf{z}_k = h(\hat{\mathbf{x}}_{k|k-1}) + \mathbf{v}_k \tag{7}$$

where $\mathbf{z}_k$ is the vector of measurements at time k. $h(\cdot)$ is the nonlinear measurement function. $\mathbf{v}_k$ is the measurement noise, which is assumed to be Gaussian with a covariance matrix $\mathbf{R}_k$. The uncertainty in the state prediction is updated using the covariance matrix as follows:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \tag{8}$$

where $\mathbf{P}_{k|k-1}$ is the predicted covariance matrix at time k, indicating the expected uncertainty in the state estimate before considering the new measurements at time k. $\mathbf{P}_{k-1}$ is the updated covariance matrix from the last time step. $\mathbf{Q}_{k-1}$ is the process noise covariance matrix for time $k-1$, introducing additional uncertainty to accommodate for model inaccuracy.

To linearize the nonlinear measurement function at the predicted state, we compute the Jacobian matrix of $h(\cdot)$ evaluated at the predicted state. In our case we have three bearing angle measurements (bearing observation model). Thus the $\mathbf{H}$ matrix is defined for each of the bearing measurements:

$$\mathbf{H}_k = \left.\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}} = \begin{bmatrix} -\dfrac{x_2}{x_1^2+x_2^2} & \dfrac{x_1}{x_1^2+x_2^2} & 0 \end{bmatrix} \tag{9}$$

where x_1 and x_2 is the distance between landmark and vehicle in the x-direction and y-direction respectively. The distance between the vehicle and a given landmark can be calculated according to Figure 1

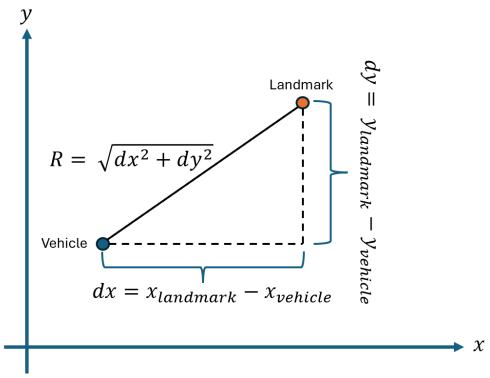


Figure 1: Distance, R, between vehicle and landmark.

The Kalman Gain, which determines the influence of the measurement residual on the state update, is then calculated:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1} \tag{10}$$

Using the Kalman Gain, the estimated state is then updated by including the measurement residual:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})) \tag{11}$$

where $\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})$ is the measurement residual, representing the difference between the actual measurement and what the model predicted. Finally, the covariance matrix of the state estimate is updated to reflect the reduction in uncertainty resulting from the incorporation of the new measurement [2]:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{12}$$

where $\mathbf{I}$ is the identity matrix with the dimensions of $\mathbf{P}_{k|k-1}$. The update reduces the uncertainty in the state estimate. These steps are then iterated for each time step in the simulation and we obtain an estimate of the vehicle states. Below is a pseudo code that describes how the algorithm was implemented in code:

Algorithm 1 Extended Kalman Filter

Initialize: $\hat{\mathbf{x}}_0$, $\mathbf{P}_0$
for $k = 1 : Ts$ **do**
  Predict
    Predict the state estimate:
      $\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1})$
    Calculate the Jacobian of the state transition:
      $\mathbf{F}_k = \frac{\partial f}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}}$
    Predict the error covariance:
      $\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_{k-1}$
  Update
    Get measurement $\mathbf{z}_k$
    Calculate the measurement residual:
      $\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})$
    Calculate the Jacobian of the measurement model:
      $\mathbf{H}_k = \frac{\partial h}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{k|k-1}}$
    Calculate the Kalman gain:
      $\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}$
    Update the state estimate:
      $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
    Update the error covariance:
      $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}$
end for

2 Algorithm Performance Evaluation

The EKF algorithm estimated projecty can be seen in Figre 2 below.

