Report

Laboratory Advanced Control Engineering (BB MECH-M-1-RTV-RTV-LB)

Lab 1 - Aerodynamic Levitation

13.01.2024 and 02.02.2024

Master Mechatronics & Smart Technologies

1. Semester

Lector: Jasper Volmer

Group: MA-MECH-23-BB-2B

Authors: Lukas Figl, Lukas Sieß, Stefan Widmann

May 24, 2024

Contents

1 Introduction

The corresponding laboratory *Aerodynamic Levitation* to the course *Advanced Control Engineering 1* gives students the opportunity to model a physical system using system identification method, create the according state space model, implement a state observer, design a state-feedback controller and implement and simulate everything with MATLAB and Simulink Coder. [1]

The aerodynamic levitation system (figure 1.1) consists of a table tennis ball, a transparent hollow cylinder with a blower fan and an infrared distance sensor on the top. By controlling the speed of the blower fan, the height of the table tennis ball can be specified. Additionally, a National Instruments data acquisition card as well as a fan speed control system in combination with laboratory power supplies are used for this laboratory. [1]

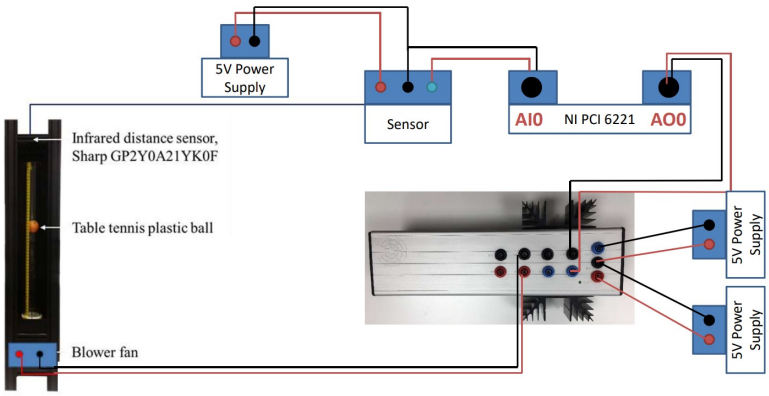

Figure 1.1: Aerodynamik levitation system setup [1]

The lab preparation for the first session is to fit the sensor output based on the data given on Sakai (see figure A.1 in the appendix A.1 on page VI). So we get a function which computes the distance to the reflective object given in voltage to create a Simulink block which implements this conversion. Also the theory of a second-order system needs to be listed, especially the standard form in Laplace-domain, the steps required to derive the step response in the time-domain and the equation for the time-domain response for underdamped, critically damped and overdamped systems.

In the first lab session the distance sensor gets calibrated, which means we have to take care for the signal offset as well as for the nonlinear behaviour of the sensor. To get rid of the signal offset we simply substract a contstant value from the sensor signal to get an output voltage of $0\,\mathrm{V}$ when the table tennis ball was at the bottom of the cylinder. To linearize the sensor output, nine datapoints of the sensor output voltage against the distance x of the table tennis ball from the sensor are taken. The values can be found in the table B.1 on page IX in the appendix B.1. A plot of the datapoints as well as the corresponding linear function

$$f(x) = -0.0315 \cdot x + 2.4758888 \tag{1}$$

is also shown in figure B.1 on page IX in the appendix B.1. Also during the first lab session the system identification part takes place. Therefore some step responses of the system behaviour are taken to trimm out some nice measurement data. The raw measurement data as well as the trimmed step response of the system can be seen in subfigure B.2a and B.2b on page X in the appendix B.2. Further a normalization of the system response was necessary to fit a PT2 System into the signal. The normalized signal as well as the fitted PT2 system can be seen in subfigure B.2c and B.2d on page X in the appendix B.2.

The lab preparation for the second session consists of first creating a simulink state-space model of the PT2 system with the step response. Then an observer together with a state feedback controller can be designed in simulink with pole placement method. The whole system in simulink can be seen in figure A.2 in appendix A.2 on page VI. The subsystems of the **Model**, the **Controller** as well as the **Observer** are shown in figure A.3, A.4 and A.5 in appendix A.2. In the end a step response and

a challenging trajectory of the whole control system is taken from simulink. The corresponding plots are shown in figure A.6 in the appendix A.2 on page VIII.

During the second and last lab session the controller is implemented on the real-world system. After the adjustments some trajectories like a sine wave are tested (appendix B.3 on page XI).

2 System identification

System identification in control engineering is the process of building mathematical models that represent the behavior of real-world systems with analyzing input-output data to infer the dynamics and parameters of the system. We use the data-driven approache which rely on techniques to extract information from measured data. Here the methods like least squares estimation takes place with MATLAB's *lsqcurvefit* function. Of course the accuracy of the identified model depends a lot on the quality of the measured data.

2.1 PT2 SYSTEM

The system behaviour of our current system is figured out by determining the range of the distance sensor in which the system behaves more or less linear. So we choose a range of 20 to $60\,\mathrm{cm}$ distance between our table tennis ball and our distance sensor. The already mentioned linearization of the distance sensor can be seen in figure B.1 on page IX.

