University of Twente

Faculty of Electrical Engineering, Mathematics and Computer Science

Assignments - Optimal Estimation in Dynamic Systems

Lecturer(s): F. van der Heijden and F.J. Siepel

Student: Guilherme Soares Silvestre

Enschede, April 12, 2023

Contents

1 Exercise 1

The Bayesian estimator is useful to estimate parameters from probabilities densities. In this case, we apply Bayesian to measure depth using a ultrasonic sensor on a boat. To do that, we have a sensor model that we assumed a mixture Gaussian model that considered harmonics. Therefore, we can use this model with the prior knowledge of the environment, assumed a generalized normal distribution.

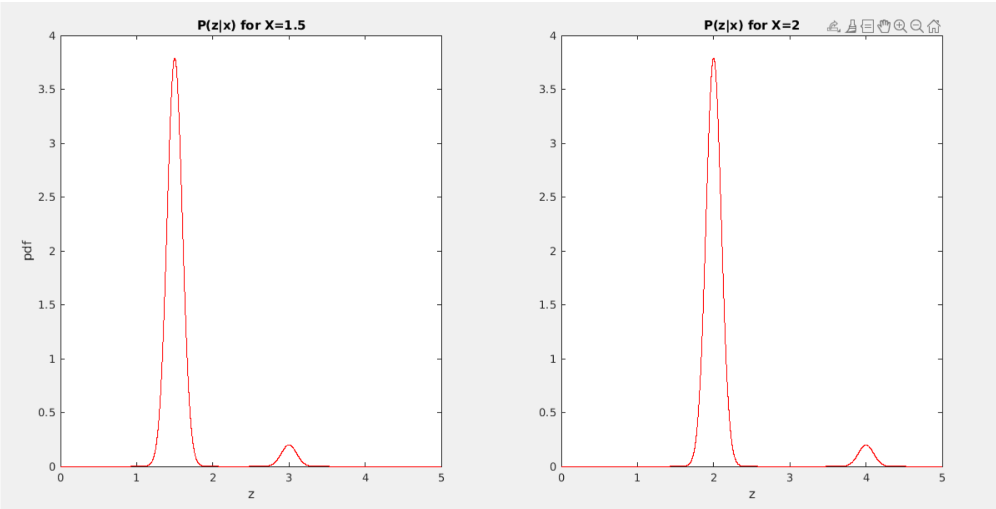

Figure 1.1: Sensor model probability density function (P(z|x)) for x=1.5 and x=2.5.

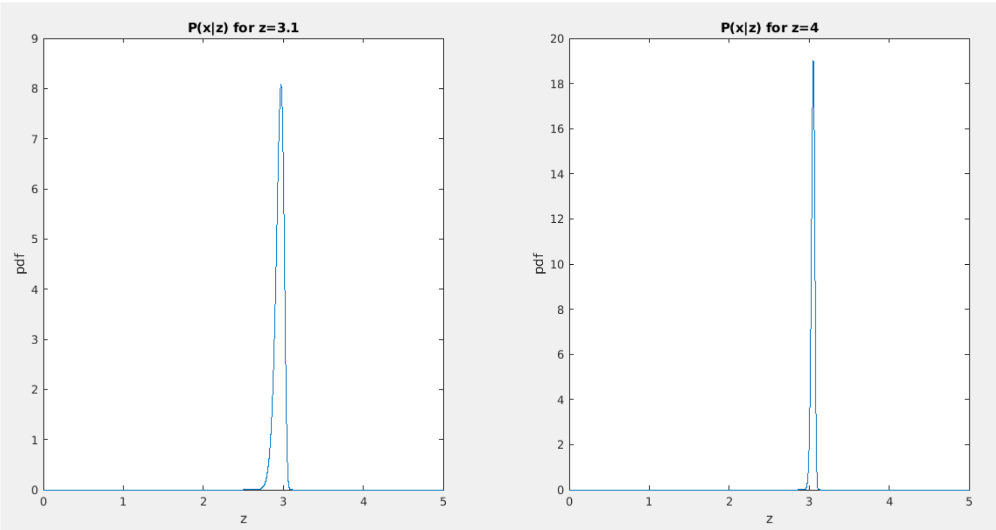

Figure 1.2: Posterior *pdf* after applying Bayes formula.

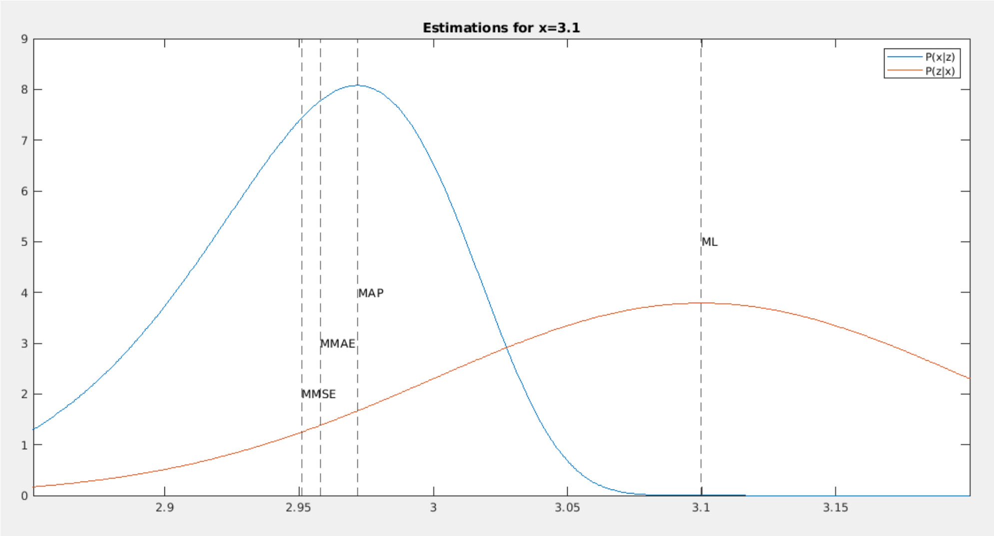

Figure 1.3: Estimator results for x = 3.1.

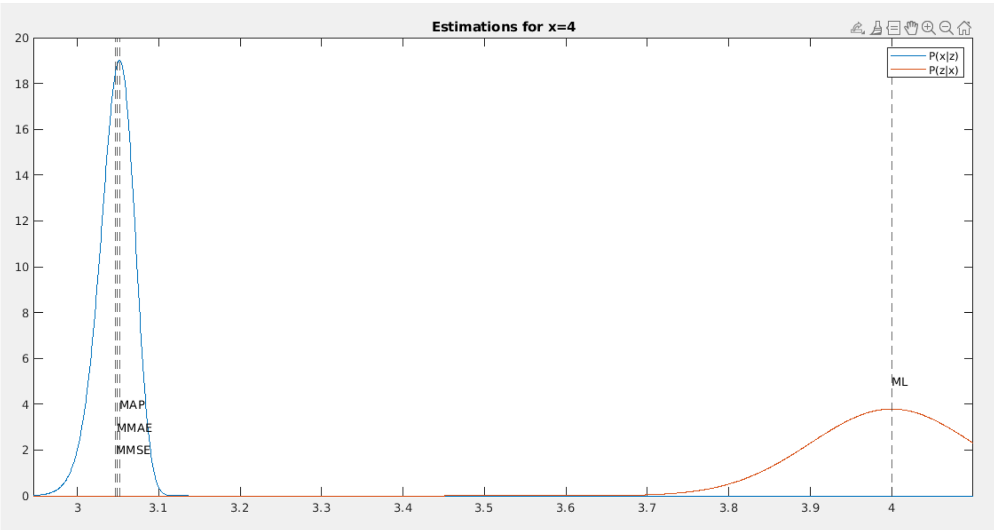

Figure 1.4: Estimator results for x = 4.0.

```
Estimations for z=3.1
MMSE: x_hat = 2.95 m | Risk = 1.09 m^2
MMAE: x_hat = 2.96 m | Risk = 0.01 m
MAP: x_hat = 2.97 m | Risk = 0.60 m
ML: x_hat = 3.10 m

Estimations for z=4
MMSE: x_hat = 3.05 m | Risk = 2.63 m^2
MMAE: x_hat = 3.05 m | Risk = 0.00 m
MAP: x_hat = 3.05 m | Risk = 0.05 m
ML: x_hat = 4.00 m
```

Figure 1.5: Risk using MAP + absolute cost, MMSE + quadratic cost and MMAE + uniform cost.

The sensor model probability density function (*pdf*) (P(z|x)) is shown in Fig. 1.1 for $x = 1.5$ and $x = 2$. Applying the Bayesian filter, we achieve the posterior *pdf*, meaning the likelihood of the depth (Fig 1.2). With that, it is needed to apply a optimal estimator to reduce a cost function (e.g absolute, quadratic or uniform). This is done by applying estimators named MAP, MMSE or MMAE, respectively.