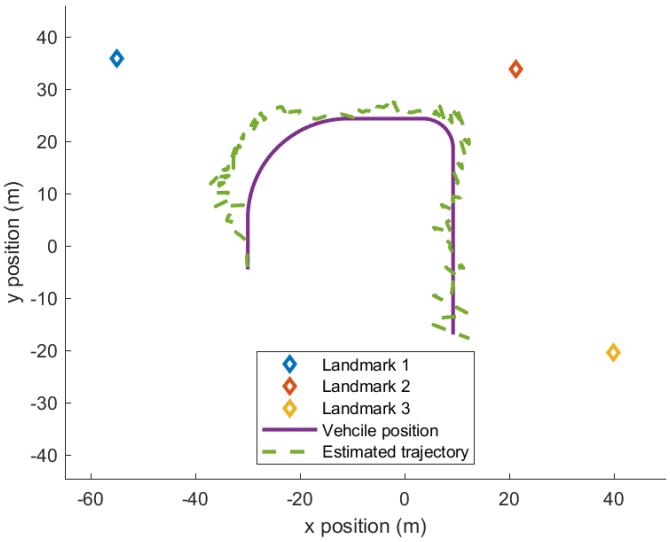
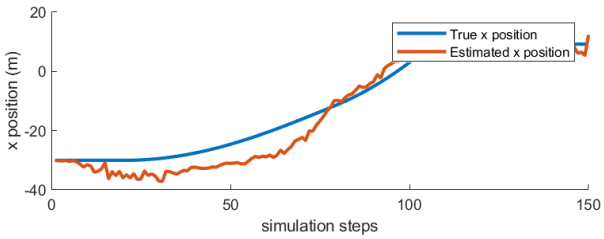
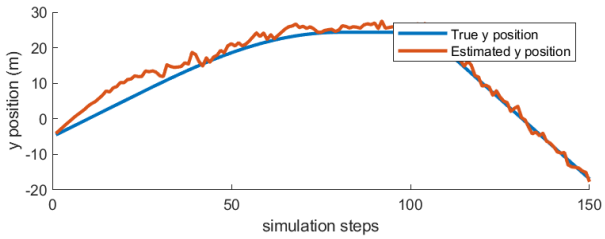


Figure 2: Estimated trajectory using EKF

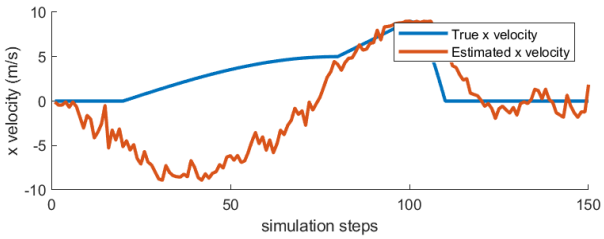
Estimated x- and y- positions as well as x- and y-velocities are illustrated in Figure 3.



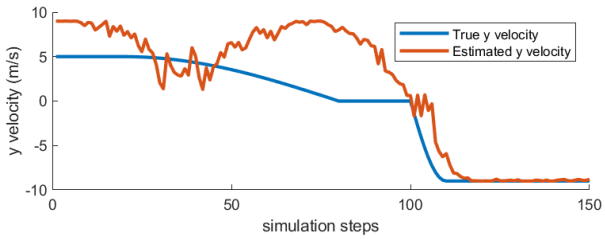
(a) Estimated x-position.



(b) Estimated y-position.



(c) Estimated x-velocity.



(d) Estimated y-velocity.

Figure 3: Estimated x- and y-position and x- and y-velocities.

As can be seen in Figure 3 the algorithm is able to estimate the position quite well with little deviation from the true position. However, its performance in velocity prediction is not as satisfactory. This discrepancy may stem from the simplicity of the model. For instance, the assumption of a constant yaw rate, ω, without recalibration in each iteration could result in incorrect vehicle directions. Implementing a more complex model might improve results.

Figure 2 further demonstrates the deviation from the actual vehicle trajectory. Additionally, it reveals oscillations that could be attributed to measurement noise. Adjusting the covariance $\mathbf{Q}$ for measurement noise, as discussed in Section 4, might mitigate these noise-induced oscillations.