The current system should be identified as a PT2 system, like described in the laboratory instruction. [1] To do so the system is applied by some input signals to get the output data (see figure B.2a on page X). A nice input/output data timeslow after $158\,\mathrm{s}$ is chosen to trim the desired system response which is already close to a PT2 system (figure B.2b on page X). From here both the input as well as the ouput signal needs to be normalized to 1. So we divide the input signal by 0.43 (*max(data.stepTrimmed.y)*) and the output signal by the mean value after settling time. The corresponding normalized plot can be found in figure B.2c on page X. From the normalized system response, the fitted PT2 transfer function can be won with *lsqcurvefit* and the response of a damped single-degree-of-freedom harmonic oscillator (eqn. 2) which was provided by the lecturer.

$$y = 1 - \frac{e^{-\zeta w_n t}}{\sqrt{1-\zeta^2}} \cdot \sin(w_d \cdot t + tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)) \tag{2}$$

With the *tf2ss* (transfer function to state space) MATLAB function we get the state-space model with the state-space matrices like in equation 3.

$$\begin{aligned} A &= \begin{bmatrix} -2.20 & -1.57 \\ 1 & 0 \end{bmatrix} \\ B &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ C &= \begin{bmatrix} 0 & 1.57 \end{bmatrix} \\ D &= \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{3}$$

With a proper state-space representation the controller design as well as the simulink control structure can be started in section 3.

3 Controller technology

Controller technology in control engineering is about the design and implementation of structures that regulate the behavior of a dynamic systems. Controllers are needed to maintain desired system outputs by adjusting system inputs usually based on feedback signals. Advanced controller technologies like an observer, specifically a state observer or an observer-based controller, is a component used in control systems to estimate the state variables of a system based on its inputs and outputs. It does not directly influence the control action but provides valuable information for feedback control.

For this laboratory the control structure was given by the lecturer to achieve proper results. In figure 3.1 one can see the whole structure with the controller and the observer in gray rectangles as well as the system in state space representation.

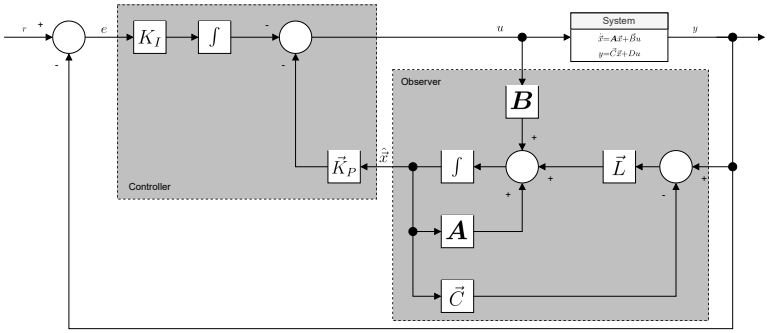

Figure 3.1: Control structure

The control architecture is based on the feedback of the system output y as well as the system states $\hat{\vec{x}}$, which are estimated by the observer. At the sum of the controller block, both the K_i and the K_p signal are substracted. This leads to a higher stability of the control structure because of a higher damping. The used observer technology is called a *Luenberger Observer* which will be described in more detail in chapter 3.2.

3.1 CONTROLLER

For the state feedback control the two parameters settling time $t_s = 4\,\mathrm{s}$ and the percentage overshoot $PO = 5$ are chosen. With equation 4 and 5 the damping ratio $\zeta = 0.69$ and the natural frequency $\omega_n = 1.45$ can be defined.

$$\zeta = \sqrt{\frac{\log(PO)^2}{\pi^2 + \log(PO)^2}} \tag{4}$$

$$\omega_n = \frac{4}{\zeta \cdot t_s} \tag{5}$$

With ζ and ω_n two complex conjugated poles can be calculated with $p_2 = -1.00 + 1.05i$ and $p_3 = -1.00 - 1.05i$. The first one will be not so dominant and therefore more left $p_1 = p_2 \cdot 2 = -2.00 + 2.10i$. Because of no permanent control error due to a step input we get two new matrices like in equation 6. [2]

$$\begin{aligned} A' &= \begin{bmatrix} 0 & 0 & -1.57 \\ 0 & -2.20 & -1.57 \\ 0 & 1 & 0 \end{bmatrix} \\ B' &= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \end{aligned} \tag{6}$$

The control parameters $K_i = -3.67$ and $K_p = \begin{bmatrix} 1.80 & 4.53 \end{bmatrix}$ are then calculated by pole placement method place($A', B', \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}$) in MATLAB.