Furthermore, estimators that doesn't need a cost function was also applied, named Maximum Likelihood (ML) estimator. Fig. 1.3 and Fig. 1.4 show the estimators results for each one with x = 3.1 and x = 4.0 respectively. Beside, it's important to analyze that how the prior knowledge affect the final *pdf* and then the final estimates. However, with ML estimator, the prior is not considered, being a good choice in a case that the prior knowledge is not present.

For evaluate the estimators, the risk was analysed. Fig. 1.5 shows the risk for each estimator using x = 3.1 and x = 4.0. In this situation, the lower risk was the MAP estimator. However, it is not a general rule and in this case, the MAP estimator with absolute cost function presented the lowest risk in comparison of the other methods.

```matlab

% exercise 1 optimal estimation of dynamic systems
% author: Guilherme Soares Silvestre

close all
clear

%%%%%%%%%%%%%%%%%%%%%%%%
% sensor model for x=2
[x, ~, pdf_z_x, ~, ~, ~, ~, ~, ~, ~, ~] = full_estimates(2);
% subplot of pdf_z_x
fig_sensor = figure("Name", "Sensor model");
subplot(1,2,2)
plot(x, pdf_z_x, 'r')
title("P(z|x) for X=2")
xlabel('z')

%%%%%%%%%%%%%%%%%%%%%%%%
% sensor model for x=1.5
[x, ~, pdf_z_x, ~, ~, ~, ~, ~, ~, ~, ~] = full_estimates(1.5);
% subplot of pdf_z_x
figure(fig_sensor)
subplot(1,2,1)
plot(x, pdf_z_x, 'r')
title("P(z|x) for X=1.5")
ylabel("pdf")
xlabel('z')

%%%%%%%%%%%%%%%%%%%%%%%%
% full anlysis for x=3.1
[x, ~, pdf_z_x, pdf_x_z, x_mmse, x_mmae, x_map, x_ml, risk_mmse, ...
    risk_mmae, risk_map] = full_estimates(3.1);

fprintf('\n')
disp("Estimations for z=3.1")
fprintf("MMSE: x_hat = %.2f m | Risk = %.2f m^2\n", x_mmse, risk_mmse)
fprintf("MMAE: x_hat = %.2f m | Risk = %.2f m\n", x_mmae, risk_mmae)
fprintf("MAP: x_hat = %.2f m | Risk = %.2f m\n", x_map, risk_map)
fprintf('ML: x_hat = %.2f m\n', x_ml)
```

```


fig_post = figure("Name", "Posterior probabilities");
subplot(1,2,1)
plot(x, pdf_x_z)
title("P(x|z) for z=3.1")
ylabel("pdf")
xlabel("z")

figure
plot(x, pdf_x_z)
hold on
plot(x, pdf_z_x)
x_labels = [x_mmse x_mmae x_map x_ml];
labels = ["MMSE" "MMAE" "MAP" "ML"];
xline(x_labels, '--')
xlim([x_mmse - 0.1, x_ml + 0.1])
text(x_labels, [2 3 4 5], labels)
legend(["P(x|z)" "P(z|x)"])
title("Estimations for x=3.1")

%%%%%%%%%%%%%%%%%%%%%%%%
% full analysis of x=4
[x, ~, pdf_z_x, pdf_x_z, x_mmse, x_mmae, x_map, x_ml, risk_mmse, ...
    risk_mmae, risk_map] = full_estimates(4);

% display estimates and risks
fprintf('\n')
disp("Estimations for z=4")
fprintf("MMSE: x_hat = %.2f m | Risk = %.2f m^2\n", x_mmse, risk_mmse)
fprintf("MMAE: x_hat = %.2f m | Risk = %.2f m\n", x_mmae, risk_mmae)
fprintf("MAP: x_hat = %.2f m | Risk = %.2f m\n", x_map, risk_map)
fprintf('ML: x_hat = %.2f m\n', x_ml)

% plot posterior pdf
figure(fig_post)
subplot(1,2,2)
plot(x, pdf_x_z)
title("P(x|z) for z=4")
```

```
ylabel("pdf")
xlabel("z")

% plot estimates on top of the pdf
figure
plot(x, pdf_x_z)
hold on
plot(x, pdf_z_x)
x_labels = [x_mmse x_mmae x_map x_ml];
labels = ["MMSE" "MMAE" "MAP" "ML"];
xline(x_labels, '--')
xlim([x_mmse - 0.1, x_ml + 0.1])
text(x_labels, [2 3 4 5], labels)
legend(["P(x|z)" "P(z|x)"])
title("Estimations for x=4")
```

Listing 1.1: Source code for exercise 1

2 Exercise 2

In the exercise 2 we developed the knowledge about the most general form of the Linear MMSE estimator and the Unbiased Linear MMSE estimator. Fig. 2.1 and Fig. 2.2 show the simulated measurements and estimation of each measurement. It was good for understanding performance of the estimators like bias and error variance. Fig. 2.3 show the performance parameters of each estimator.

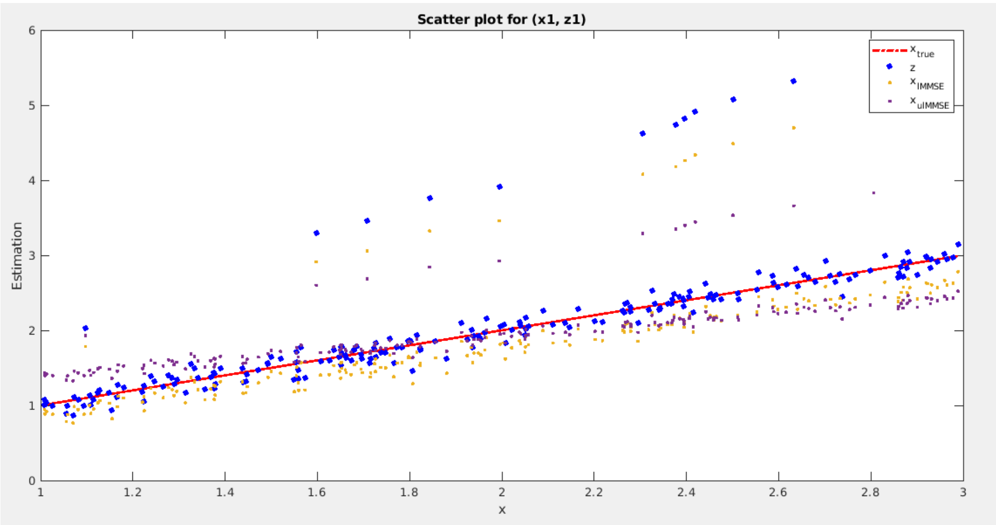

Figure 2.1: lMMSE and ulMMSE in a simulated data.

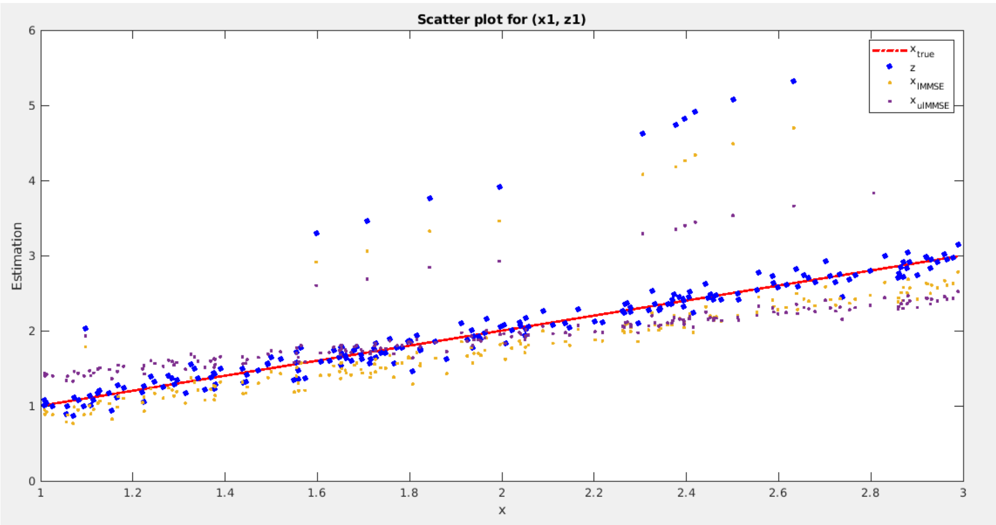

Figure 2.2: lMMSE and ulMMSE in a second dataset.

```
Dataset 1
Optimal linear MMSE: x_{lMMSE} = 0.883 * z
Overall bias: 0.11791
Error variance: 0.224
Optimal Unbiased Linear MMSE: x_{ulMMSE} = 0.523 * z + 0.874
Overall bias: -0.00000
Error variance: 0.135

Dataset 2
Optimal linear MMSE: x_{lMMSE} = 0.905 * z
Overall bias: 0.10110
Error variance: 0.209
Optimal Unbiased Linear MMSE: x_{ulMMSE} = 0.570 * z + 0.824
Overall bias: -0.00000
Error variance: 0.136
```

Figure 2.3: Performance parameters of the lMMSE and ulMMSE estimators.