3 Algorithm Evaluation with Additional Dataset

Figure illustrates the estimated trajectory for the new dataset.

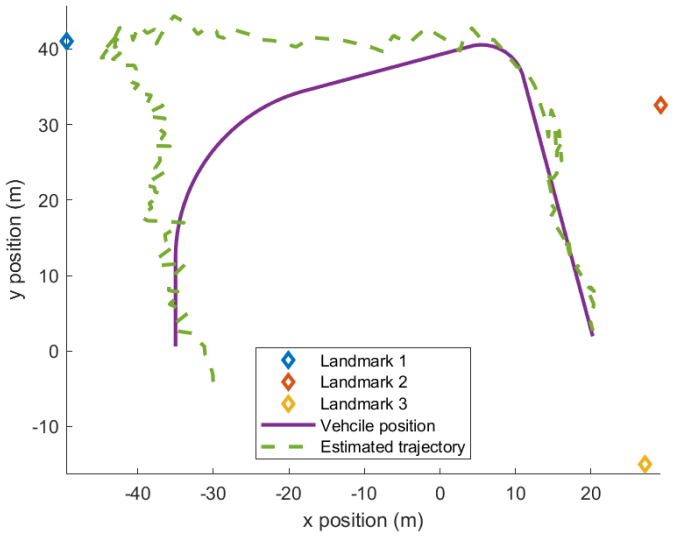
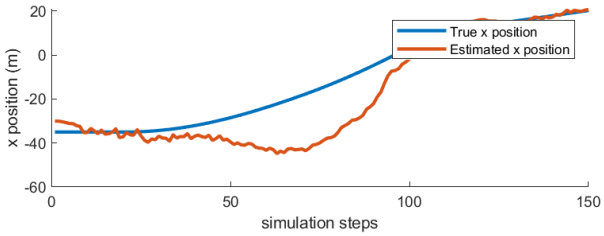
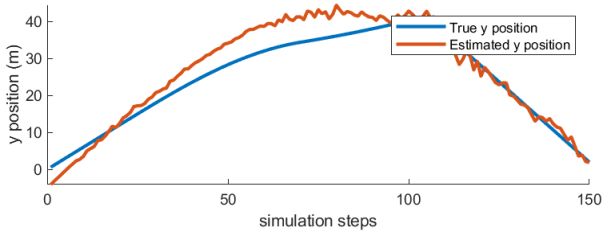


Figure 4: Estimated trajectory for the new dataset.

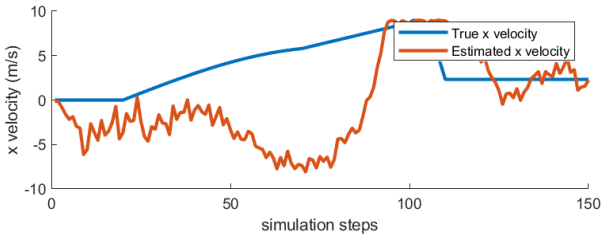
Estimated x- and y- positions as well as x- and y-velocities for the additional dataset are illustrated in Figure 5.



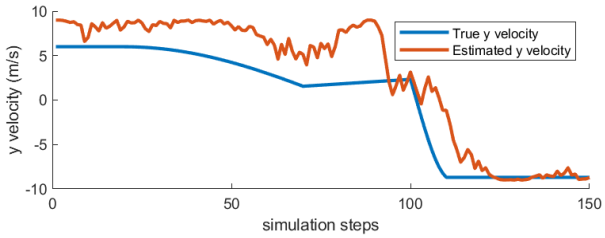
(a) Estimated x-position.



(b) Estimated y-position.



(c) Estimated x-velocity.



(d) Estimated y-velocity.

Figure 5: Estimated x- and y-position and x- and y-velocities.

The figure in Figure 5 reveals that the algorithm's estimation of x- and y-positions isn't as accurate as it was with the original dataset. Several factors could contribute to this discrepancy.

