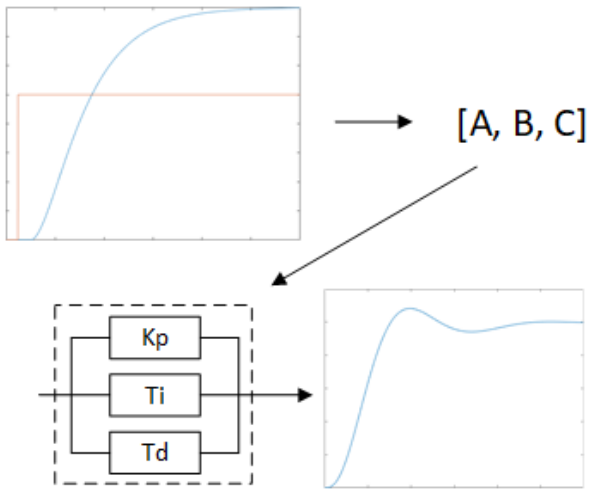
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2019
Industrial IT and Automation

State Space Model Based PID Controller Tuning



Preben Sandve Solvang

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2019
Title: *State Space Model Based PID Controller Tuning*
Pages: *171*
Keywords: *PID and PI controllers, tuning, process control, robustness, performance*

Student: *Preben Sandve Solvang*
Supervisor: *David Di Ruscio*
External partner: *None*
Availability: ***Confidential***

Summary:
Advances in digital computing over the last years have resulted in new and powerful tools for obtaining process models. An example of such a tool is the dsr toolbox, which gives a state space model based on measured input/output data. Also, new control strategies based on these models have developed, usually involving optimization techniques. Despite this, the classical PID controller still has advantages and remain the most used control technique.

The goal of the thesis was to compare different methods for tuning PID controllers. The advantages and disadvantages of the different methods should be explained and suggestions of how the methods could be used with state space models should be discussed. The Matlab pidtune function and the delta tuning rules should be explained and evaluated in relation to state space models.

The tuning methods Ziegler Nichols, SIMC, Cohen-Coon, and optimization tuning in addition to δ-tuning and pidtune, was chosen to examine in detail. To obtain model parameters for controller tuning from state space models, a graphical method, an optimization method and the Matlab function procest was used. Pidtune, mftune, megatuner, and optimization based tuning is used directly with SSM and was also tested. For method comparison, both commonly known process models and random models were used.

The methods which can be used directly on state space models give the best results in terms of successful tuning attempts. For many higher order SSM, process describing variables such as K, θ , T, R, and L can be found successfully by graphical estimation or optimization. These variables are then used for PID controller tuning. The graphical method is the fastest and gives the highest success-rate, while optimization estimation results in higher closed-loop performance.

Preface

This report presents an answer to the master thesis titled ”State Space Model Based PID Controller Tuning”, and is a partial fulfillment of Master of Science in Industrial IT and Automation. The project was conducted during the 4th semester of the masters' program, at the University of Southeast Norway.

The reader should possess basic knowledge in the field of control engineering and Matlab, which have been extensively used in this work. However, the report gives a short introduction to the topics system identification, state space models, PID controllers, and PID controller tuning methods. The thesis aims to further connect classical control theory and PID tuning methods to state space models. Many of the experiments conducted during this work are based on simulating large amounts of randomly generated state space models.

I would like to express my sincere gratitude to my supervisor, Associate Professor David Luigi Di Ruscio for always being available and providing me with his guidance. His published articles on the delta tuning method, as well as his lecture notes have been a great help during my work with this thesis. Finally, I would like to thank my friends and fellow students who have helped me during this period, especially Jonas Nilsen, who provided his help with proofreading.

The following tools and software were used during the project:

- TeXmaker
- Matlab
- MS Office
- MS Visio

Porsgrunn, 14th June 2019

Preben Sandve Solvang

Contents

List of Figures

List of Tables

Nomenclature

Symbols and abbreviations used in the report

Abbreviation

2DOF	2 Degree Of Freedom
CC	Cohen and Coon tuning
DIPTD	Double Integrating Plus Time Delay
DS	Direct Synthesis
FOIPTD	First Order Integrating Plus Time Delay
FOPTD	First Order Plus Time Delay
IAE	Integrated Absolute Error
IAEr	IAE for SP step response
IAEv	IAE for disturbance step response
IMC	Internal Model Control
IPTD	Integrating Plus Time Delay
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
MIMO	Multiple Input Multiple Output
MPC	Model Predictive Control
MSE	Mean Square Error
PID	Proportional, Integral, Derivative
PM	Phase Margin
PRC	Process Reaction Curve
RHP	Right Half Plane
SID	System Identification
SIMC	Simple/Skogestad Internal Model Control
SISO	Single Input Single Output
SOPTD	Second Order Plus Time Delay
SP	Set Point
SSM	State Space Model
SVD	Singular Value Decomposition
TF	Transfer Function
UC	Ultimate Cycle
UG	Ultimate Gain
ZN	Ziegler and Nichols