We can analyse that the unbiased linear MMSE (ulMMSE) presents a null bias, so that the mean error is zero. In this caso the ulMMSE was more stable than lMMSE as we can see in the error variance of both datasets. Furthermore, the second dataset is important to validate our analysis made on the first dataset, making it also unbiased.

```
% exercise 2 optimal estimation of dynamic systems
% author: Guilherme Soares Silvestre

close all
clear

% first dataset
load("data exercise 2/depthgauge_data_set1.mat")
estimate_from_data(x1, z1, 1)

% second dataset
load("data exercise 2/depthgauge_data_set2.mat")
estimate_from_data(x2, z2, 2)

% function for estimating and alaysing
function estimate_from_data(x, z, data_id)

    % scatter plot
    figure
    plot(x, x, 'Color', 'r', 'LineStyle','-.', 'LineWidth', 2)
    hold on
    plot(x, z, 'Marker','o','MarkerSize',5,'LineStyle','none','MarkerFaceColor','b', '
   Color', 'b')

    % calculating the mean of the data
    alpha_l = mean(x.*z)./mean(z.*z);

    % plot linear MMSE
    x_l = alpha_l.*z;
    plot(x, x_l, '.')

    % unbiased estimator
    alpha_ul = (mean(x.*z)-mean(x)*mean(z))./var(z);
    beta_ul = mean(x) - alpha_ul.*mean(z);
    x_ul = alpha_ul.*z+beta_ul;

    % plot unbiased linear MMSE
    plot(x, x_ul, '.')
```

```

    legend('x_{true}', 'z', 'x_{lMMSE}', 'x_{ulMMSE}')
    t = sprintf("Scatter plot for (x%d, z%d)", data_id, data_id);
    title(t)
    xlabel("x")
    ylabel("Estimation")


    % linear MMSE performance
    e_l = x - x_l;
    bias_l = mean(e_l);
    var_l = var(e_l);

    % Unbiased linear MMSE performance
    e_ul = x - x_ul;
    bias_ul = mean(e_ul);
    var_ul = var(e_ul);

    fprintf("\nDataset %d\n", data_id)
    fprintf("Optimal linear MMSE: x_{lMMSE} = %.3f * z\n", alpha_l)
    fprintf("Overall bias: %.5f\n", bias_l)
    fprintf("Error variance: %.3f\n", var_l)
    fprintf("Optimal Unbiased Linear MMSE: x_{ulMMSE} = %.3f * z + %.3f\n", alpha_ul,
    beta_ul)
    fprintf("Overall bias: %.5f\n", bias_ul)
    fprintf("Error variance: %.3f\n", var_ul)

end
```

Listing 2.1: Source code for exercise 2

3 Exercise 3

Covariance matrix does an important role on the understanding of the uncertainty of a set of variables. First defined as the component that represent the spread of the data on a multivariate Gaussian distribution, the same function is used in the optimal estimation field. In this exercise we developed ways to visualize the uncertainty of a estimation using the covariance matrix applied to a estimation of a boat that sees a lighthouse with known position.

With the covariance matrix, we can find the diagonal matrix represented by the eigenvalues of the covariance matrix, that each element represents the squared standard deviation of a state. Also, the eigenvectors represents a coordinate system that we can use to represent a ellipse in the plane with scale determined by the eigenvalues. This representation is useful because it permits us to know where is the most probable region that is given by our estimators.

The analysis was performed with an update step. With that, it was possible to see the covariance matrix changing and getting smaller with a measurement. Fig 3.1 shows the uncertainty regions defined by the covariance matrix. The bigger one represent the prior state of the boat before the measurement. The smaller and translated one represents the new region of belief of the boat, after the measurement.

Furthermore, we analyzed the effect of changing system parameters. The standard deviation of the measure of the angle to the lighthouse (Fig 3.2) and the covariance matrix of the prior pose (Fig 3.3) were changed. We can analyze that a poor measurement impact on how our belief changes latter, but if we have a poor prior belief but a good measurement, the knowledge of the location of the system is highly increased.

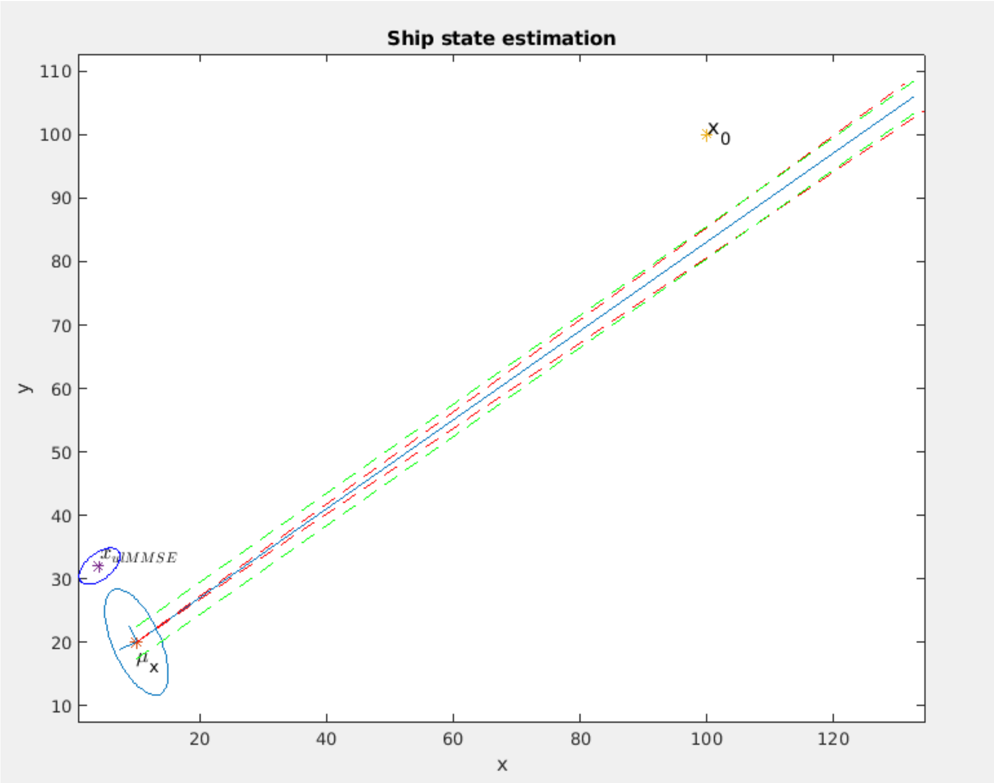

Figure 3.1: Representation of the line of sight with some uncertainty, likelihood ellipses given by covariance matrix and lighthouse position x0.

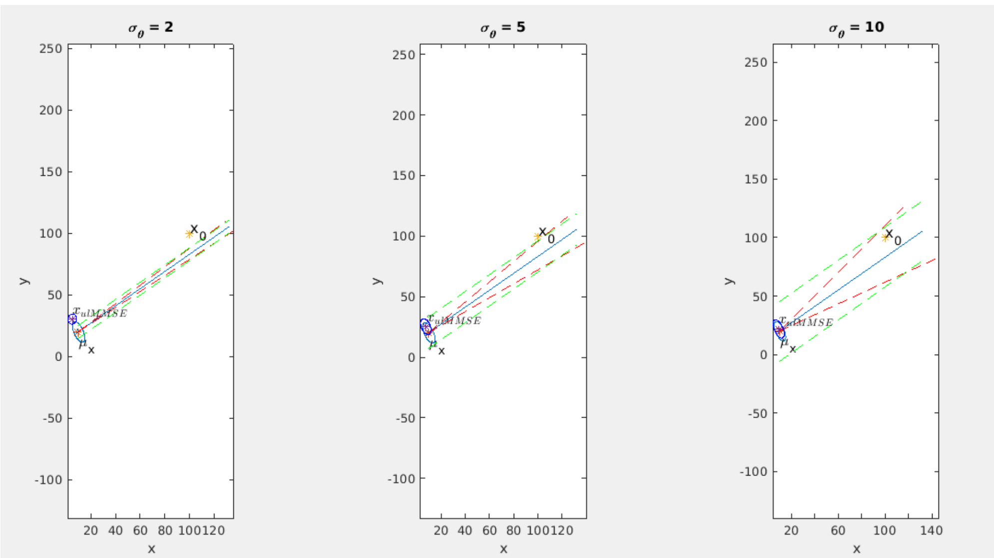

Figure 3.2: Effect of changing the measurement standard deviation before the update step.