3.2 OBSERVER

The Luenberger observer estimates the state(s) of the system. It may contain the controlled variable, the derivative of the controlled variable or other internal variables of the system. To achive this the Luenberger observer uses a model of the real system to create an estimation. In our case the observer uses the measurement of the controlled system output y and the control signal u to estimate the state of the system. To design a proper Luenberger observe we simply define two poles at $p_1 = -4$ and $p_2 = -6$. Afterwards with the transposed system matrix $\boldsymbol{A}'$ and output matrix $\boldsymbol{C}'$ the *place* method in MATLAB creates the observe gain matrix $\boldsymbol{L} = place(\boldsymbol{A}', \boldsymbol{C}', \begin{bmatrix} p_1 & p_2 \end{bmatrix})$ whicht leads to equation 7.

$$\boldsymbol{L} = \begin{bmatrix} 3.36 \\ 4.96 \end{bmatrix} \tag{7}$$

4 Simulation

In preparation for the laboratory exercise, the system was modeled, and a controller and observer were implemented in Simulink using the provided measurement data. The methodology involved system identification, as detailed in chapter 2, where parameters for the State Space Model were computed from the measurement data. These identified parameters were subsequently integrated into the Simulink model (figure A.2) utilizing the "Statespace" block (figure A.3).

For the controller, as outlined in chapter 3.1, a State Feedback Control approach was employed, with predefined parameters for percentage overshoot and settling time to ensure precise system control. Additionally, a Luenberger Observer was designed for the modeled system, incorporating considerations of two poles.

In Simulink, the system (figure A.3), the controller (figure A.4), and the observer (figure A.5) were configured as distinct subsystems. This organizational structure not only facilitated better comprehension but also allowed for the facile exchange of the modeled system with input and output blocks for real-world applications.

These simulation and modeling procedures established a methodological framework for the laboratory exercise, as expounded in chapter 2 and chapter 3.1. The Simulink model, illustrated in figure A.2, figure A.3, figure A.4, and figure A.5, served as a comprehensive visual representation of the interconnected components, providing insights into the intricacies of system dynamics. Detailed discussions on system identification, controller design, and observer implementation are available in the referenced chapters, forming the scientific foundation for the conducted simulations and subsequent analyses.

5 Results and interpretation

To assess the modeled system, it underwent testing with both a step input and a trajectory, in our case, characterized by an arbitrarily chosen rectangular profile.

Upon application of the step input, the controller regulated the system as designed, exhibiting the expected settling time and percentage overshoot, as depicted in figure A.6a. Additional measurements from the observer and the system are presented in figure A.6c and A.6e, respectively.

Unfortunately, the trajectory signal was selected with a slightly accelerated pace, leading to amplitude changes after 2 seconds, beyond the response capability of the controller. Nonetheless, as illustrated in figure A.6b, it is evident that the controller lags slightly behind the trajectory. Further measurements from the observer and the system are provided in figure A.6d and A.6f.

The observed results from both the step input and trajectory testing provide valuable insights into the system's response dynamics under different inputs. These findings contribute to a comprehensive understanding of the controller's performance and the system's behavior, facilitating further refinement and optimization in subsequent iterations of the control design. Detailed visual representations and quantitative data for each scenario are available in the respective figures.

6 Controlling the plant

In this laboratory experiment, the control system was implemented on the physical structure that had previously been designed and discussed in detail in the preparation phase (see chapter 3.1). After a series of optimizations and code adaptations, initial results were achieved after several test runs.

As already known, the tests showed that the sensor only exhibits linearity in a certain range and only delivers linear measured values in this interval. Consequently, controlling the height of the table tennis ball in this middle area of the column led to the best results.

Furthermore, ensuring accurate conversion of the sensor readings was crucial, which was achieved by using amplification in Simulink, as shown in figure B.4. This enabled accurate interpretation of the sensor data.

By conducting iterative tests and making adjustments to parameters such as settling time, percentage overshoot, system poles, and observer poles, satisfactory outcomes were attained. Various input stimuli were employed during the experimentation, including a step input aimed at stabilizing the system at a constant height, as well as a custom trajectory input. Additionally, it was observed that careful consideration of system dynamics and sensor characteristics was essential for achieving optimal performance. This involved fine-tuning control parameters and implementing robust control strategies to mitigate disturbances and uncertainties in the system.

The finalized and optimized MATLAB and MATLAB Simulink files associated with this experiment are provided in the appendix, serving as resources for future research and experimentation in the field of control systems engineering, especially for aerodynamic levitation.

7 Results

With meticulous preparation and thorough analysis, significant advancements can be achieved within the confines of this laboratory setting.

Optimization of the PT2 system to the desired specifications was facilitated by strategic adjustments to the positions of both the system poles and observer poles. Particularly noteworthy is the imperative to ensure sensor linearity within a specific operational range, as this significantly influences the attainment of favorable outcomes. A discernible trend observed in the recorded data is the challenge encountered by the controller in maintaining system stability beyond the linearized range of the sensor.

Furthermore, critical considerations were directed towards the setting time and percentage overshoot parameters of the system. These parameters play a pivotal role in expediting system response and controlling overshoot magnitude. It is imperative to strike a harmonious balance among all control parameters to uphold system stability and prevent undesirable phenomena such as oscillations.

In addition to the aforementioned parameters, the impact of disturbances and uncertainties on system performance cannot be overstated. Robust control strategies must be employed to mitigate the effects of external disturbances and uncertainties, ensuring the system's resilience and adaptability in real-world scenarios.

The comprehensive analysis of control strategies and sensor linearization outcomes is encapsulated within the supplementary materials provided in the appendix, serving as a reference for further research and experimentation in the area of control systems engineering.