Symbol	**Description**
A, B, C, D	System matrices
dt	Sampling time
e	Error signal
h_0	Loop transfer function
h_c	Controller transfer function
h_p	Plant transfer function
J	Performance index, cost function
k	Discrete time
K	Process gain
K_p	Proportional gain
K_i	Integral gain
K_d	Derivative gain
K_u	Ultimate gain
L	Ziegler's Lag
M_s	Maximum sensitivity peak
P_u	Ultimate period
r	Reference variable
R	Reaction rate
s	Transfer function operator
T	time constant
T_i	Integral time
T_d	Derivative time
u	Control signal
v	Disturbance signal
W	Weighting coefficient
x	State
y	Output signal
ω_c	Gain crossover frequency
ω_{180}	Phase crossover frequency
θ	time delay

1 Introduction

The main goal in the field of control engineering is to design a system which behaves in a certain manner with little to no human interaction. Today, various control techniques are applied in all industries and fields, from simple thermostats to medical applications and large process plants. This makes control an important part of people's lives, even though it may not be noticed, a sign that it works well. Even humans themselves are performing control operations in everyday life when adjusting something based on what is being experienced. Examples of this are adjusting the water temperature while taking a shower or adjusting the force applied to the gas pedal while driving a car. This is what is known as closed-loop control, based on feedback.

In closed-loop control, the process output is measured with a sensing device, and based on what the desired value of the output is, a process input is applied to the system. One way of deciding the magnitude of the process input is to use a PID controller, and this is also the most commonly applied technique. Because of the PID controllers popularity, it is interesting to examine various tuning methods and compare them to uncover differences in performance and robustness.

1.1 Background

The field of system identification has its roots in statistical methods in the 1950s, but it is reckoned that the theory of system identification had its beginning in the 1960s [1]. With advances in digital computing in the 1980s, the methods for system identification became more powerful, and today various algorithms for creating state space, and other models based on measured input and output data are common. These models can be of any order and contain important information about the system, such as gain, zeros, and poles. One of the most important uses for such models is the design of controllers, as stated by Ljung [2]. Common PID tuning methods such as SIMC and Ziegler Nichols, are based on lower order transfer function models. When the model is of a higher order, steps must be taken to approximate a model. This might be difficult in some cases, and for that reason it is interesting to investigate methods which can be used for tuning based directly on state space models. One method that can do this is the Matlab pidtune function. The described workflow for tuning controllers is illustrated in figure 1.1.

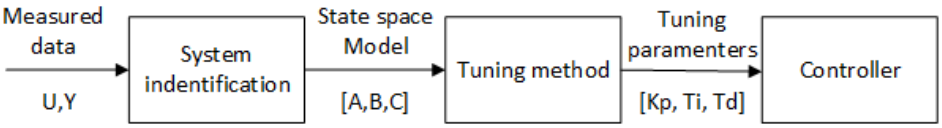


Figure 1.1: Workflow which forms the basis for the thesis, tuning a PID controller based on a state space model

1.2 Objectives and goals

Specific goals and objectives for the thesis, based on the task description given in appendix A:

- Give an overview of the process of tuning a controller based on SSM
  - Describe how system identification can be used to obtain a SSM
  - Describe properties of SSM and how they can be used to tune a controller
- Give an overview of methods for tuning PID controllers
  - Give background information on PID controllers
  - Select different tuning methods to examine more closely
  - Identify the advantages and disadvantages of the different tuning methods
  - Discuss how the methods can be used with state space models
- Give a more detailed user-specific description of MATLAB pidtune
  - Explain how the method is used
  - Examine the different options which can be specified by the user
  - Evaluate the performance of the tuning method
- Evaluate the possibility to extend the δ tuning rules to be based on state space models
  - Give an overview and explanation of the δ tuning rules
  - Evaluate how they can be used with SSM
- Compare the different methods by using simulation experiments
  - Use randomly generated SSM to compare different methods
  - Evaluate the robustness and performance of the different methods
  - Evaluate the performance of different approaches for tuning from SSM

Tuning of controllers for unstable processes and the effects of signal noise was not included in the scope of this thesis. Extra PID controller functionality, such as anti-windup and limited derivative action using filtering has not been included in the testing presented in this thesis. Signal noise has not been considered when evaluating controllers either.

