ME 425 LAB 4 Report
Kalman Filter Implementation For A Turtlebot

Arda Gencer
34124
Instructor: Mustafa Ünel

Fall 2025-2026
Due Date: 31/12/2025

Sabancı Üniversitesi

1 Introduction

In the lab 6 of ME425, a discrete time Kalman Filter was implemented on a Turtlebot to estimate the robot, position and orientation relative to a wall. To implement the filter, a dynamic model of the system and sensor measurements that are received from the Lidar sensor on top of the robot were used. These data were then fused by using the 5 Kalman equations for error correction. The necessary variables such as the predictions over time, the kalman gain over time and variance P over time was also obtained and saved to a vector for later plotting (although note that some of the data are missing due to some missing parts in the code which should save this data and due to limited lab time).

2 Procedure

1. For the first task, the distance from a wall was estimated by placing the robot parallel to a wall and getting the measurements as a single point measurement in the direction of the wall (this gives the perpendicular distance of the robot to the wall.). In this case, the state that is estimated is only a scalar, so all the Kalman equations and the variables used in them will also be scalars. The 5 implemented Kalman Filter quations are as follows:

$$x_k^- = x_{k-1}^+ \tag{1}$$
$$P_k^- = P_{k-1}^+ + Q \tag{2}$$
$$K_k = \frac{P_k^-}{P_k^- + R} \tag{3}$$
$$x_k^+ = x_k^- + K_k\left(z_k - x_k^-\right) \tag{4}$$
$$P_k^+ = (1 - K_k)\,P_k^- \tag{5}$$

   Equation 1 is formulated like this since the perpendicular distance to the wall should stay the same throughout the motion. These 5 equations were then implemented inside a loop in MATLAB to estimate the distance to the wall. The measurements and the state is updated at each iteration of the loop.

   For this task, the robot was tested 3 times for both a parallel start and an angular start. One for equal variances for both model and sensor, one for high variance for model (sensor reliance), and one for higher sensor noise (model reliance).

2. Before the implementation of the Kalman Filter in the second task, first, a least squares method was used to fit the readings that are gotten from the lidar to produce a line that is as accurate as possible. This fitting was done by extracting all the valid Lidar readings that are gotten and using the polyfit function on them by specifying the dimension as 1.
   For the second task, the 1D filter that was implemented in task 1 was expanded so that it could estimate both the perpendicular distance to the wall and also the angle that the robot makes with the wall. Now, since the state of the robot that will be estimated

is a 2D vector, the Filter equations were also modified and expanded to support a state of 2 dimensions. The variance was also changed here to be a 2X2 matrix. The equations implemented here are as follows:

$$A = I_2, \qquad H = I_2, \tag{6}$$

$$B = \begin{bmatrix} \sin\big(x_{k-1}^{+}(2)\big)\,\Delta t \\ 0 \end{bmatrix}, \qquad u_k = v_k \tag{7}$$

$$\text{Prediction:} \qquad x_k^{-} = A\,x_{k-1}^{+} + B\,u_k, \tag{8}$$

$$P_k^{-} = A\,P_{k-1}^{+}A^{\mathsf{T}} + Q \tag{9}$$

$$\text{Update:} \qquad S_k = H\,P_k^{-}H^{\mathsf{T}} + R, \tag{10}$$

$$K_k = P_k^{-}H^{\mathsf{T}}S_k^{-1}, \tag{11}$$

$$x_k^{+} = x_k^{-} + K_k\big(z_k - H\,x_k^{-}\big), \tag{12}$$

$$P_k^{+} = \big(I_2 - K_k H\big)P_k^{-} \tag{13}$$

The results gotten from both of these experiments were plotted both in real time and they were also saved into vectors to be later used in a third file that is dedicated to plotting them after the lab only. The results obtained can be seen below in the Results section(except for the missing ones).