One crucial consideration is the starting position. In the new dataset, the vehicle commences from a different starting point (see deviation in starting position in Figure 5a and Figure 5b), thus resulting in inaccurate initial x- and y-position.

Additionally, the noise characteristics in the new dataset might differ from those in the original dataset. Refining the measurement covariance $\mathbf{R}$ could potentially address this issue.

Furthermore the inaccurate estimation of velocities remain. Again this might be due to the model being to simple to capture the true movement of the vehicle.

4 Advantages, Disadvantages and Tuning Parameters

The Extended Kalman Filter is a powerful tool for estimating the state of nonlinear dynamic systems, compared to traditional Kalman Filter which only estimates non-linear systems. However, EKF has its limitations. One significant limitation is the need to linearize the process and measurement models using Taylor expansion, which can introduce errors, especially if the system dynamics are highly nonlinear [3].

In the Extended Kalman Filter (EKF) algorithm, the parameters representing process noise covariance Q and measurement noise covariance R are pivotal in shaping the filter's performance.

As previously mentioned the process noise covariance, $\mathbf{Q}$ characterizes the uncertainty in the system's state transition model, capturing random disturbances impacting the state evolution over time. The measurement noise covariance, $\mathbf{R}$, depicts uncertainty within the observation model, representing random errors affecting the system state measurements. Different values for noise- and odometry covariances was tested iteratively.

The initial noise covariance matrix can be seen in Equation 5 with $\sigma = 0.045$. This generated wery "spiky" estimations, see Figure 6. Increasing $\mathbf{R}$ by a factor of 50 resulted in much smoother estimations, see final estimation graphs in Section 2. The odometry covariance was maintained at its original value since altering it did not yield improved results.

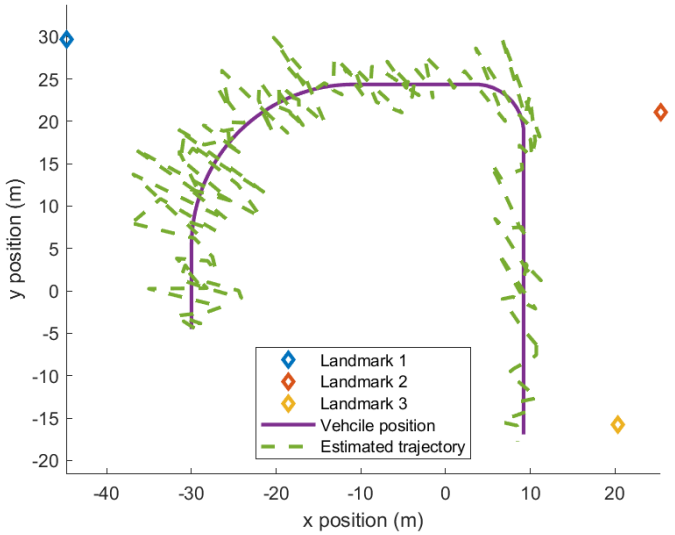


Figure 6: Estimated trajectory with initial measurement covariance.

References

[1] Daum, F. (2014). *Extended Kalman filters*. In Springer eBooks (pp. 1–3). https://doi.org/10.1007/978-1-4471-5102-9_62-2

[2] Andrade-Cetto, J., Sanfeliu, A. A *The Kalman Filter*. In: Environment Learning for Indoor Mobile Robots. Springer Tracts in Advanced Robotics, vol 23. Springer, Berlin, Heidelberg. https://doi-org.proxy.lib.chalmers.se/10.1007/11418382_6

[3] Kris Kitani. *Extended Kalman Filter*. Carnegie Mellon University, Pittsburgh, Pennsylvania. Accessed: 07-02-2024. [Online]. Available https://www.cs.cmu.edu/~16385/s17/Slides/16.4_Extended_Kalman_Filter.pdf