1.3 Report structure

The structure of the report is as follows:

- Chapter 1 is the introduction, containing the background and the objectives for the thesis.
- Chapter 2 contains background theory on system identification and SSM, as well as how to obtain process parameters for controller tuning from them
- Chapter 3 gives background information on PID controllers, different formulations and structures, as well as measures for robustness and performance used in the thesis
- Chapter 4 presents different methods for tuning PID controllers, with advantages, disadvantages and possibility for use with state space models
- Chapter 5 compares different tuning methods using common system models, as well as randomly generated state space models. This is where the results of the thesis is presented and the reader should have appendix B and C available while reading this chapter
- Chapter 6 contains the discussion and suggestions for further work
- Chapter 7 gives a short conclusion to the thesis

2 Background theory on state space models and system identification

This chapter describes the state space representation of dynamic systems and how such models can be obtained. State space methods for system representation and controller design has some advantages over frequency domain methods. They are better suited for digital implementations, makes MIMO systems easier to handle, describe the internal state of the system and gives information about the initial state of the system [3]. SSM is also the cornerstone of modern control theory. The classical transfer functions are limited to represent the amount of Laplace transform at the output relative to the Laplace transform of the input [4]. State space models can be used to represent any system, whereas transfer functions are only valid for LTI systems.

2.1 State space models

A state space model is a type of dynamic model, used to describe a physical system. Characteristic for the SSM is that it can only consist of 1st order differential equations. A k-order system is then described by k number of differential equations on the form shown in equation 2.1 [3]. The state variables, x, does not have to be physical quantities that are related to the system [3], which can make the term ”state” challenging to comprehend.

$$\begin{aligned}
\dot{x}_1 &= \frac{dx_1}{dt} = f_1(x_1, x_2, .., x_k, u_1, u_2, .., u_l, t) \\
\dot{x}_2 &= \frac{dx_2}{dt} = f_1(x_1, x_2, .., x_k, u_1, u_2, .., u_l, t) \\
&\quad \ldots\ldots\ldots\ldots\ldots \\
\dot{x}_k &= \frac{dx_k}{dt} = f_1(x_1, x_2, .., x_k, u_1, u_2, .., u_l, t)
\end{aligned} \tag{2.1}$$

By placing the states and inputs in vectors, called state vector and input vector, as shown in equation 2.2, the SSM can be formulated as shown in equation 2.3, which is a non-linear model.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_k \end{bmatrix} \quad u = \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ u_l \end{bmatrix} \tag{2.2}$$

$$\dot{x} = \frac{dx}{dt} = f(x,u,t) \tag{2.3}$$

The state space differential equations for linear systems are on the form seen in 2.4. For time-variant systems, coefficients a and b are functions of time.

$$\begin{aligned} \dot{x}_1 &= \frac{dx_1}{dt} = a_{11}x_1 + .. + a_{1k}x_k + b_{11}u_1 + .. + b_{1l}u_l \\ \dot{x}_2 &= \frac{dx_2}{dt} = a_{21}x_1 + .. + a_{2k}x_k + b_{21}u_1 + .. + b_{2l}u_l \\ &\text{.....} \\ \dot{x}_k &= \frac{dx_k}{dt} = a_{k1}x_1 + .. + a_{kk}x_k + b_{k1}u_1 + .. + b_{kl}u_l \end{aligned} \tag{2.4}$$

As with the non-linear model, the model can be made more compact by using vector notation. Equation 2.5 shows the linear time-invariant, state equation. This is the form used for control purposes.

$$\dot{x} = \frac{dx}{dt} = Ax + Bu \tag{2.5}$$

A and B in equation 2.4 are matrices given by 2.6 and with the following properties:

- A - State or system matrix, always square kxk, the eigenvalues of A equals the poles of the system
- B - Input matrix, kxl, where l is the number of inputs. In most cases, when the number of inputs is less than the number of states, B is a thin, tall matrix. If there is one input B is a column vector

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} \end{bmatrix} \; B = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,l} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,l} \end{bmatrix} \tag{2.6}$$

The output equation has the same structure as the state equation, with y in compact vector form. This equation is given by 2.7.

$$y = Cx + Du \tag{2.7}$$

Where C and D are matrices with the following properties:

- C - Output matrix, mxk, where m is the number of outputs. If the state is measured directly, C is 1
- D - Feed trough matrix, $mx1$, means that there is a direct connection between the output and the input, u, and the output, y. D is usually 0, and for control purposes, it is desired that D is 0. For this reason, the D matrix set to 0 in all experiments conducted in this thesis.