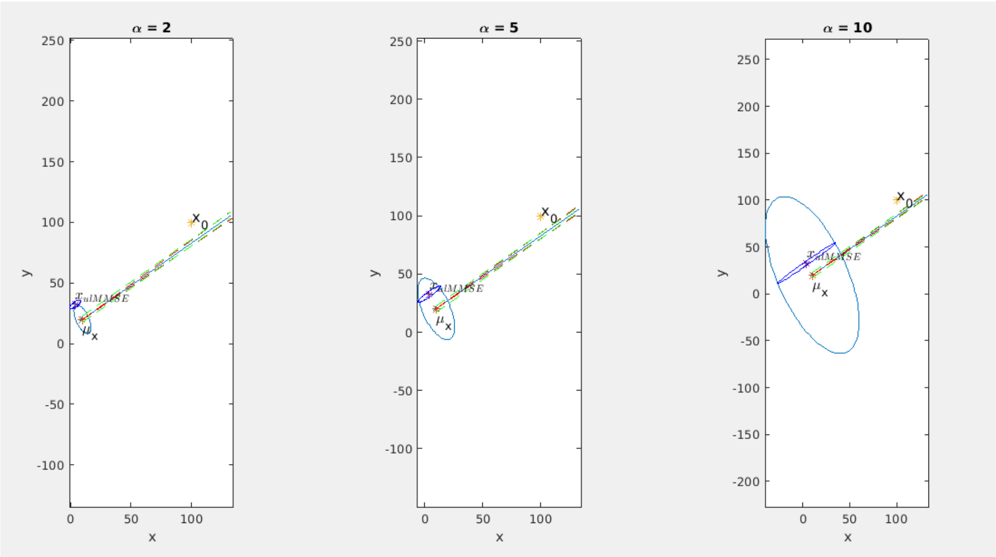

Figure 3.3: Effect of changing the prior pose covariance before the update step.

```


% ex 3
clear
close all

% defining the system
ux = [10 20]'; %m
x0 = [100 100]'; %m
theta = deg2rad(35); %rad
sd_theta = deg2rad(1); %rad
Cx = [25 -25;
      -25 70];

% Figure 1
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)

%%%%%%%%%%%%%%%%%%%%
% changing sd theta
% sd_theta = 2
sd_theta = deg2rad(2);
figure
subplot(1,3,1)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\sigma_{\theta} = 2")
```

```
% sd_theta = 5
sd_theta = deg2rad(5);
subplot(1,3,2)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\sigma_{\theta} = 5")

% sd_theta = 10
sd_theta = deg2rad(10);
subplot(1,3,3)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\sigma_{\theta} = 10")

%%%%%%%%%%%%%%%%%%%%
% changing covariance to a factor of alpha
% reseting sd_theta to 1
sd_theta = deg2rad(1);

% alpha = 2
Cx = 2*Cx;
figure
subplot(1,3,1)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\alpha = 2")

% alpha = 5
Cx = 5*Cx;
subplot(1,3,2)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\alpha = 5")

% alpha = 10
Cx = 10*Cx;
subplot(1,3,3)
estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
title("\alpha = 10")

function estimate_ulmmse(ux, Cx, x0, theta, sd_theta)
    %%%%%%%%%%%%%%%%%%%%
```

```
% draw uncertainty

% eigenvalue of Cx
[eig_vec, eig_vals] = eig(Cx);
disp("EigenValues of Cx = ")
disp(eig_vals)
disp("EigenVectors of Cx = ")
disp(eig_vec)

% unit circle
th = 0:pi/50:2*pi;
xelp = cos(th);
yelp = sin(th);

% scale circle accordingly
scale_x = sqrt(eig_vals(1,1));
scale_y = sqrt(eig_vals(2,2));
xelp = scale_x*xelp;
yelp = scale_y*yelp;

% rotate elipse
pts = eig_vec * [xelp; yelp];
xelp = pts(1,:) + ux(1);
yelp = pts(2,:) + ux(2);

% plot elipse
plot(xelp, yelp)
hold on
plot(ux(1), ux(2), '*')
text(ux(1), ux(2), "\mu_{x}",...
    'HorizontalAlignment','left',...
    'VerticalAlignment','top',...
    'FontSize',12)
line([ux(1) 3*eig_vec(1,1) + ux(1)], [ux(2) 3*eig_vec(2,1) + ux(2)])
line([ux(1) 3*eig_vec(1,2) + ux(1)], [ux(2) 3*eig_vec(2,2) + ux(2)])
xlabel('x')
ylabel('y')
title("Ship state estimation")
axis equal
```

```

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

% add  line of sight

% plot principal line and corner lines
line([ux(1) ux(1)+150*cos(theta)],...
    [ux(2) ux(2)+150*sin(theta)])
line([ux(1) ux(1)+150*cos(theta+sd_theta)],...
    [ux(2) ux(2)+150*sin(theta+sd_theta)],...
    'Color','red','LineStyle','--')
line([ux(1) ux(1)+150*cos(theta-sd_theta)],...
    [ux(2) ux(2)+150*sin(theta-sd_theta)],...
    'Color','red','LineStyle','--')

% plot lighthouse real position x0
plot(x0(1), x0(2), '*')
text(x0(1), x0(2), 'x_0',...
    'HorizontalAlignment','left',...
    'VerticalAlignment','baseline',...
    'FontSize',12)

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

% plot linearized uncertainty bar region of the line of sight

% two parallel lines width
% first calculate the distance
d = norm(ux-x0);
sd_v = d*sd_theta;

% difference between the center line and the top and bottom lines in y axis
y_diff = sd_v/cos(theta);

% plot parallels lines in green
line([ux(1) ux(1)+150*cos(theta)],...
    [ux(2)+y_diff ux(2)+150*sin(theta)+y_diff],...
    'Color','green','LineStyle','--')
line([ux(1) ux(1)+150*cos(theta)],...
```

```
        [ux(2)-y_diff ux(2)+150*sin(theta)-y_diff],...
        'Color','green','LineStyle','--')

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

    % ulMMSE estimation

    % kalman measurement matrix: H
    H = [sin(theta) -cos(theta)];
    disp("Kalman gain matrix: H = ")
    disp(H)

    % derived measurement: z
    z = x0(1)*sin(theta) - x0(2)*cos(theta);
    disp("Derived measurement: z = ")
    disp(z)

    % kalman gain matrix: K
    Cv = sd_v^2;
    K = Cx*H'/(H*Cx*H'+Cv);
    disp("Kalman Gain matrix: K = ")
    disp(K)

    % perform estimation with ulMMSE
    x_hat = ux +K*(z-H*ux);
    disp("Estimated state with ulMMSE")
    disp(x_hat)

    % plot estimated state
    plot(x_hat(1), x_hat(2), "*")
    text(x_hat(1), x_hat(2), '$$x_{ulMMSE}$$',...
        'Interpreter','Latex',...
        'HorizontalAlignment','left',...
        'VerticalAlignment','bottom',...
        'FontSize',12)

    % calculate covariance Cxz
    Cx_z = inv(inv(Cx) + H'/Cv*H);
    disp("Posterior covariance: Cx_z = ")
```

```
    disp(Cx_z)

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

    % draw posterior uncertainty

    % eigenvalue of Cxz
    [eig_vec, eig_vals] = eig(Cx_z);
    disp("EigenValues of Cxz = ")
    disp(eig_vals)
    disp("EigenVectors of Cxz = ")
    disp(eig_vec)

    % unit circle
    th = 0:pi/50:2*pi;
    xelp = cos(th);
    yelp = sin(th);

    % scale circle accordingly
    scale_x = sqrt(eig_vals(1,1));
    scale_y = sqrt(eig_vals(2,2));
    xelp = scale_x*xelp;
    yelp = scale_y*yelp;

    % rotate elipse
    pts = eig_vec * [xelp; yelp];
    xelp = pts(1,:) + x_hat(1);
    yelp = pts(2,:) + x_hat(2);

    % plot elipse
    plot(xelp, yelp, 'b')

end
```

Listing 3.1: Source code for exercise 3

4 Exercise 4

On this exercise, we analyzed the process of prediction and how the covariance is increased during it. The state space model of the system was useful to infer what is the behavior of the system before it happened. In this scope, it is possible to analyze that the main reason to the increase of the covariance along the prediction steps is the process noise. The covariance matrix of the process noise contributes to the increase the terms of the state covariance matrix. Fig. 4.1 shows the predicted path and the ellipses defined by the covariance matrix for a prediction started in the step i=11 until i=101. Fig. 4.2 show the results of changing start step for prediction.