3 Results

3.1 Task 1: 1D Lateral Estimation

3.1.1 Straight Heading (Parallel to Wall)

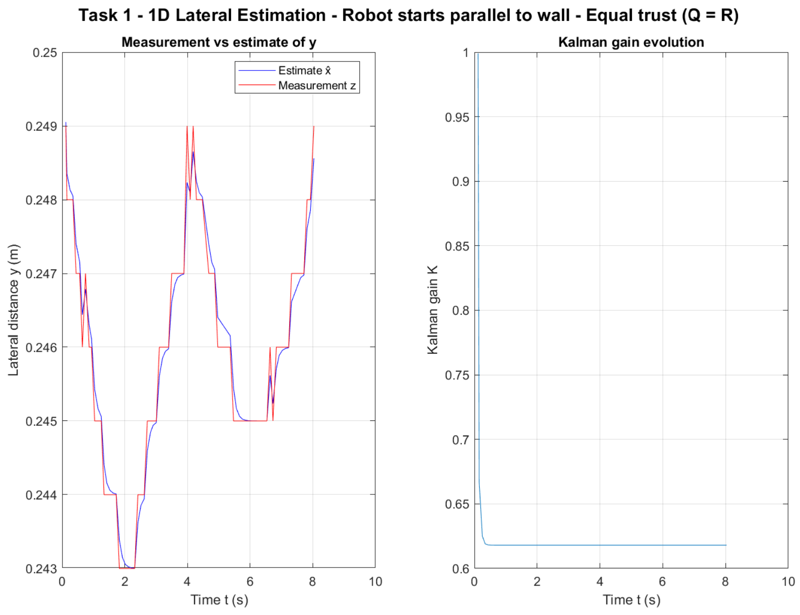

Figure 1: Task 1 - Robot starts parallel to wall - Equal trust $(Q = R)$

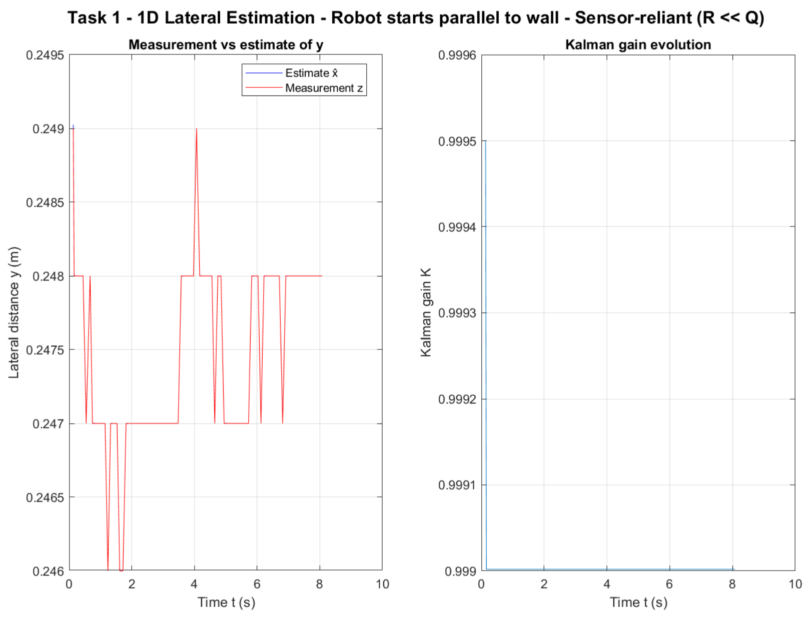

Figure 2: Task 1 - Robot starts parallel to wall - Sensor-reliant ($R \ll Q$)

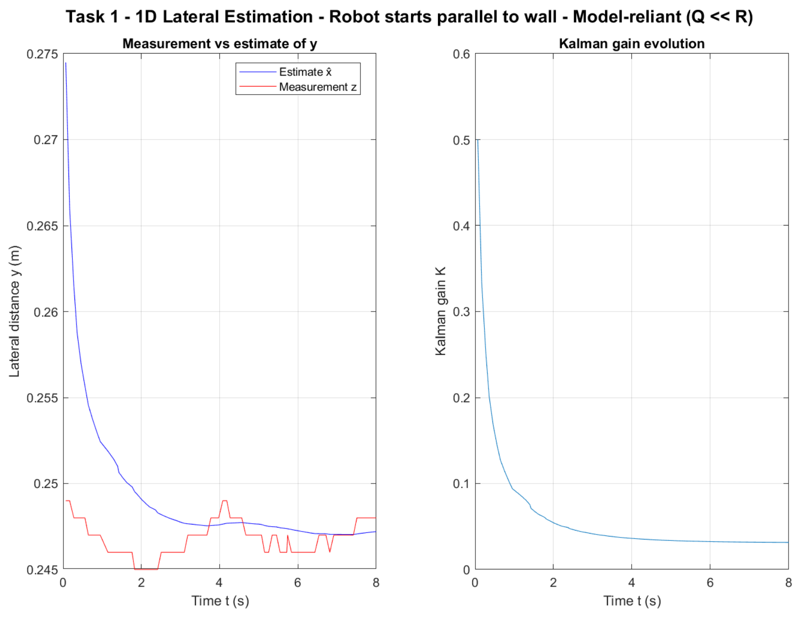

Figure 3: Task 1 - Robot starts parallel to wall - Model-reliant ($Q \ll R$)