The discrete form of the SSM is given by:

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k$$

The state space model structure represented in a block diagram is shown in figure 2.1. This representation makes it easier to see how the SSM is structured.

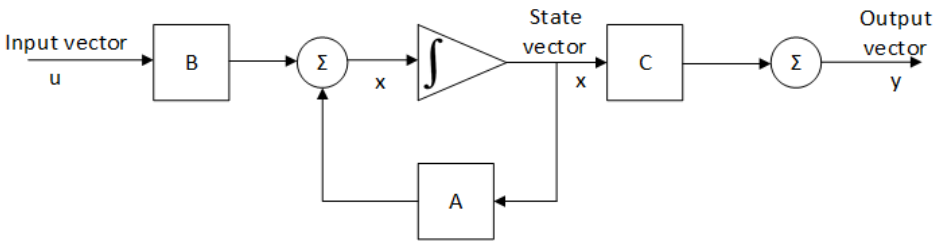


Figure 2.1: Block diagram representation of a state space model

2.1.1 State space model conversion

Classical control theory, together with most methods for tuning PID controllers rely on transfer functions. This makes converting state space models to transfer functions an important tool when tuning controllers. This can be done using the procedure described here. Consider the transfer function and state space model formulas:

$$\dot{x} = Ax + Bu, \quad y = Cx, \quad hp(s) = \frac{Y(s)}{U(s)}$$

Taking the Laplace transform on the state space model gives:

$$sX(s) = AX(s) + BU(s)$$
$$sY(s) = CX(s)$$

Solving for X in the state equation and then inserting the expression in the output equation results in:

$$Y(s) = C[Is - A]^{-1}BU(s)$$

This gives 2.8, which is used for conversion between state space models and transfer functions. This conversion can also be done in Matlab by using the function ss2tf. Nonlinear models must be linearized before converting to transfer function.

$$hp(s) = \frac{Y(s)}{U(s)} = C[Is - A]^{-1}B \tag{2.8}$$

The half rule

When using system identification tools, as presented in section 2.2, the obtained model may be of a higher order. In many cases, these higher order models can be approximated to a 1st or 2nd order model. One way of doing this is by using the half rule, where T_1 and θ for a FOPTD plant are given by:

$$T_1 = T_1 + \frac{1}{2}T_2$$
$$\theta = \theta + \frac{1}{2}T_2 + T_3 + ... + T_n$$

The time constants and dead time for a SOPTD model are given by:

$$T_1 = T_1$$
$$T_2 = T_2 + \frac{1}{2}T_3$$
$$\theta = \theta + \frac{1}{2}T_3 + T_4 + ... + T_n$$

2.1.2 Numeric simulation of a state space model

A SSM can be simulated numerically using the formulas presented in this section. This is done by using a modified version of the state equation in 2.5 and the output equation, 2.7. The step size, initial states, and input, u, must be specified to perform the simulation. Equation 2.9 is used to simulate a SSM in open loop. For closed loop simulation with a PID controller, u is found using a discrete representation of the controller.

$$\begin{aligned} y_k &= Cx_k \\ x_{k+1} &= x_k + dt(Ax + Bu) \\ &\text{For } k = 0...N \end{aligned} \tag{2.9}$$

A way of implementing time delay is using a vector of $length = \theta/dt$. For each iteration all the values in the array are shifted by one, the calculated value for y is inserted into index 1, and finally, the value of the last index is used as the actual output.

2.2 System identification

When tuning a PID controller, the first step is often to obtain a model of the system. This is known as the identification problem and is illustrated in figure 2.2. One way of creating a dynamic model of the system is to log the inputs and corresponding outputs over a time-period and use this information to estimate a model. This method is known as system identification and has the advantage that it is not necessary to know the underlying system dynamics in detail, which makes SID a black box method. Another way is to derive a model based on first principles, using laws as mass balance and heat equations.