List of Figures

List of Tables

References

[1] J. Volmer, "Laboratory instructions," *Advanced Control Engineering*, 2022.

[2] A. Mehrle, "State space systems," 2021.

A Preparation

A.1 FIRST PREPARATION

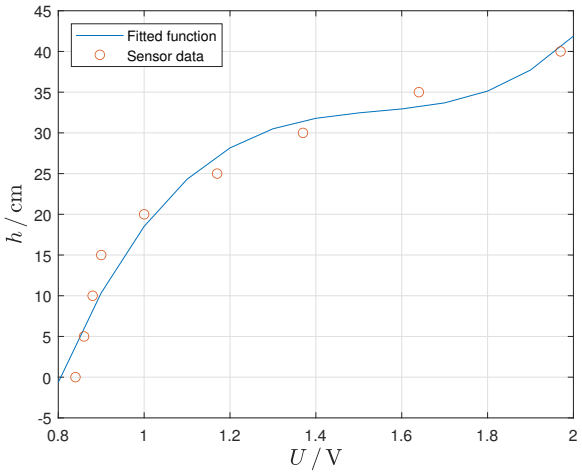

Figure A.1: Fitted function $h(u) = (73.36 \cdot u^3) + (-339.13 \cdot u^2) + (527.23 \cdot u) + (-242.94)$