3.1.2 Angled Heading

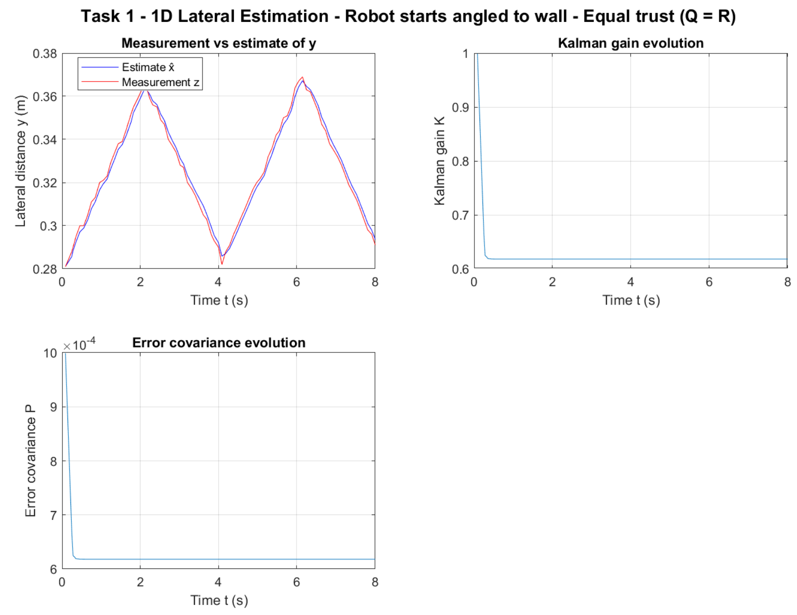

Figure 4: Task 1 - Robot starts angled to wall - Equal trust $(Q = R)$

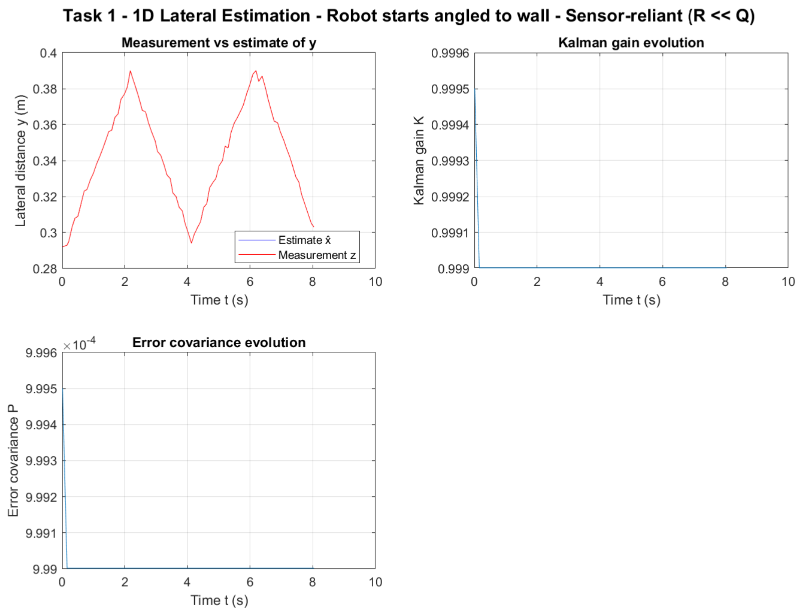

Figure 5: Task 1 - Robot starts angled to wall - Sensor-reliant $(R \ll Q)$

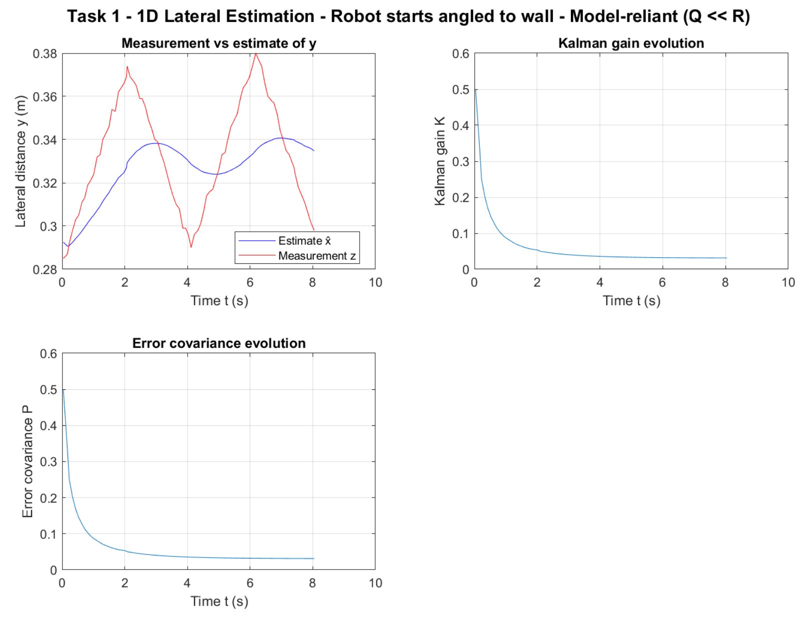

Figure 6: Task 1 - Robot starts angled to wall - Model-reliant ($Q \ll R$)