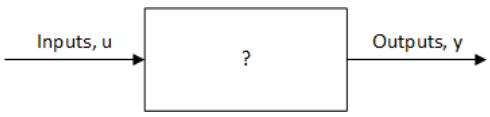


Figure 2.2: The identification problem

2.2.1 Manual system identification

In SID, realization theory can be used to find the matrices A, B, C and the system order, n. The first step is to calculate impulse responses H_k from the measured inputs and outputs, and organize these in Hankel matrices $H_{1|L}$ and $H_{2|L}$, as shown in 2.11. Equation 2.10 is the impulse response matrices for SISO systems [5].

$$H_k = \frac{y_k}{u_0} \tag{2.10}$$

$$H_{1|L} = \begin{bmatrix} H_1 & H_2 & \cdots & H_J \\ H_2 & H_3 & \cdots & H_{J+1} \\ \vdots & \vdots & \ddots & \vdots \\ H_L & H_{L+1} & \cdots & H_{L+J-1} \end{bmatrix}, \quad H_{2|L} = \begin{bmatrix} H_2 & H_3 & \cdots & H_{J+1} \\ H_3 & H_4 & \cdots & H_{J+2} \\ \vdots & \vdots & \ddots & \vdots \\ H_{L+1} & H_{L+2} & \cdots & H_{L+J} \end{bmatrix} \tag{2.11}$$

The Hankel matrices are related to the observability and controllability matrices O_L and C_J in 2.12, by the equations 2.13. This relationship is used in realization theory to find the state space matrices.

$$O_L = \begin{bmatrix} D \\ DA \\ \vdots \\ DA^{L-1} \end{bmatrix}, \quad C_J = \begin{bmatrix} B & AB & A^2B & \cdots & A^{J-1}B \end{bmatrix} \tag{2.12}$$

$$\begin{aligned} H_{1|L} &= O_L C_J \\ H_{2|L} &= O_L A C_J \end{aligned} \tag{2.13}$$

Using SVD on $H_{1|L}$, and output realization gives the equations in 2.14, where $U_2 S_2 V_2^- 1$ can be neglected. B and C can be found from O_L and C_J. The number of non-zero values in the diagonal matrix S_1, is the system order, n.

$$\begin{aligned} SVD(H_{1|L}) &= U_1 S_1 V_1^T + U_2 S_2 V_2^T \\ O_L &= U_1 \\ C_J &= S_1 V_1^T \end{aligned} \tag{2.14}$$

A is found by performing SVD on $H_{2|L}$, and equation 2.15.

$$\begin{aligned} SVD(H_{2|L}) &= U_1 A S_1 V_1^T + U_2 S_2 V_2^T \\ A &= U_1^T H_{2|L} V_1 S_1^{-1} \end{aligned} \tag{2.15}$$

2.2.2 System identification using Matlab

SID using Matlab or other computer tools are a simple way of obtaining models when the input/output data for the process is available. Inputs can also be variables that are not controllable. 3 methods for creating state space models from the system identification toolbox and the dsr toolbox have been tested for demonstration purposes. This demonstration shows that state space models can be obtained using very few lines of code. The following system identification methods are used:

- n4sid (system identification toolbox) - Estimates a SSM using the subspace method [6]
- ssest (system identification toolbox) [7]
- dsr (dsr toolbox) - Uses subspace identification [8]

The data for a simple SISO heating process, shown in figure 2.3 has been used to estimate the models. The data-set consists of 1000 samples, taken at an interval of 0.08 seconds. The input is the voltage applied to a heater, and the output is voltage measured from a thermocouple. The first 500 samples are used for system identification, and the rest is used for verification.

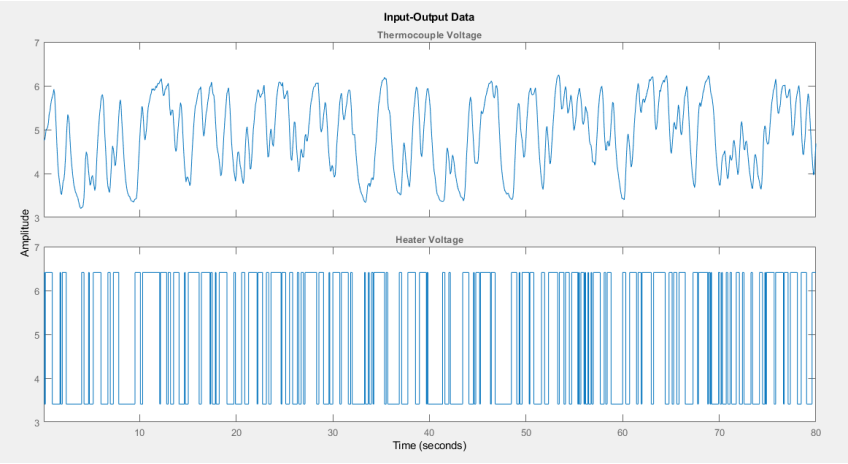


Figure 2.3: SISO air-heater dataset used for system identification example, y and u

When using n4sid and ssest, the data is structured using the ”iddata” function which takes the inputs U, Y, and dt. The dsr function takes the input/output data directly. U is a Nxl matrix, where N is the number of observations and l is the number of input variables. Y is a N x m matrix, where m is the number of output variables. Feature standardization should be used to make the data zero mean.