A.2 SECOND PREPARATION

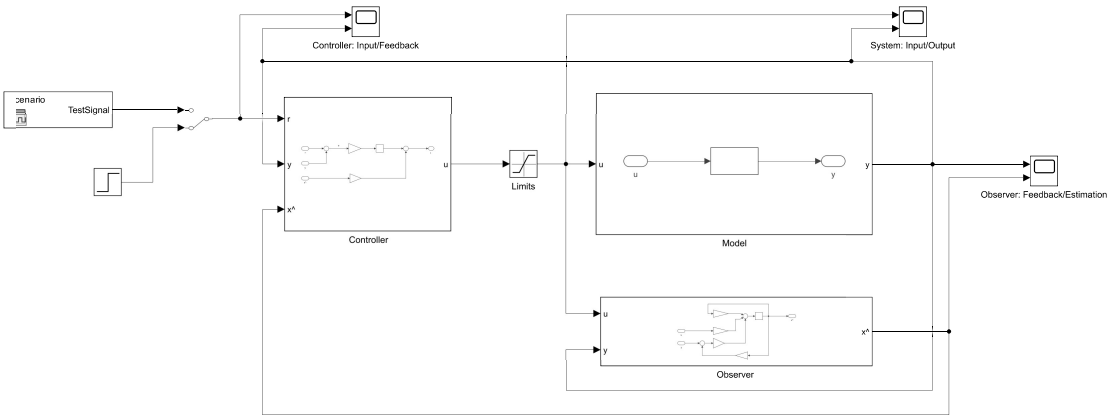

Figure A.2: Simulink simulation of the control system

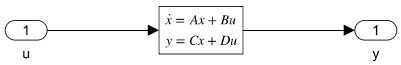

Figure A.3: Simulink model of the system

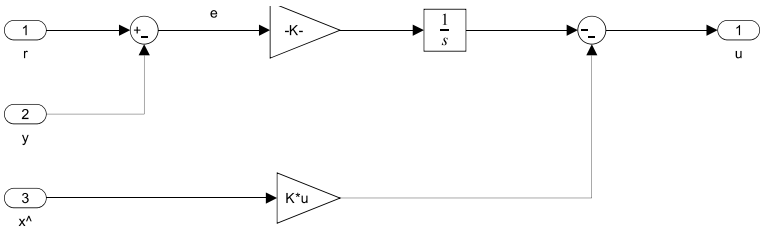

Figure A.4: Simulink controller

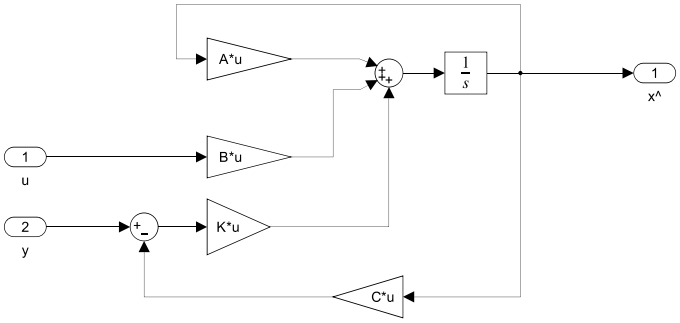

Figure A.5: Simulink observer

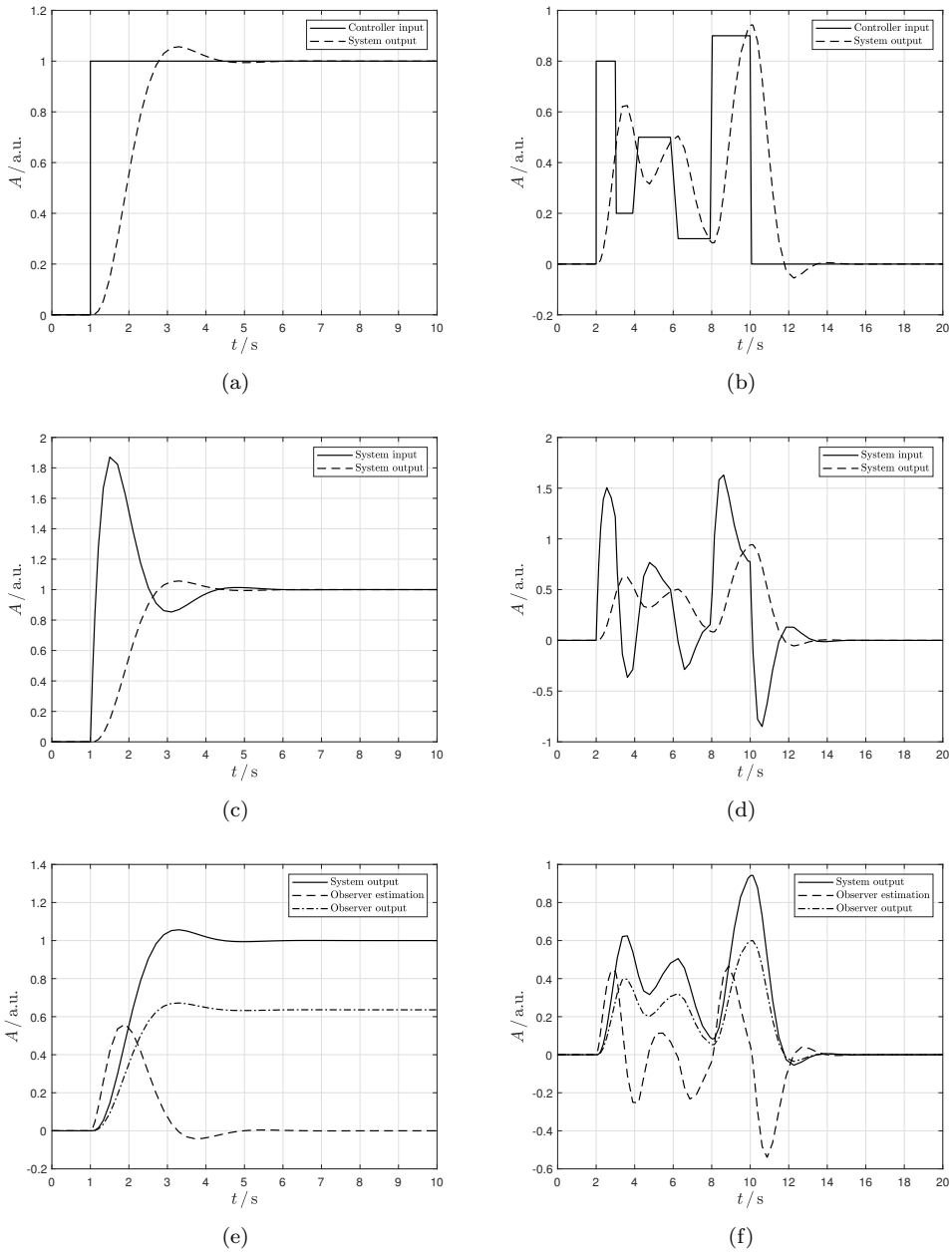

Figure A.6: Left for step input with (a) controller (c) system and (e) observer. Right trajectory input with (b) controller (d) system and (f) observer.

B Laboratory

B.1 SENSOR DATA

Table B.1: Sensor output voltage against distance

Distance	Sensor voltage
18	1.93
23	1.91
28	1.68
33	1.32
38	1.11
43	0.96
48	0.90
53	0.88
58	0.82