3.2 Task 2: 2D Pose Estimation

3.2.1 Straight Heading (Parallel to Wall)

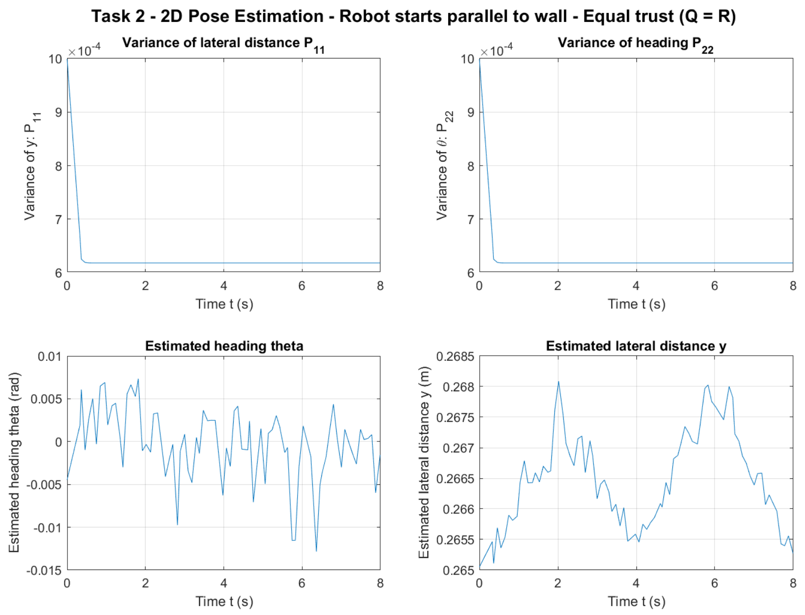

Figure 7: Task 2 - Robot starts parallel to wall - Equal trust ($Q = R$)

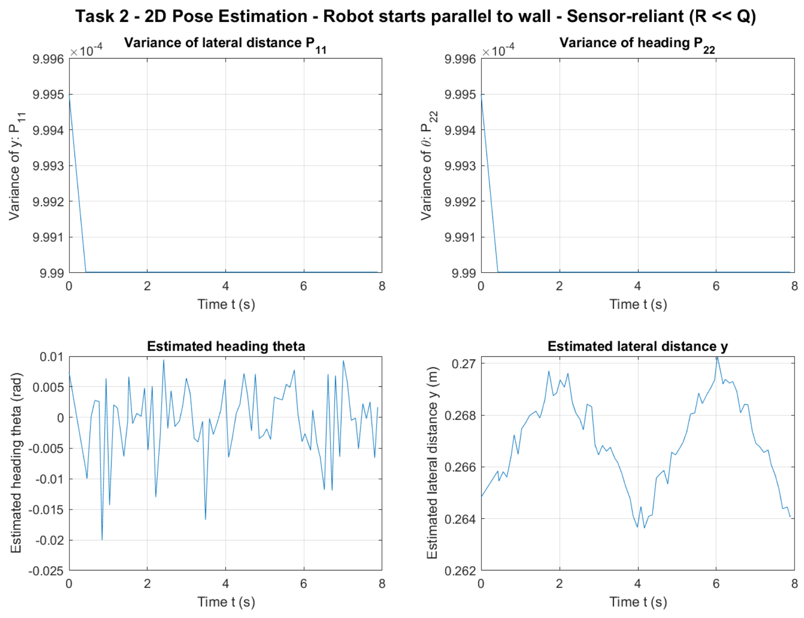

Figure 8: Task 2 - Robot starts parallel to wall - Sensor-reliant ($R \ll Q$)