It is possible to analyze that the ellipses get larger with the prediction steps. This behavior is due by the accumulated effect of the process noise. Caused by some disturbance in the system, misalignment in model parameters or model description. So, less prediction steps is better for a system that need a good estimate of each state.

```

% exercise 4 optimal estimation of dynamic systems
% author: Guilherme Soares Silvestre

clear
close all

% create states vector
% load position: (2, N) matrix
load 'data exercise 4'/log.mat
% lenght of xsi
N = length(xsi);

% velocity: (2, N-1) matrix
vsi = zeros(2,N-1);
for i=1:(N-1)
    vsi(:,i) = xsi(:,i+1)-xsi(:,i);
end

```

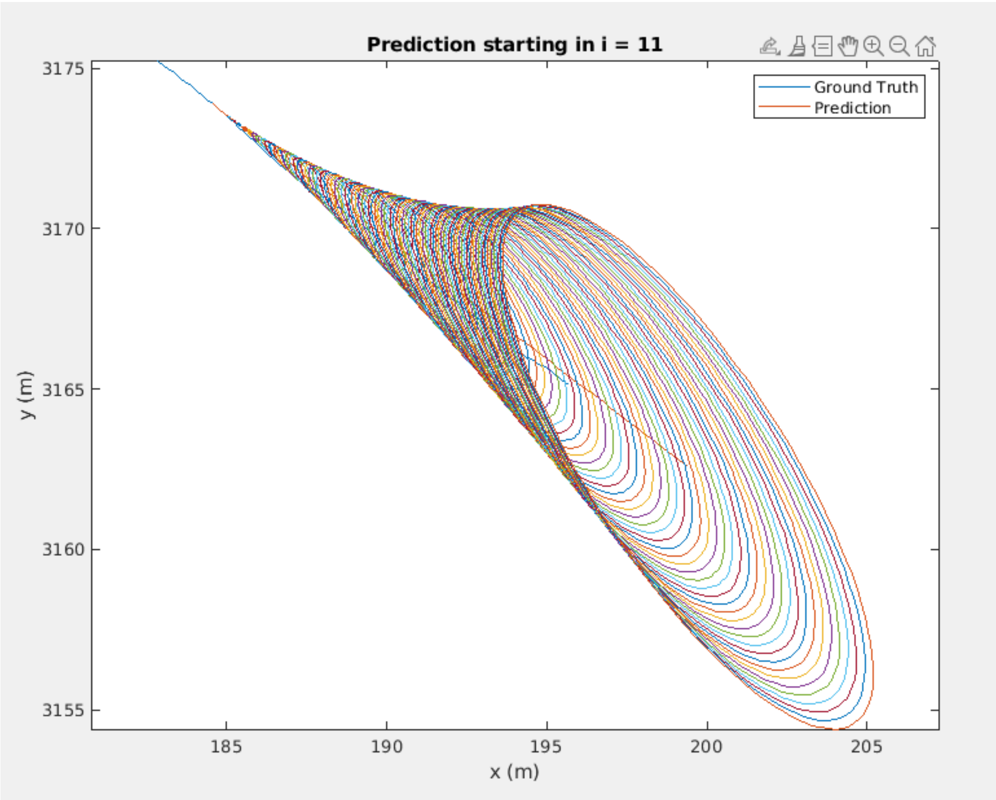

Figure 4.1: Prediction behavior started in step i=11 until step i=101

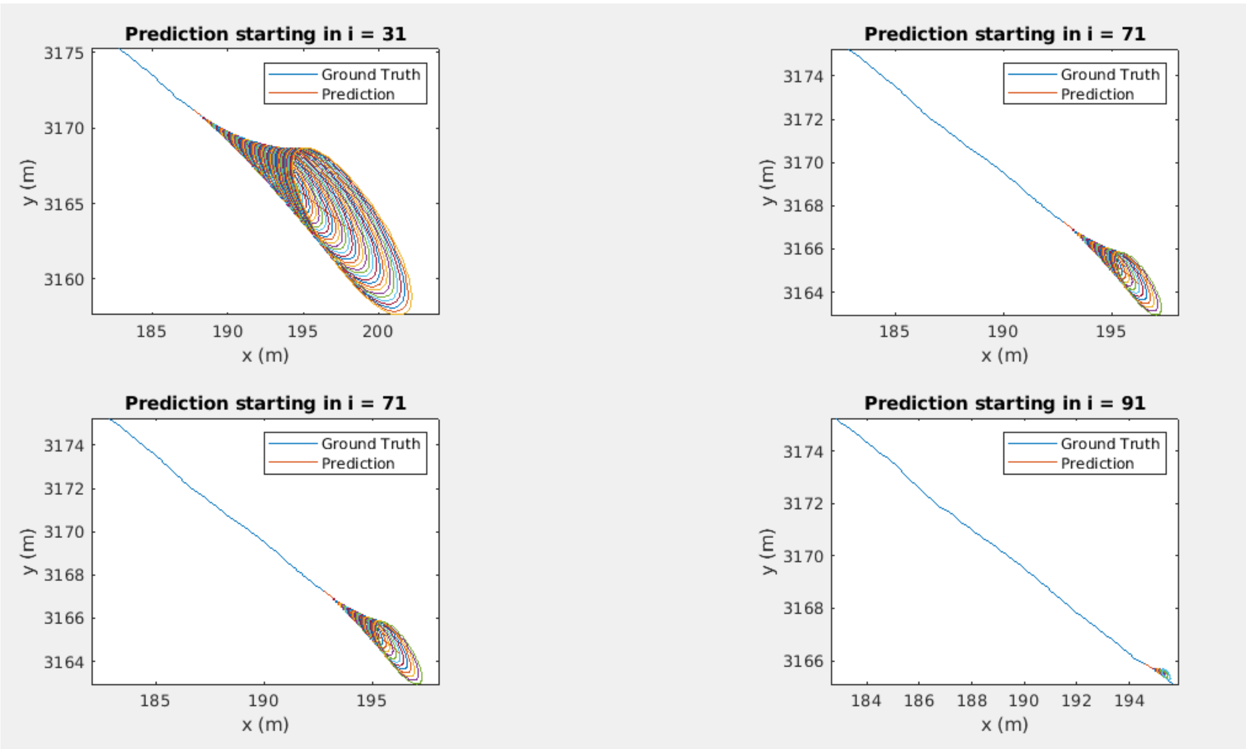

Figure 4.2: Analysis of the effect of changing start step for the prediction process.

```
% acceleration: (2, N-2) matrix
asi = zeros(2,N-2);
for i=1:(N-2)
    asi(:,i) = vsi(:,i+1)-vsi(:,i);
end

% state space matrix
global x
x = [xsi(:,1:N-2); vsi(:,1:N-2); asi(:,1:N-2)];

% given accelration system matrix and covariance
F_1 = [-0.0595 -0.1530;
      -0.813 0.1716];

Cw_1 = [0.1177e-3 -0.0026e-3;
      -0.0026e-3 0.0782e-3];

% intialize calculated system matrix for full state
global F
F = [1 0 1 0 0 0;
    0 1 0 1 0 0;
    0 0 1 0 1 0;
    0 0 0 1 0 1;
    0 0 0 0 F_1(1,1) F_1(1,2);
    0 0 0 0 F_1(2,1) F_1(2,2);];

% Noise covariance matrix only in the acceleration part
global Cw
Cw =  [0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 Cw_1(1,1) Cw_1(1,2);
      0 0 0 0 Cw_1(2,1) Cw_1(2,2)];

% starting point = 11
simulate_prediction(11)

% starting point = 31
```

```
figure
subplot(2,2,1)
simulate_prediction(31)

% starting point = 71
subplot(2,2,2)
simulate_prediction(71)

% starting point = 71
subplot(2,2,3)
simulate_prediction(71)

% starting point = 91
subplot(2,2,4)
simulate_prediction(91)

function simulate_prediction(start)

    global x
    global F
    global Cw

    steps = 101-start;
    x_10 = x(:,start);
    Cx_10 = zeros(6,6);

    % predict
    [x_ahead, Cx_ahead] = predict(F, Cw, Cx_10, x_10, steps);

    % plot prediction
    plot(x(1,:), x(2,:))
    hold on
    plot(x_ahead(1,:), x_ahead(2,:))

    plot_elipses(x_ahead, Cx_ahead)
    xlabel('x (m)')
    ylabel('y (m)')
    axis equal
    t=sprintf("Prediction starting in i = %d", start);
```

```
    title(t)
    legend(["Ground Truth" "Prediction"])
end

function [x_ahead, Cx_ahead] = predict(F, Cw, Cx, x, steps)

    % create state vector for prediction
    size_x = size(x);
    x_ahead = zeros(size_x(1), steps);
    Cx_ahead = zeros(size_x(1), size_x(1), steps);

    % start state and cov matrix with initial states
    x_ahead(:,1) = x;
    Cx_ahead(:,:, 1) = Cx;

    % loop for 'steps' times calculating the state prediction and
    % covariance for each state
    for i = 1:steps
        x_ahead(:,i+1) = F*x_ahead(:,i);
        Cx_ahead(:,:,i+1) = F*Cx_ahead(:,:,i)*F' + Cw;
    end
end

function plot_elipses(x_ahead, Cx_arr)

    % for each cov matrix inside Cx_Arr, calculate the eigen vector
    for index = 1:length(Cx_arr)
        Cx_curr = Cx_arr(1:2,1:2, index);
        [eig_vec, eig_vals] = eig(Cx_curr);

        % unit circle for further use
        th = 0:pi/50:2*pi;
        xelp = cos(th);
        yelp = sin(th);

        % scale circle accordingly
        scale_x = sqrt(eig_vals(1,1));
        scale_y = sqrt(eig_vals(2,2));
        xelp = scale_x*xelp;
```

```
        yelp = scale_y*yelp;

        % rotate elipse
        pts = eig_vec * [xelp; yelp];
        xelp = pts(1,:) + x_ahead(1, index);
        yelp = pts(2,:) + x_ahead(2, index);

        % plot elipse
        plot(xelp, yelp)
    end
end
```