Appendix

Listing 1: MATLAB code EKF Algorithm

```
1
2  % 23TTP409 coursework 2: EKF localisation
3  % Gabriel Wendel
4
5  clear all
6  close all
7  clc
8
9  %-------------------------------------------------------------------------
10 % simulation settings
11
12 timeStep = 150; % total simulation steps
13 Ts=0.1; % sampling interval
14
15
16 sig_bearing = 0.045; % standard deviation of bearing noise (rad) orignially
       0.045
17 sig_odometry = 0.7; % standard deviation of odometry noise (m/s) originally 0.7
18 %-------------------------------------------------------------------------
19
20 % Landmark locations
21
22 nLandmark = 3; % three landmarks are considered
23
24 % define the default landmark map based on random points
25 % inital map area: -50m to 50m on both x and y directions
26 landmarkMap = 100*rand(2,nLandmark)-50;
27
28 % re-define three landmarks in designated positions
29 landmarkMap(:,1) = [20*rand-60; 20*rand+20];
30 landmarkMap(:,2) = [20*rand+10; 20*rand+15];
31 landmarkMap(:,3) = [20*rand+20; 20*rand-25];
32
33 % plot the landmarks
34 figure(1)
35 hold on
36 axis equal
37 xlabel('x position (m)')
38 ylabel('y position (m)')
39
40 for kk = 1:nLandmark
41     plot(landmarkMap(1,kk),landmarkMap(2,kk),'d','LineWidth',2);
42     str = sprintf('The position of landmark No.%d is (%0.5g, %0.5g)', kk,
          landmarkMap(1,kk), landmarkMap(2,kk));
43     disp(str);
44 end
45 %-------------------------------------------------------------------------
46
47 % Generate vehicle trajecotry
48
49 % define the array that contains the vehicle trajectory
50
51 % generate vehicle trajecotry using the unicycle vehicle model
```

```
xpos = -30; % x position
ypos = -5; % y position
xvel = 0;
yvel = 5;
vel = 5; % vehicle speed
psi = atan2(yvel,xvel); % vehicle heading

xTrue = [];
for kk = 1:timeStep
    % define heading angle at different time steps
    if kk<=20
        omg = 0;
    elseif kk<=80
        omg = -pi/12;
    elseif kk<=100
        omg = 0;
        vel = vel + 0.2;
    elseif kk<=110
        omg = -pi/2;
    else
        omg = 0;
    end

    xpos = xpos + vel*cos(psi)*Ts;
    ypos = ypos + vel*sin(psi)*Ts;
    psi = psi + omg*Ts;

    newState = [xpos ypos vel*cos(psi) vel*sin(psi) vel omg]';

    xTrue = [xTrue newState]; % vehicle state trajectory
end

plot(xTrue(1,:),xTrue(2,:),'LineWidth',2);
legend('Landmark 1','Landmark 2','Landmark 3','Vehcile position', 'Location','
    south');
%--------------------------------------------------------------------------

% Generate sensor readings

% define the 2-D array that stores the bearing readings w.r.t all landmarks
% and the velocity readings by the odometry
% at each time step zObv = [bearing1; bearing2; bearing3];

zObv = zeros(nLandmark,size(xTrue,2)); % bearing

% generate sensor readings for each landmark
for kk = 1:nLandmark
    xLandmark = landmarkMap(1,kk); % landmark x position
    yLandmark = landmarkMap(2,kk); % landmark y position
    xDist = xLandmark-xTrue(1,:);
    yDist = yLandmark-xTrue(2,:);
    zObv(kk,:) = atan2(yDist,xDist) + sig_bearing*randn(1,timeStep);
end

%==========================================================================
% Evaluation
```

```
%=========================================================================

% Note: to make sure the evaluation process can be carried out smoothly,
% please put landmark_localization_EKF_eval.p in your current folder

% uncomment the following three lines of code to evaluate of your code using
    another dataset

% clear xTrue landmarkMap zObv