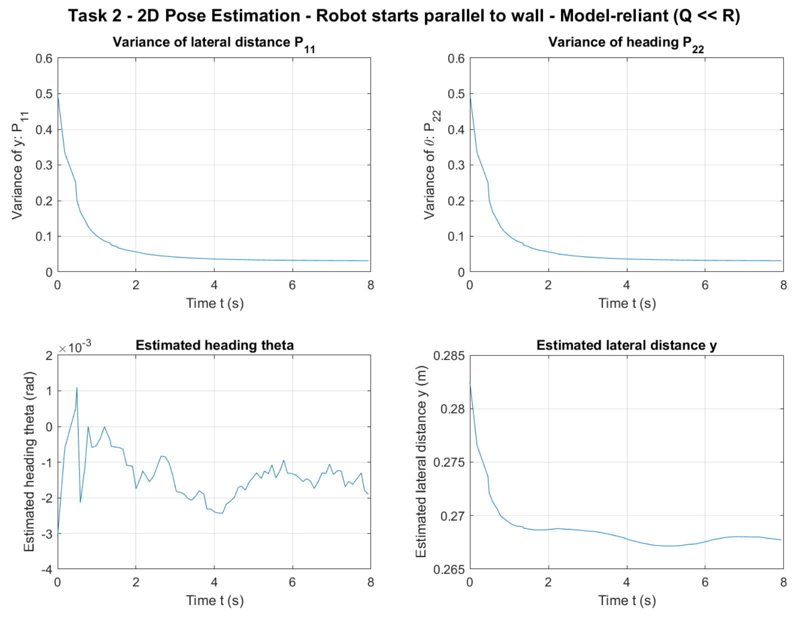

Figure 9: Task 2 - Robot starts parallel to wall - Model-reliant ($Q \ll R$)

3.2.2 Angled Heading

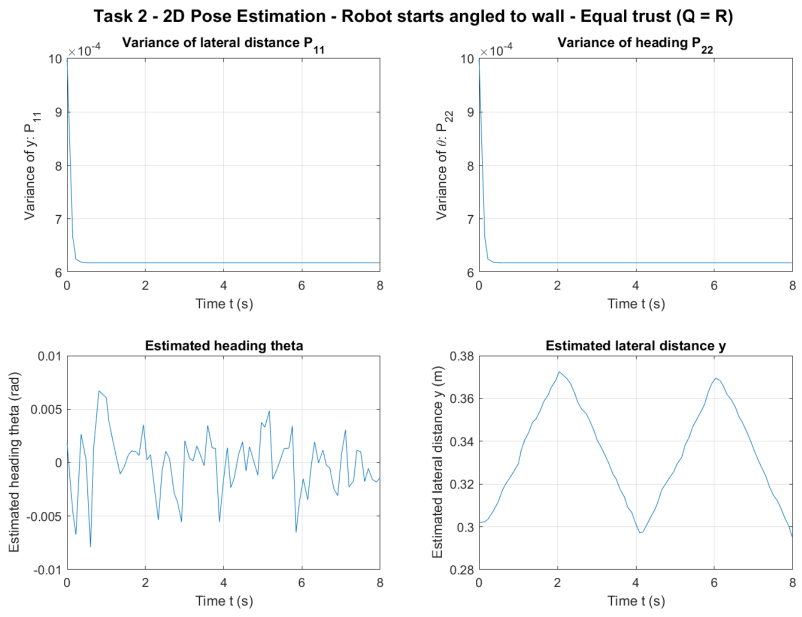

Figure 10: Task 2 - Robot starts angled to wall - Equal trust ($Q = R$)