Figure 2.4 shows the resulting system matrices using ssest, n4sid, and dsr functions, respectively. The matrices from ssest and n4sid are almost identical, while the result

from dsr differs more, which shows that the same system can be represented in several ways.

$$A = \begin{bmatrix} -0.1399 & 2.0584 & 2.4748 \\ -3.2571 & -0.8590 & -6.8529 \\ 1.6798 & 17.7272 & -16.5520 \end{bmatrix} B = \begin{bmatrix} 0.0375 \\ 0.0031 \\ -0.8077 \end{bmatrix} C = \begin{bmatrix} 18.3580 & 0.1248 & -0.1913 \end{bmatrix}$$

$$A = \begin{bmatrix} -0.2356 & 2.0267 & 2.4581 \\ -3.2290 & -1.0028 & -6.7818 \\ 1.6855 & 17.6985 & -16.7547 \end{bmatrix} B = \begin{bmatrix} 0.0395 \\ 0.0045 \\ -0.8526 \end{bmatrix} C = \begin{bmatrix} 18.3565 & 0.1266 & -0.1841 \end{bmatrix}$$

$$A = \begin{bmatrix} -0.3814 & -13.5077 & -11.0568 \\ 0.5788 & -0.3946 & -45.9274 \\ -0.1315 & 2.3663 & -13.9618 \end{bmatrix} B = \begin{bmatrix} -0.4852 \\ -0.1617 \\ -0.6361 \end{bmatrix} C = \begin{bmatrix} -0.5856 & -0.7100 & -0.3912 \end{bmatrix}$$

Figure 2.4: SSM matrices A, B, C obtained using different SID tools: ssest, n4sid, and dsr

2.2.3 System validation and simulation

To simulate a system is referred to as the simulation problem, illustrated in figure 2.5. Solving this problem gives information about how the known system responds given a set of input signals. System simulation is useful for validating the identified model, as well as investigating the behavior of the system and tuning a controller.

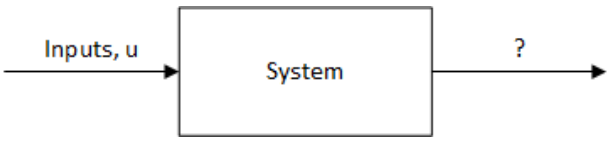


Figure 2.5: The simulation problem

Figure 2.6 shows the simulated output from the 3 identified state space models, together with the actual recorded data. The simulation is performed using the sim Matlab function, and the last 500 samples from the data set. This data was not used in the identification. The curves follow the original data closely and demonstrate the power of system identification. A way of validating models is to use the mean square error, MSE, given by 2.16. In this example, dsr performed best, with an MSE value of 0.03. Ssest and n4sid scored 0.05 and 0.06, respectively.

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(y-\hat{y})^2 \tag{2.16}$$

where:

N = Number of samples
y = Real process output
$\hat{y}$ = Simulated process output