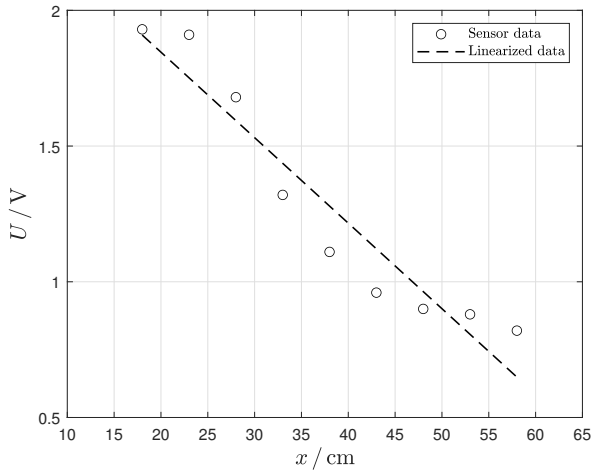

Figure B.1: Sensor linearization: $f(x) = -0.0315 \cdot x + 2.4758888$

B.2 SYSTEM IDENTIFICATION

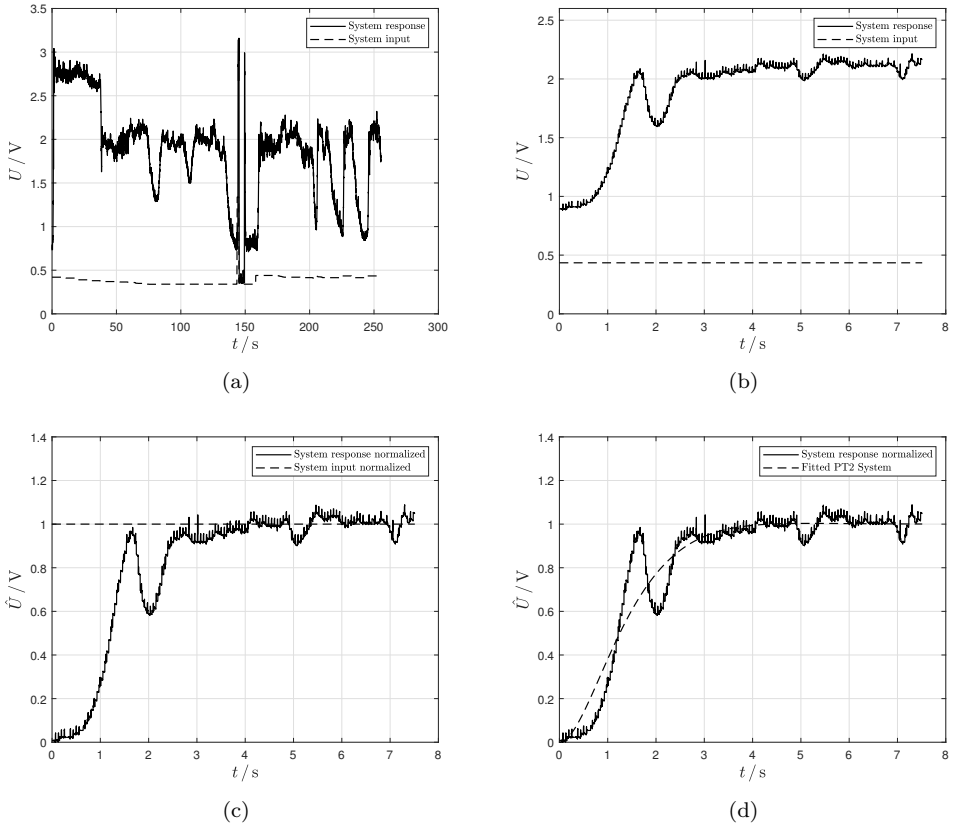

Figure B.2: (a) Raw data (b) Trimmed (c) Normalized (d) Fitted PT2 system

B.3 CONTROLLER PERFORMANCE

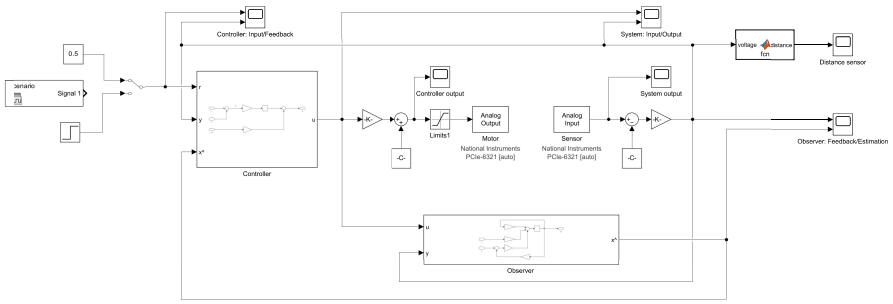

Figure B.3: Simulink laboratory application

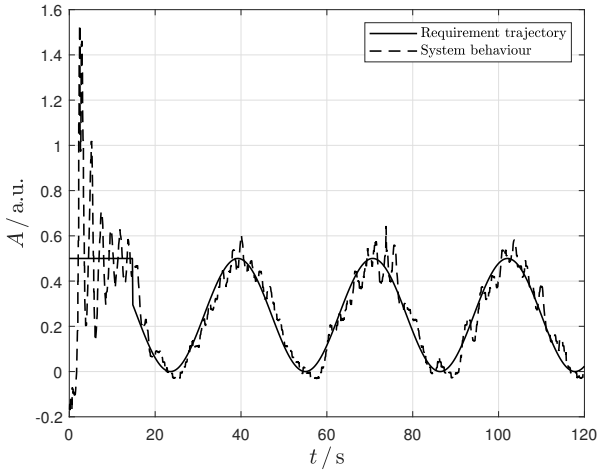

Figure B.4: System behaviour with sine wave trajectory

C MATLAB

C.1 FIRST LABORATORY

```
%% Clean up
clear all
close all
clc


%% Raw data
measurement = importdata('measure.mat')
data.ist.t  = measurement.v_sensor.time;
data.ist.y  = measurement.v_sensor.data;
data.step.t = measurement.v_fan.time;
data.step.y = measurement.v_fan.data;


%% Trimming data
idx_1 = 122244;
idx_2 = 126000;
dt = 2e-3;
data.istTrimmed.t = [0:idx_2-idx_1]'*dt;
data.istTrimmed.y = data.ist.y(idx_1:idx_2);
data.stepTrimmed.t = [0:idx_2-idx_1]'*dt;
data.stepTrimmed.y = data.step.y(idx_1:idx_2);


%% Normalize data
% Offset
data.istNorm.y = data.istTrimmed.y-min(data.istTrimmed.y);
% Normalize
idx_1 = 1876;
idx_2 = 3757;
data.istNorm.y = data.istNorm.y/mean(data.istNorm.y(idx_1:idx_2));
data.istNorm.t = data.istTrimmed.t;
data.stepNorm.y = data.stepTrimmed.y/max(data.stepTrimmed.y);
data.stepNorm.t = data.stepTrimmed.t;


%% Fitting data
data.fitting.Para.a=1; %k
data.fitting.Para.b=2; %w_n
data.fitting.Para.c=0.8; %zeta
data.fitting.coeff = lsqcurvefit(
    @unit_step_PT2,
    [data.fitting.Para.a data.fitting.Para.b data.fitting.Para.c],
    data.istNorm.t, data.istNorm.y);


%% State space model
[A B C D] = tf2ss(
    data.fitting.coeff(2)^2,
    [1
     2*data.fitting.coeff(3)*data.fitting.coeff(2)
     data.fitting.coeff(2)^2]);


%% Pole placement for state feedback
data.pp.ss.ts = 3; % in s
data.pp.ss.ps = 10/100; % in %
```

```
% Calculation of zeta
data.pp.ss.zeta = sqrt(log(data.pp.ss.ps)^2/(pi^2+log(data.pp.ss.ps)^2));

% Calculation of natural frequency
data.pp.ss.w_n = 4/(data.pp.ss.zeta*data.pp.ss.ts);

% Calcuate the pole regions complex conjugated:
data.pp.ss.p2 =
    -data.pp.ss.zeta*data.pp.ss.w_n +
    1i*data.pp.ss.w_n*sqrt(1-data.pp.ss.zeta^2);
data.pp.ss.p3 =
    -data.pp.ss.zeta*data.pp.ss.w_n -
    1i*data.pp.ss.w_n*sqrt(1-data.pp.ss.zeta^2);

% Add 3th pole which is nondominant ==> so more left
data.pp.ss.p1 = real(data.pp.ss.p2) * 2;
data.pp.ss.pl = [data.pp.ss.p1;data.pp.ss.p2;data.pp.ss.p3]

% Calculate new matrices:
% No error according due to step input see script ex. 3.4
A_strich = [0,-C;
            0, A(1,1),A(1,2);
            0 ,A(2,1),A(2,2)];

B_strich = [0;B];

% Poleplacement:
data.pp.ss.K = place(A_strich,B_strich,data.pp.ss.pl);
data.pp.ss.Ki = data.pp.ss.K(1);
data.pp.ss.Kp = data.pp.ss.K(2:end);


%% Pole placement for obeserver gain
data.pp.ss.os = [-5; -6];
% Poleplacement
data.pp.ss.L = place(A',C',data.pp.ss.os);
data.pp.ss.L = data.pp.ss.L';
```

C.1.1 Plots

```
%% RAW DATA
figure('Name', 'Raw Data', 'NumberTitle', 'off');
p1 = plot(
    data.ist.t, data.ist.y, 'k', "Linestyle", '-', 'LineWidth', 1);
axis([0 300 0 3.5]);
grid on;
xlabel('$$t\rm{\,/\,s}$$', 'Interpreter', 'latex','fontsize', 14)
ylabel('$$U\rm{\,/\,V}$$', 'Interpreter','latex','fontsize', 14)

hold on;
p2 = plot(
    data.step.t, data.step.y, 'k', "Linestyle", '--', 'LineWidth', 1);
legend('System response', 'System input', 'Interpreter','latex');


%% TRIMMING
figure('Name', 'Trimming', 'NumberTitle', 'off');
p1 = plot(
    data.istTrimmed.t, data.istTrimmed.y,
    'k', "Linestyle", '-', 'LineWidth', 1);
axis([0 8 0 2.6]);
grid on;
xlabel('$$t\rm{\,/\,s}$$', 'Interpreter', 'latex','fontsize', 14)
```

```
ylabel('$$U\rm{\,/\,V}$$', 'Interpreter','latex','fontsize', 14)

hold on;
p2 = plot(
    data.stepTrimmed.t, data.stepTrimmed.y,
    'k', "Linestyle", '--', 'LineWidth', 1);
legend('System response', 'System input', 'Interpreter','latex');


%% NORMALIZE
figure('Name', 'Normalize', 'NumberTitle', 'off');
p1 = plot(
    data.istNorm.t, data.istNorm.y,
    'k', "Linestyle", '-', 'LineWidth', 1);
axis([0 8 0 1.4]);
grid on;
xlabel('$$t\rm{\,/\,s}$$', 'Interpreter', 'latex','fontsize', 14)
ylabel('$$\hat{U}\rm{\,/\,V}$$', 'Interpreter','latex','fontsize', 14)

hold on;
p2 = plot(
    data.stepNorm.t, data.stepNorm.y,
    'k', "Linestyle", '--', 'LineWidth', 1);
legend('System response normalized', 'System input normalized',
       'Interpreter','latex');


%% FITTING
figure('Name', 'PT2 Fitting', 'NumberTitle', 'off');
p1 = plot(
    data.istNorm.t, data.istNorm.y,
    'k', "Linestyle", '-', 'LineWidth', 1);
axis([0 8 0 1.4]);
grid on;
xlabel('$$t\rm{\,/\,s}$$', 'Interpreter', 'latex','fontsize', 14)
ylabel('$$\hat{U}\rm{\,/\,V}$$', 'Interpreter','latex','fontsize', 14)

hold on;
p2 = plot(
    data.istNorm.t, unit_step_PT2(data.fitting.coeff, data.istNorm.t),
    'k', "Linestyle", '--', 'LineWidth', 1);
legend('System response normalized', 'Fitted PT2 System',
       'Interpreter','latex');
```