% close all
%
%
% [xTrue, landmarkMap, zObv] = landmark_localization_EKF_eval(1234); % input
    the last four digits of your student number

%========== Write your code here to process the sensor readings ==========

% Initialize EKF parameters
xInit = [-30 -5 atan2(5,0)]'; % initial state
xhat = xInit; % Initial state estimate (position x, position y and heading
    angle)
P = eye(3); % initial covariance matrix

omega = 0; % yaw rate

% Process noise covariance (Q)
Q = diag([sig_odometry^2, sig_odometry^2, sig_odometry^2]); % process noise
    covariance

% Measurement noise covariance (R)
R = sig_bearing^2 * eye(nLandmark)*50;

% Initialize output variables
xhatOut = zeros(3, timeStep);

% EKF Algorithm
for k = 1:timeStep
    % Prediction Step
    % State prediction based on bicycle model
    xhat(1) = xhat(1) + vel*cos(xhat(3))* Ts;
    xhat(2) = xhat(2) + vel*sin(xhat(3))*Ts;
    xhat(3) = xhat(3) + omega * Ts;


    % Update covariance prediction based on the motion model
    F = [1, 0, -vel*sin(xhat(3))*Ts;
         0, 1, vel*cos(xhat(3))*Ts;
         0, 0, 1]; % Jacobian of the motion model
    P_minus = F * P * F' + Q; % Covariance prediction

    % Update Step
    for kk = 1:nLandmark
        % Measurement model (h(x))
        dx = landmarkMap(1,kk) - xhat(1);
        dy = landmarkMap(2,kk) - xhat(2);
        range = sqrt(dx^2 + dy^2);
```

```
        H = [dy/range^2, -dx/range^2, 0]; % Jacobian of the measurement model
            per landmark

        % Predicted measurement
        zhat = atan2(dy, dx);

        % Measurement residual
        z = zObv(kk, k);
        z_res = z - zhat;

        % Ensure the residual is within -pi to pi
        if z_res > pi
            z_res = z_res - 2*pi;
        elseif z_res < -pi
            z_res = z_res + 2*pi;
        end

        % Kalman Gain
        K = P_minus * H' / (H * P_minus * H' + R(kk,kk));

        % Update state estimate and covariance
        xhat = xhat + K * z_res;
        P = (eye(3) - K * H) * P_minus;
    end

    % Store the estimated state at each time step
    xhatOut(:, k) = xhat;
end

% Plot estimated trajectory
figure(1)
plot(xhatOut(1,:), xhatOut(2,:), '--', 'LineWidth', 2);
legend('Landmark 1', 'Landmark 2', 'Landmark 3', 'Vehcile position', 'Estimated
    trajectory', 'Location', 'south');
xlabel('x position (m)');
ylabel('y position (m)');
axis equal;

xhatvel = vel*cos(xhatOut(3,:));
yhatvel = vel*sin(xhatOut(3,:));

%% postprocess of the results
% uncomment to use the code to plot the results
% be aware of the variable names when using this code
%
% posErr = xhatOut(1:2,:)-xTrue(1:2,:);
% posMSE = (sum(posErr(1,:).^2)+sum(posErr(2,:).^2))/size(posErr,2);

figure; title('Position estimation')
subplot(2,1,1);hold on;
plot(xTrue(1,:),'LineWidth',2);plot(xhatOut(1,:),'LineWidth',2);
legend('True x position','Estimated x position')
xlabel('simulation steps')
ylabel('x position (m)')

subplot(2,1,2);hold on;
```

```
plot(xTrue(2,:),'LineWidth',2);plot(xhatOut(2,:),'LineWidth',2);
legend('True y position','Estimated y position')
xlabel('simulation steps')
ylabel('y position (m)')


figure; title('Velocity estimation in x and y directions')
subplot(2,1,1);hold on;
plot(xTrue(3,:),'LineWidth',2);plot(xhatvel,'LineWidth',2);
legend('True x velocity','Estimated x velocity')
xlabel('simulation steps')
ylabel('x velocity (m/s)')

subplot(2,1,2);hold on;
plot(xTrue(4,:),'LineWidth',2);plot(yhatvel,'LineWidth',2);
legend('True y velocity','Estimated y velocity')
xlabel('simulation steps')
ylabel('y velocity (m/s)')
```