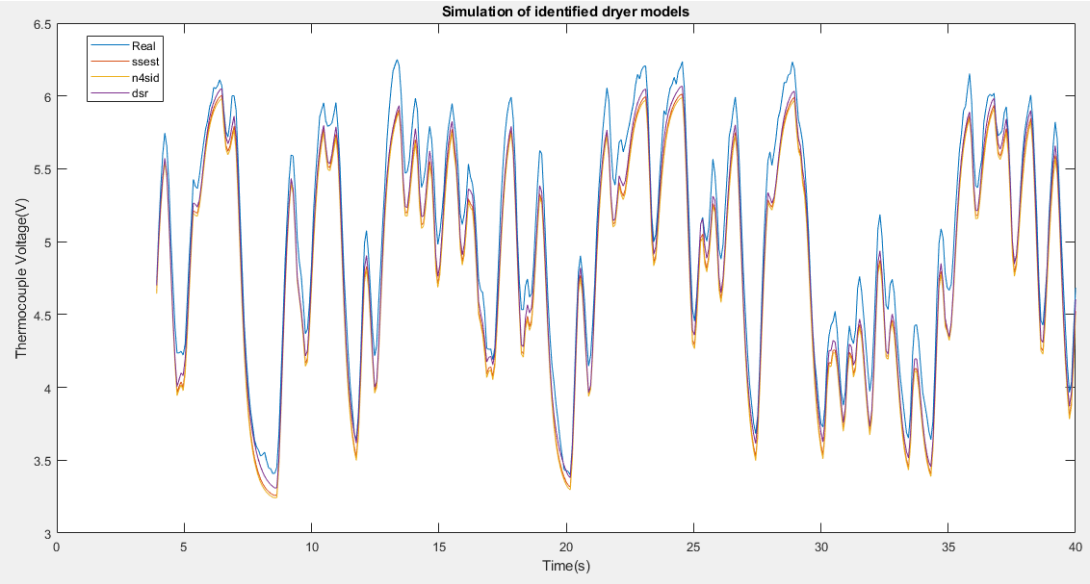


Figure 2.6: Comparison between output, y, for identified models and measurement from the real process

2.3 Common process models used for control

This section gives a short description of common model types that are essential in control engineering, as many tuning methods is based on these models. Transfer function and SSM representation of the models is given, as well as practical examples. Figure 2.7 shows the open loop step response for these characteristic process models.

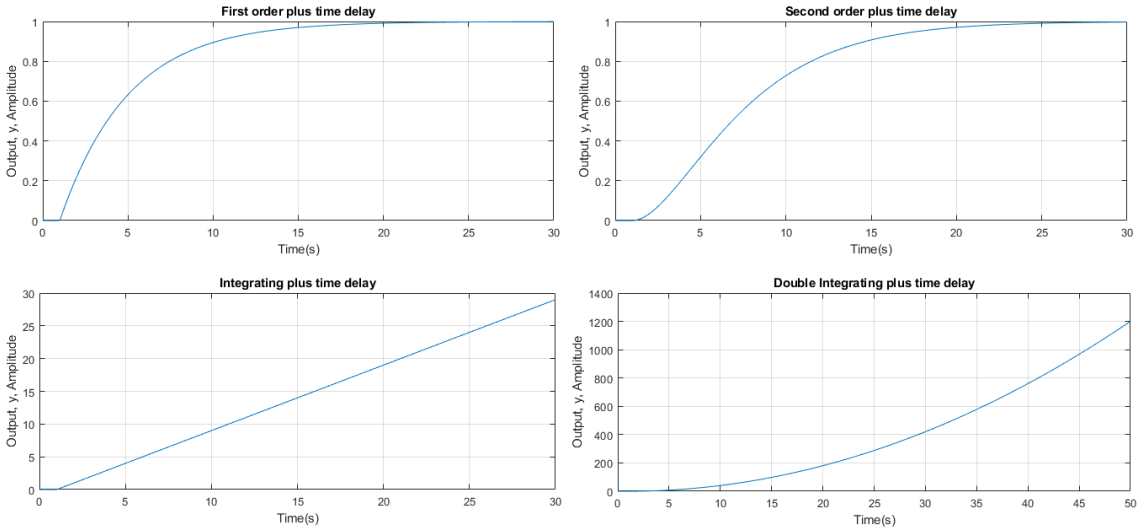


Figure 2.7: Open loop step response for FOPTD, SOPTD, IPTD and DIPTD systems

First order plus time delay

This is a common type of model that can be used to describe a variety of processes, one example is temperature. The model contains 3 parameters; gain, time constant, and dead time. Many PID tuning rules are based on this type of model, which makes it crucial for control, examples of this are ZN PRC, SIMC PI, and the Cohen-Coon tuning rules. The transfer function is given by 2.17 and the state space representations, with $K = 1$ and $T = 1$, by 2.18.

$$h_p = K\frac{1}{(Ts+1)}e^{-\theta s} \tag{2.17}$$

$$A = -1, B = 1, C = 1 \tag{2.18}$$

Second order plus time delay

The SOPTD model is like the FOPTD model used to represent a self-regulating process. In addition to the parameters used in the FOPTD model, it contains a second time constant and can be used to represent under-damped systems which have overshoot. The SIMC PID tuning rule is based on a SOPTD model, which are given by the transfer function 2.19. The state space representation with $K = 1$, $T_1 = 2$ and $T_2 = 1$ is given by 2.20.

$$h_p = K\frac{1}{(T_1 s+1)(T_2 s+1)}e^{-\theta s} \tag{2.19}$$

$$A = \begin{bmatrix} -2 & -1 \\ 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{2.20}$$