C.1.2 PT2 fitting function

```
function y = unit_step_PT2(par_PT2, t)
k=par_PT2(1);
w_n=par_PT2(2);
zeta=par_PT2(3);

w_d = w_n * sqrt(1 - zeta^2);

y = 1 - (exp(-zeta .* w_n .* t))./(sqrt(1 - zeta^2))  .*
         sin(w_d .* t + atan((sqrt(1 - zeta^2))./(zeta)));
end
```

C.2 SECOND LABORATORY

```
%% Clean up
clear all
close all
clc


%% Raw data
measure = importdata('measure.mat');
data.ist.t = measure.v_sensor.time;
data.ist.y = measure.v_sensor.data;
data.step.t = measure.v_fan.time;
data.step.y = measure.v_fan.data;


%% Trimming data
i_before = 122240;
idx_1 = 122244;
idx_2 = 126000;
dt = 2e-3;
data.istTrimmed.t = [0:idx_2-idx_1]'*dt;
data.istTrimmed.y = data.ist.y(idx_1:idx_2);
data.stepTrimmed.t = [0:idx_2-idx_1]'*dt;
data.stepTrimmed.y = data.step.y(idx_1:idx_2);

% Normalize Motor
data.GainMotor = data.step.y(idx_1) - data.step.y(i_before);
data.OffsetMotor = data.step.y(i_before);


%% Normalize data
% Offset
data.OffsetSensor = min(data.istTrimmed.y);
data.istNorm.y = data.istTrimmed.y-data.OffsetSensor;

% Normalize Sensor
idx_1 = 1876;
idx_2 = 3757;
data.GainSensor = mean(data.istNorm.y(idx_1:idx_2));

data.istNorm.y = data.istNorm.y/data.GainSensor;
data.istNorm.t = data.istTrimmed.t;

data.stepNorm.y = data.stepTrimmed.y/max(data.stepTrimmed.y);
data.stepNorm.t = data.stepTrimmed.t;


%% Fitting data
data.fitting.Para.a=1; %k
data.fitting.Para.b=2; %w_n
data.fitting.Para.c=0.8; %zeta
data.fitting.coeff = lsqcurvefit(
    @unit_step_PT2,
    [data.fitting.Para.a data.fitting.Para.b data.fitting.Para.c],
    data.istNorm.t, data.istNorm.y);


%% State space model
[A B C D] = tf2ss(
    data.fitting.coeff(2)^2,
    [1
    2*data.fitting.coeff(3)*data.fitting.coeff(2)
```

```
    data.fitting.coeff(2)^2]);


%% pole placement for state feedback
data.pp.ss.ts = 4; % in s
data.pp.ss.ps = 5/100; % in %

% Calculation of zeta
data.pp.ss.zeta = sqrt(log(data.pp.ss.ps)^2/(pi^2+log(data.pp.ss.ps)^2));

% Calculation of natural frequency
data.pp.ss.w_n = 4/(data.pp.ss.zeta*data.pp.ss.ts);

% Calcuate the pole regions complex conjugated:
data.pp.ss.p2 =
    -data.pp.ss.zeta*data.pp.ss.w_n +
    1i * data.pp.ss.w_n*sqrt(1-data.pp.ss.zeta^2);
data.pp.ss.p3 =
    -data.pp.ss.zeta*data.pp.ss.w_n -
    1i * data.pp.ss.w_n*sqrt(1-data.pp.ss.zeta^2);

% Add 3th pole which is nondominant ==> so more left
data.pp.ss.p1 = real(data.pp.ss.p2) * 2;
data.pp.ss.pl = [data.pp.ss.p1;data.pp.ss.p2;data.pp.ss.p3]

% Calculate new matrices:
% No error according due to step input see script ex. 3.4
A_strich = [0,-C;
            0, A(1,1),A(1,2);
            0 ,A(2,1),A(2,2)];

B_strich = [0;B];

% Poleplacement:
data.pp.ss.K = place(A_strich,B_strich,data.pp.ss.pl)
data.pp.ss.Ki = data.pp.ss.K(1)-1;
data.pp.ss.Kp = data.pp.ss.K(2:end);


%% Pole placement for obeserver gain
data.pp.ss.os = [-4; -6];
% Poleplacement
data.pp.ss.L = place(A',C',data.pp.ss.os);
data.pp.ss.L = data.pp.ss.L';
```