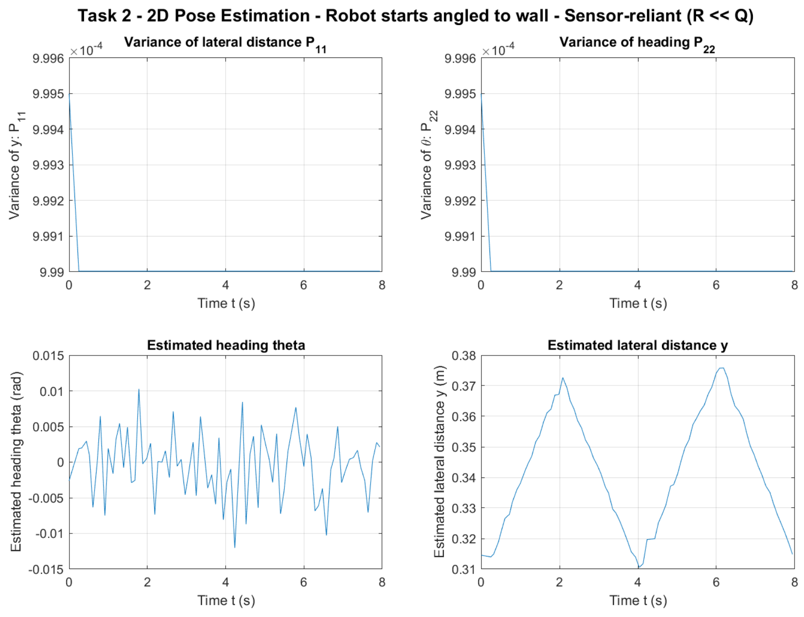

Figure 11: Task 2 - Robot starts angled to wall - Sensor-reliant ($R \ll Q$)

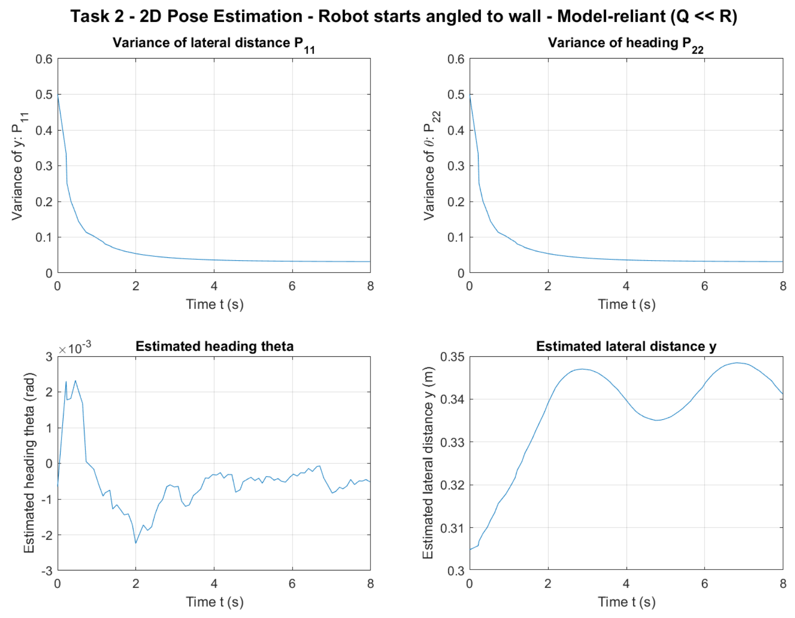

Figure 12: Task 2 - Robot starts angled to wall - Model-reliant ($Q \ll R$)

4 Conclusion

In the sixth lab of ME425, a Kalman filter for a differential drive robot was implemented for pose estimation. While doing this, the filter was tested with several starting conditions and filter parameters and the effect of these parameters/ initial conditions were also observed and plotted.

5 Discussion

1. 1D vs. 2D Dynamics: In Task 1, the filter assumes the robot is parallel to the wall. If the robot starts with a 10○ heading, how does the 1D estimate ($\hat{x}$) behave compared to the 2D estimate from Task 2?

   When estimating $\hat{x}$, no major difference was seen for the 1D Kalman Filter and the 2D one. Both the estimated values, the Kalman gain, and the covariances were almost entirely the same for the two cases.

2. Filter Tuning: Discuss how the Kalman Gain (K) and error covariance (P) evolved in your plots. Did the variance Pk converge to zero? Explain why the filter never reaches absolute certainty.

   Both the Kalman gain and the covariances usually start off high at t = 0. However, after a very small time has passed, they converge to a lower value than they have started. Although, for the Kalman gain this final value is vastly different depending on the noise levels of the model and the sensors. As for why the filter never reaches a variance level of 0, this is because it is always assumed for the sensor to have a small non-zero noise level(Q). Due to this the P value that is computed pre-measurement always has a non zero value. And when we update this value, it obtains another (possibly very small) but still a non-zero value.

3. Trust Scenarios: Compare your results for the following three scenarios:

   - Q and R are very close to each other.
   - Q is significantly higher than R.
   - R is significantly higher than Q.

   Which scenario makes the robot ”trust” its physical sensors (LiDAR) more than its mathematical model?