Listing 4.1: Source code for exercise 4

5 Exercise 5

In the fifth exercise it was possible to use all the estimation theory studied before. The application was a localization of 2D dynamic system. The state space equations were used to implement the Discrete Kalman Filter. This filter is applied in linear dynamic systems as it uses linear transformations to achieve the optimal estimation of the system's state. It's is useful for non linear systems that have their non linearity smooth. The benefit of the linear Discrete Kalman Filter is the, beside it's lack of accuracy in non linear systems, it's has the characteristics of non diverging as other filters do (e.g Extended Kalman Filter).

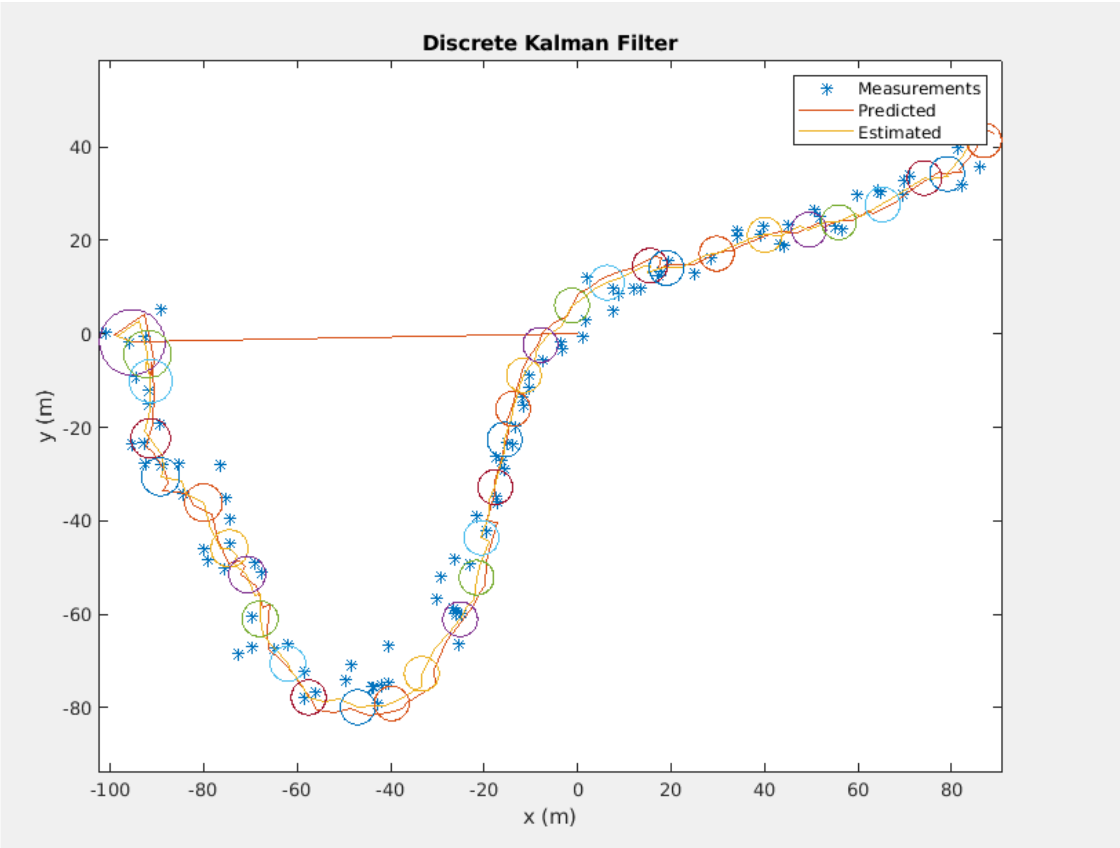

Figure 5.1: Position of a dynamic object estimated with Discrete Kalman Filter with time variant matrix.

Figure 5.1 shows the implementation of the DKF using time variant kalman gain and covariance matrix. Figure 5.2 shows the result of the DKF using steady state matrix calculated by dlqe method in Matlab. The estimator takes longer to converge using the steady state matrix,

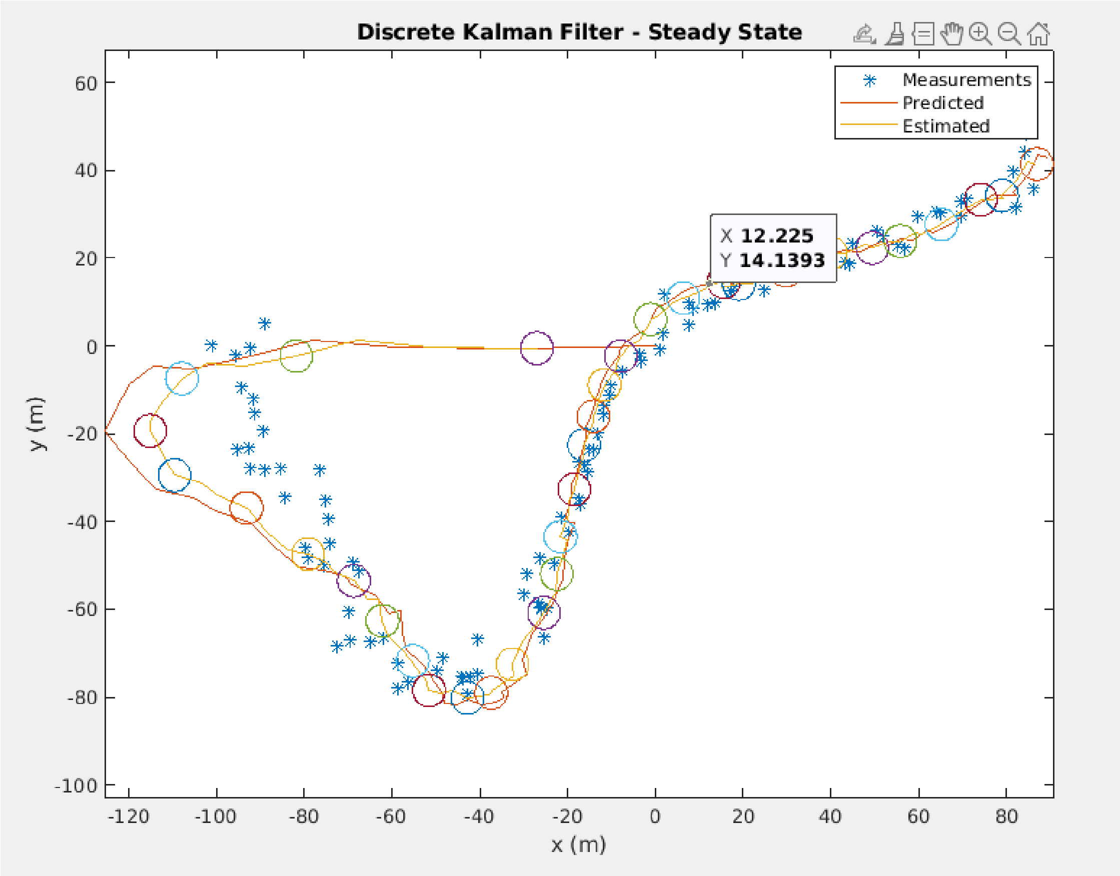

Figure 5.2: Position of a dynamic object estimated with Discrete Kalman Filter with steady state matrix.