   When R and Q are taken equally, the estimates look mostly like the sensor measurements, albeit with some variances throughout the motion. As for the Kalman gain, it remains at a medium value of about 0.5-0.6. For the variance of the error, much higher variance values were observed when a model-reliant model was used. A sensor reliance model causes much lower levels of variance, but the lowest variance levels were usually observed on equal reliance filters. As for the behaviour of the robot, taking the model noise higher usually causes the robot to trust its sensors more, whereas taking the sensor noise high causes it to rely more on model and less on the sensors.

6 Appendix

Listing 1: Kalman1D.m. Implements the 1D Kalman Filter for task 1

```
clc; clear; close all;
%% REAL - TIME PLOTTING SETUP
x_values = [];
z_values = [];
P_values = [];
K_values = [];
t_values = [];
figure;
subplot(2,2,1);
xGraph = plot(nan,nan);
title("X Values");
subplot(2,2,2);
zGraph = plot(nan,nan);
title("Measurements");
subplot(2,2,3);
PGraph = plot(nan,nan);
title("Covariances");
subplot(2,2,4);
```

```
KGraph = plot(nan,nan);
title("Kalman Gains");

clear node; % clear previous node handle if it exists

setenv('ROS_DOMAIN_ID','43'); % Set to your robots ROS_DOMAIN_ID (check
    the robot"s ID card)
setenv('ROS_LOCALHOST_ONLY', '0'); % 0 implies multi-host communication
setenv('RMW_IMPLEMENTATION','rmw_fastrtps_cpp'); % Middleware

% Create a unique node name for this MATLAB session
node = ros2node('Kalman1D');
%% Subscribers And Publishers

%Lidar Subscriber
scanSub = ros2subscriber(node, "/scan", "sensor_msgs/LaserScan",...
"Reliability", "besteffort", "Durability", "volatile", "Depth", 10);

%Velocity Publisher
velPub = ros2publisher(node,"/cmd_vel","geometry_msgs/Twist", ...
"Reliability","reliable","Durability","volatile","Depth",10);

velMsg = ros2message(velPub);

%% Variables
Q = 1;  %Model Variance
R = 0.001;   %Measurement Variance

%Initial Measurement
scanMsg = receive(scanSub, 10) ;
idx = round((pi /2 - scanMsg.angle_min) / scanMsg.angle_increment) + 1;
old_z = scanMsg.ranges(max(1, min( idx, length(scanMsg.ranges ))));
old_x_k = 0.3;
old_P = 1;

%% MAIN LOOP
timer = tic; duration = 8;
while toc( timer ) < duration
% Forward Backward motion
currentTime = toc( timer ) ;
if mod(currentTime, 4) < 2 , v = 0.1; else , v = -0.1; end
velMsg.linear.x = v; send(velPub, velMsg);
% Get Measurements
scanMsg = receive ( scanSub , 10) ;
idx = round (( pi /2 - scanMsg.angle_min ) / scanMsg.angle_increment ) +1;
z = scanMsg.ranges(max(1, min(idx, length(scanMsg.ranges))));
if isnan(z)
    z = old_z;
end
% Kalman Filter
x_k_before = old_x_k;
P_before = old_P + Q;
kalman_gain = P_before/(P_before+R);
x_k_after = x_k_before + kalman_gain * (z - x_k_before);
```

```
P_after = (1-kalman_gain) * P_before;
% Update Data and Plots
old_x_k = x_k_after;
old_P = P_after;
old_z = z;

x_values = [x_values x_k_after];
z_values = [z_values z];
P_values = [P_values P_after];
K_values = [K_values kalman_gain];
t_values = [t_values toc( timer )];
set(xGraph,"XData",t_values,"YData",x_values);
set(zGraph,"XData",t_values,"YData",z_values);
set(PGraph,"XData",t_values,"YData",P_values);
set(KGraph,"XData",t_values,"YData",K_values);

drawnow limitrate ;
end

%Stop Robot
velMsg.linear.x = 0; send(velPub, velMsg);
```

Listing 2: Kalman2D.m. Implements the 2D Kalman Filter along with line fitting etc. for pose estimation for task 2

```
clc; clear; close all;

%% REAL - TIME PLOTTING SETUP
y_est_values = [];
theta_est_values = [];
P11_values = [];
P22_values = [];
t_values = [];

figure('Position', [100 100 1200 600]);

subplot(2,2,1);
yGraph = plot(nan, nan, 'b-', 'LineWidth', 2);
xlabel('Time␣(s)'); ylabel('y␣(m)');
title('Lateral␣Distance␣y');
grid on;

subplot(2,2,2);
thetaGraph = plot(nan, nan, 'r-', 'LineWidth', 2);
xlabel('Time␣(s)'); ylabel('\theta␣(rad)');
title('Heading␣Angle␣\theta');
grid on;

subplot(2,2,3);
P11Graph = plot(nan, nan, 'b-', 'LineWidth', 2);
xlabel('Time␣(s)'); ylabel('Var(y)');
title('Variance␣P_{11}');
grid on;

```

```
subplot(2,2,4);
P22Graph = plot(nan, nan, 'r-', 'LineWidth', 2);
xlabel('Time␣(s)'); ylabel('Var(\theta)');
title('Variance␣P_{22}');
grid on;

clear node;

setenv('ROS_DOMAIN_ID','43');
setenv('ROS_LOCALHOST_ONLY', '0');
setenv('RMW_IMPLEMENTATION','rmw_fastrtps_cpp');

node = ros2node('Kalman2D');

%% Subscribers And Publishers

% Lidar Subscriber
scanSub = ros2subscriber(node, "/scan", "sensor_msgs/LaserScan",...
    "Reliability", "besteffort", "Durability", "volatile", "Depth", 10);

% Velocity Publisher
velPub = ros2publisher(node, "/cmd_vel", "geometry_msgs/Twist", ...
    "Reliability", "reliable", "Durability", "volatile", "Depth", 10);

velMsg = ros2message(velPub);

%% Initialization

Q = 0.001 * eye(2);
R = 0.01 * eye(2);
delta_t = 0.1;

old_P = eye(2);
old_x_k = [0.3; 0];

%% MAIN LOOP

timer = tic;
duration = 8;

while toc(timer) < duration
    % Forward Backward motion
    currentTime = toc(timer);
    if mod(currentTime, 4) < 2
        v = 0.1;
    else
        v = -0.1;
    end
    velMsg.linear.x = v;
    send(velPub, velMsg);

    % Get Measurements
    scanMsg = receive(scanSub, 10);
    angles = scanMsg.angle_min + (0:length(scanMsg.ranges)-1) * scanMsg.
```

```
        angle_increment;
    idx = find(angles >= pi/3 & angles <= 2*pi/3);
    r_wall = scanMsg.ranges(idx);
    th_wall = angles(idx);
    valid = r_wall > scanMsg.range_min & r_wall < scanMsg.range_max;

    % Fit Measurements
    x_pts = r_wall(valid) .* cos(th_wall(valid));
    y_pts = r_wall(valid) .* sin(th_wall(valid));

    if length(x_pts) < 5  %     SAFETY CHECK!
        continue;
    end

    p = polyfit(x_pts, y_pts, 1);
    m = p(1);
    b = p(2);

    theta_meas = atan(m);
    y_meas = b / sqrt(m^2 + 1);
    z_k = [y_meas; theta_meas];

    % Kalman Filter
    A = eye(2);
    H = eye(2);

    B = [sin(old_x_k(2)) * delta_t; 0];  % 2x1
    u_k = v;

    % Prediction
    x_k_before = A * old_x_k + B * u_k;
    P_before = A * old_P * A' + Q;

    % Update
    S = H * P_before * H' + R;
    kalman_gain = P_before * H' / S;
    x_k_after = x_k_before + kalman_gain * (z_k - H * x_k_before);
    P_after = (eye(2) - kalman_gain * H) * P_before;

    % Update state
    old_P = P_after;
    old_x_k = x_k_after;

    % Store data
    y_est_values = [y_est_values x_k_after(1)];
    theta_est_values = [theta_est_values x_k_after(2)];
    P11_values = [P11_values P_after(1,1)];
    P22_values = [P22_values P_after(2,2)];
    t_values = [t_values currentTime];  %     DON'T FORGET THIS!

    % Update all 4 plots
    subplot(2,2,1);
    set(yGraph, "XData", t_values, "YData", y_est_values);

    subplot(2,2,2);
    set(thetaGraph, "XData", t_values, "YData", theta_est_values);

    subplot(2,2,3);
    set(P11Graph, "XData", t_values, "YData", P11_values);

    subplot(2,2,4);
    set(P22Graph, "XData", t_values, "YData", P22_values);

    drawnow limitrate;
end

% Stop Robot
velMsg.linear.x = 0;
send(velPub, velMsg);
```