but after it the results are almost the same.

```

% exercise 5 optimal estimation of dynamic systems
% author: Guilherme Soares Silvestre

clear
close all

% given accelration system matrix and covariance
F_1 = [0.97 0;
      0 0.97];

% process noise covariance
Cw_1 = [0.0016 0;
       0 0.0016];

% measurement noise covariance
```

```matlab
Cn_1 = [49 0;
       0 49];

% intialize calculated system matrix for full state
global F
F = [1 0 1 0 0 0;
    0 1 0 1 0 0;
    0 0 1 0 1 0;
    0 0 0 1 0 1;
    0 0 0 0 F_1(1,1) F_1(1,2);
    0 0 0 0 F_1(2,1) F_1(2,2);];

% noise covariance matrix only in the acceleration part (proccess noise)
global Cw
Cw =  [0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 0 0;
      0 0 0 0 Cw_1(1,1) Cw_1(1,2);
      0 0 0 0 Cw_1(2,1) Cw_1(2,2)];

% measurement noise
% Cn = [Cn_1(1,1) Cn_1(1,2) 0 0 0 0;
%       Cn_1(2,1) Cn_1(2,2) 0 0 0 0;
%       0 0 Cn_1(1,1) Cn_1(1,2) 0 0;
%       0 0 Cn_1(2,1) Cn_1(2,2) 0 0;
%       0 0 0 0 Cn_1(1,1) Cn_1(1,2);
%       0 0 0 0 Cn_1(2,1) Cn_1(2,2)];
Cn = [Cn_1(1,1) Cn_1(1,2) 0 0 0 0;
      Cn_1(2,1) Cn_1(2,2) 0 0 0 0;
      0 0 1 0 0 0;
      0 0 0 1 0 0;
      0 0 0 0 1 0;
      0 0 0 0 0 1];

% initial covariance
sd_x = 100; %m
sd_v = 4; %m/s
sd_a = 0.2; %m/s^2
```

```
Cx_0 = [sd_x.^2 0 0 0 0 0;
        0 sd_x.^2 0 0 0 0;
        0 0 sd_v.^2 0 0 0;
        0 0 0 sd_v.^2 0 0;
        0 0 0 0 sd_a.^2 0;
        0 0 0 0 0 sd_a.^2];

% initial mean
x_0 = [0;0;0;0;0;0];

% first define measurement matrix
H = [1 0 0 0 0 0;
     0 1 0 0 0 0;
     0 0 0 0 0 0;
     0 0 0 0 0 0;
     0 0 0 0 0 0;
     0 0 0 0 0 0];

% create states vector
% load position: (2, N) matrix
load 'data exercise 5'/zradar.mat

% lenght of xsi
N = length(z);

z_full = zeros(6, N);
z_full(1:2,:) = z;

% discrete kalman filter main loop
x_hat = zeros(6, N);
x_pred = x_hat;

Cx_hat = zeros(6, 6, N);
Cx_pred = Cx_hat;

x_pred(:,1) = x_0;
Cx_pred(:,:,1) = Cx_0;
for i = 1:N

```

```
    % update step

    % innovation
    inov = z_full(:,i) - H*x_pred(:,i);
    % innovation covariance (pre-fit residual)
    S = H*Cx_pred(:,:,i)*H' + Cn;
    % kalman gain
    K = Cx_pred(:,:,i)*H'/S;
    % a posteriori estimation
    x_hat(:,i) = x_pred(:,i) + K*inov;
    % a posteriori covariance
    Cx_hat(:,:,i) = (eye(6)-K*H)*Cx_pred(:,:,i);

    % predict step

    % prior estimate
    x_pred(:,i+1) = F*x_hat(:,i);
    % estimate covariance
    Cx_pred(:,:,i+1) = F*Cx_hat(:,:,i)*F' + Cw;
end

% plot measurements, predicted and estimated position
plot(z(1,:), z(2,:), '*')
hold on
plot(x_pred(1,:), x_pred(2,:))
plot(x_hat(1,:), x_hat(2,:))

% draw covariance ellipses
plot_elipses(x_hat, Cx_hat)

axis equal
xlabel('x (m)')
ylabel('y (m)')
title("Discrete Kalman Filter")
legend(["Measurements" "Predicted" "Estimated"])


% calculating kalman gain and covariances with matlab default method
[K_dlqe,Cx_pred_dlqe,Cx_hat_dlqe] = dlqe(F,eye(6),H,Cw,Cn);
```

```
% rerunning the system with steady state matrix
x_hat_dlqe = zeros(6, N);
x_pred_dlqe = x_hat;

x_pred_dlqe(:,1) = x_0;
for i = 1:N

    % update step

    % innovation
    inov = z_full(:,i) - H*x_pred_dlqe(:,i);
    % a posteriori estimation
    x_hat_dlqe(:,i) = x_pred_dlqe(:,i) + K_dlqe*inov;

    % predict step

    % prior estimate
    x_pred_dlqe(:,i+1) = F*x_hat_dlqe(:,i);
end


% plot measurements, predicted and estimated position for steady state
figure
plot(z(1,:), z(2,:), '*')
hold on
plot(x_pred_dlqe(1,:), x_pred_dlqe(2,:))
plot(x_hat_dlqe(1,:), x_hat_dlqe(2,:))

% draw covariance ellipses
Cx_hat_dlqe_arr = repmat(Cx_hat_dlqe, [6,6,N]);
plot_elipses(x_hat_dlqe, Cx_hat_dlqe_arr)

axis equal
xlabel('x (m)')
ylabel('y (m)')
title("Discrete Kalman Filter - Steady State")
legend(["Measurements" "Predicted" "Estimated"])

```

```
%
% % velocity: (2, N-1) matrix
% z_v = zeros(2,N-1);
% for i=1:(N-1)
%     z_v(:,i) = z(:,i+1)-z(:,i);
% end
%
% % acceleration: (2, N-2) matrix
% z_a = zeros(2,N-2);
% for i=1:(N-2)
%     z_a(:,i) = z_v(:,i+1)-z_v(:,i);
% end
%
% % state space matrix
% global x
% x = [xsi(:,1:N-2); vsi(:,1:N-2); asi(:,1:N-2)];


function plot_elipses(x_arr, Cx_arr)

    % for each cov matrix inside Cx_Arr, calculate the eigen vector
    for index = 1:3:length(Cx_arr)
        Cx_curr = Cx_arr(1:2,1:2, index);
        [eig_vec, eig_vals] = eig(Cx_curr);

        % unit circle for further use
        th = 0:pi/50:2*pi;
        xelp = cos(th);
        yelp = sin(th);

        % scale circle accordingly
        scale_x = sqrt(eig_vals(1,1));
        scale_y = sqrt(eig_vals(2,2));
        xelp = scale_x*xelp;
        yelp = scale_y*yelp;

        % rotate elipse
        pts = eig_vec * [xelp; yelp];
        xelp = pts(1,:) + x_arr(1, index);
```

```
        yelp = pts(2,:) + x_arr(2, index);

        % plot elipse
        plot(xelp, yelp)
    end
end
```

Listing 5.1: Source code for exercise 5

6 Exercise 6

On the exercise 6, it was possible to understand the Extended Kalman Filter and its implications. The motivation of the Extended Kalman Filter is to build a better filter for non-linear systems. It is done by linearizing the neighborhood of the state in each time. Therefore, the system is not defined by matrix, but by non-linear functions that are always approximated using Taylor Series (applying Jacobian matrix).

The accuracy of the estimator is increased but it was possible to see the randomness of the estimator as it doesn't perform the same way each time that we run. The EKF can also diverges of the real value due to singularities and not modeled behaviors. Figure 6.1 shows the estimated states of the dynamic system of the boat with the EKF estimator. Figure **??** shows the path performed by the boat estimated using the EKF estimator.

As with the others estimators, this case of the EKF application converged in few step and even with low update steps it was able to perform the prediction without increasing so much the covariance. When the updates occurs, the states of the system was corrected and it is possible to see gaps between the states. Thus, the study of the family of estimators was a good experience and necessary path for a robotics engineer.

```

% exercise 6 optimal estimation of dynamic systems
% author: Guilherme Soares Silvestre

clear
close all

% extended kalman filter: the yatch case

% load measurements
load 'data exercise 6'/z_yacht.mat

% % load usefull functions
% load 'data exercise 6'/Fjacobian.m
```

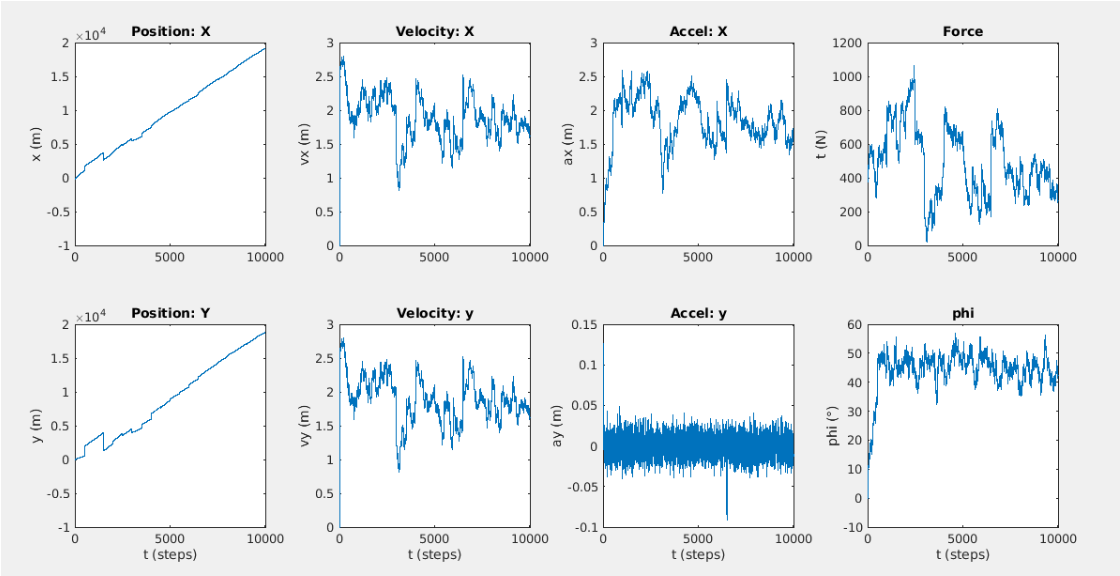

Figure 6.1: Position, Velocity, Acceleration, Force and Angle estimated using the EKF of the boat.

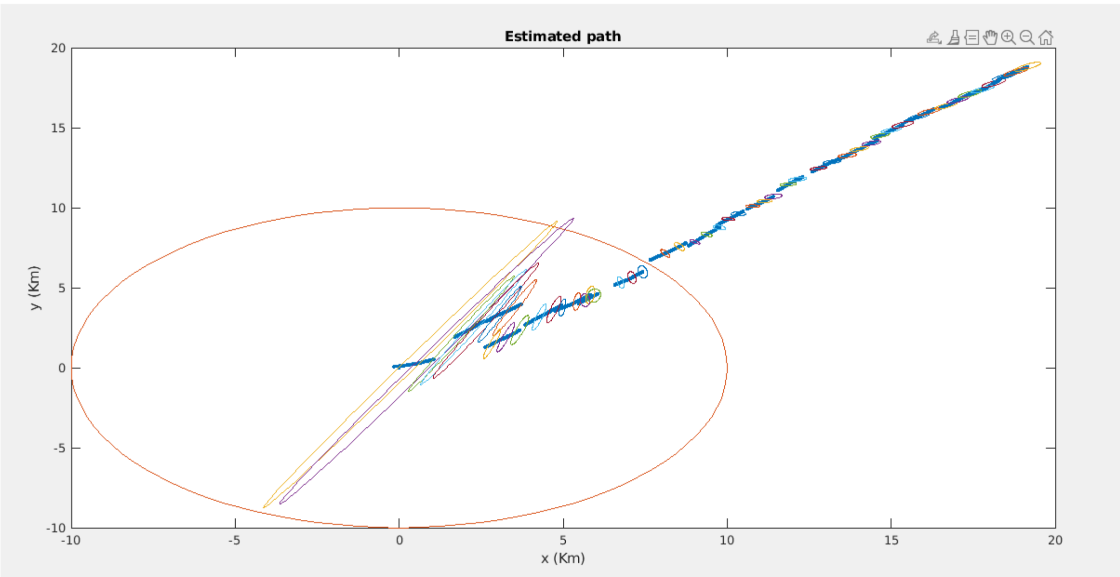

Figure 6.2: Path performed by the boat and ellipses given by the covariance matrix.

```
% load 'data exercise 6'/fsys.m
% load 'data exercise 6'/Hjacobian.m
% load 'data exercise 6'/hmeas.m

% build the process noise covariance matrix
sd_w_x = 0.01; % m
sd_w_v = 0.01; % m
sd_w_a = 0.01; % m
sd_w_t = 8; % N
sd_w_phi = 0.5; % deg

Cw = eye(8).*[sd_w_x.^2;
              sd_w_x.^2;
              sd_w_v.^2;
              sd_w_v.^2;
              sd_w_a.^2;
              sd_w_a.^2;
              sd_w_t.^2;
              sd_w_phi.^2];

% build the measurement noise covariance matrix
sd_compass = 1; % deg
sd_speed = 0.3; % m/s
sd_heading = 1; % deg

Cv = [sd_compass.^2 0 0;
      0 sd_speed.^2 0;
      0 0 sd_heading.^2];

% beacon position
x0 = [5000 10000]';

% desired thrust and heading
t_des = 400; % N (t0)
phi_des = 45; % deg (phi0)
u = [t_des phi_des]';

% initial prior pose
x_hat_0 = [0 0 0 0 0 0 400 0]';
```

```
% build the initial covariance matrix
sd_x_0 = 10000; % m
sd_v_0 = 2; % m/s
sd_a_0 = 0.04; % m/s
sd_t_0 = 300; % N
sd_phi_0 = 10; % deg

Cx_0 = eye(8).*[sd_x_0.^2;
              sd_x_0.^2;
              sd_v_0.^2;
              sd_v_0.^2;
              sd_a_0.^2;
              sd_a_0.^2;
              sd_t_0.^2;
              sd_phi_0.^2];

% main loop for extended kalman filter
N = 1000;

% initialize arrays
x_hat = zeros(8, N);
Cx_hat = zeros(8,8,N);

x_hat(:,1) = x_hat_0;
Cx_hat(:,:,1) = Cx_0;
for i=1:10000
    % update
    if any(imeas(:)==i)
        imeas_idx = find(imeas==i);
        Cx = Cx_hat(:,:,i);
        x = x_hat(:,i);
        H = Hjacobian(x_hat(:,i), x0);
        z_curr = z(:,imeas_idx);

        % estimated measurement
        z_hat = hmeas(x, x0, Cv);

        % innovation
```

```
        S = H*Cx*H' + Cv;

        % Kalman Gain
        K = Cx*H'/S;

        % update posterior mean
        x_hat(:,i+1) = x + K*(z_curr-z_hat);

        % update posterior covariance
        Cx_hat(:,:,i+1) = Cx - K*S*K';
    else

        % predict
        x_hat(:,i+1) = fsys(x_hat(:,i), u, Cw);
        F = Fjacobian(x_hat(:,i));
        Cx_hat(:,:,i+1) = F*Cx_hat(:,:,i)*F' + Cw;
    end
end

% plot estimates
t = linspace(0,10001,10001);
figure

% plot x
subplot(2,4,1)
plot(t, x_hat(1,:))
title("Position: X")
ylim([-10000 20000])
ylabel("x (m)")

% plot y
subplot(2,4,5)
plot(t, x_hat(2,:))
title("Position: Y")
ylim([-10000 20000])
ylabel("y (m)")
xlabel("t (steps)")

% plot velocity x
```

```
subplot(2,4,2)
plot(t, x_hat(3,:))
title("Velocity: X")
ylabel("vx (m)")

% plot velocity y
subplot(2,4,6)
plot(t, x_hat(3,:))
title("Velocity: y")
ylabel("vy (m)")
xlabel("t (steps)")

% plot accel x
subplot(2,4,3)
plot(t, x_hat(4,:))
title("Accel: X")
ylabel("ax (m)")

% plot accel y
subplot(2,4,7)
plot(t, x_hat(5,:))
title("Accel: y")
ylabel("ay (m)")
xlabel("t (steps)")

% plot force
subplot(2,4,4)
plot(t, x_hat(7,:))
title("Force")
ylabel("t (N)")

% plot phi
subplot(2,4,8)
plot(t, x_hat(8,:))
title("phi")
ylabel("phi (  )")
xlabel("t (steps)")

% plot path
```

```
figure
plot(x_hat(1,:)./1000, x_hat(2,:)./1000, ".")
hold on
plot_elipses(x_hat./1000, Cx_hat./1000^2)
xlim([-10 20])
ylim([-10 20])
xlabel("x (Km)")
ylabel("y (Km)")
title("Estimated path")

function plot_elipses(x_arr, Cx_arr)

    % for each cov matrix inside Cx_Arr, calculate the eigen vector
    for index = 1:200:length(Cx_arr)
        Cx_curr = Cx_arr(1:2,1:2, index);
        [eig_vec, eig_vals] = eig(Cx_curr);

        % unit circle for further use
        th = 0:pi/50:2*pi;
        xelp = cos(th);
        yelp = sin(th);

        % scale circle accordingly
        scale_x = sqrt(eig_vals(1,1));
        scale_y = sqrt(eig_vals(2,2));
        xelp = scale_x*xelp;
        yelp = scale_y*yelp;

        % rotate elipse
        pts = eig_vec * [xelp; yelp];
        xelp = pts(1,:) + x_arr(1, index);
        yelp = pts(2,:) + x_arr(2, index);

        % plot elipse
        plot(xelp, yelp)
    end
end
```

Listing 6.1: Source code for exercise 5

7 Conclusion

This report summarizes the results achieved in the Optimal Estimation in Dynamic Systems course. All the code is available in the github repository including this report: `https://github.com/guisoares9/optimal-estimation-studies`. The study from Bayes formula, Markov assumptions to EKF and particle filtering was a good experience that surely will be used in next projects.