Integrating plus time delay

This type of plant is another common process type, representing for example level control. This process is not self-regulating, and the output continues to grow or shrink if proper control is not applied. The δ PI tuning rules are based on an IPTD model. 2.21 is the transfer function representations, and 2.22 is the SSM.

$$h_p = K\frac{1}{s}e^{-\theta s} \tag{2.21}$$

$$A = 0, B = 1, C = 1 \tag{2.22}$$

Double integrating plus time delay

The DIPTD plant is not uncommon and is difficult to control. The DIPTD models have 2 poles at the origin, and the output grows exponentially. Concrete examples of DIPTD processes are two large tanks in series or a system converting force to position. ZN and CC tuning rules cannot be used for this type of system, and the controller needs to have derivative action to control this type of process. The DIPTD TF is 2.23, and the state space representation is given by 2.24.

$$h_p = K\frac{1}{s^2}e^{-\theta s} \tag{2.23}$$

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{2.24}$$

2.4 Estimating process characteristics from SSM,

One of the goals for the thesis is to discuss how various PID tuning methods can be used together with SSM. This section suggests how to obtain values that are commonly used for tuning. As stated in chapter 1, classical control theory based on transfer functions is widely used as a basis for PID tuning. Variables such as gain, time constants, and time delay can be found from the models presented in section 2.3, and can then be used to tune a controller. Other process describing variables used for tuning are reaction rate, lag, ultimate gain, and ultimate period. A suggested decision tree for tuning a PID controller based on SSM is presented in figure 2.8, and the alternatives are also listed below:

- Convert SSM to transfer function as described in section 2.1.1, if necessary use model reduction (half rule) to arrive at a 1st or 2nd order model
- Simulate SSM as described in section 2.1.2, to obtain the vectors U, Y, and t. This data can then be used in the following ways to get the desired models.
  - Read R, L, and K from step response
  - Read K, T_1, and θ from step response
  - Use optimization to fit a low order transfer function to the data
  - Use Matlab function procest to estimate a transfer function
- Find margins from SSM or approximated lower order model, then obtain ultimate gain and ultimate period

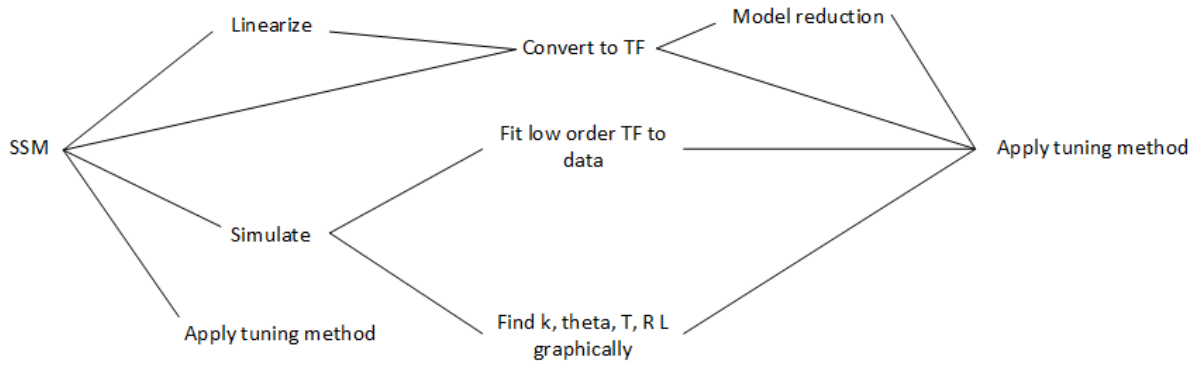


Figure 2.8: Suggested decision tree for tuning PID controllers based on SSM

2.4.1 Estimating process values graphically from an input step response curve

By estimating process values graphically from an input step response curve, it is meant to plot vector, Y, after a input step and read the values from the graph.

Reaction rate and lag from an input step response curve

The measures reaction rate, R, and lag, L, was introduced by Ziegler and Nichols and is used to describe a process. R and L describes FOPTD and IPTD processes well. R is the steepest gradient of Y, i.e., the tangent line with the steepest slope. The lag, L, is an approximation of the time delay and is found by locating the intersection between the x-axis and the tangent to Y at the point where the R was found. These properties are shown graphically in 2.9.

Lag can be found using the following formula:

$$L = t_R - \frac{Y_R - Y_0}{R}$$

where:

t_R = Time instant when R occurs
Y_R = Value of Y, at t_R
Y_0 = Value of Y, at t_0

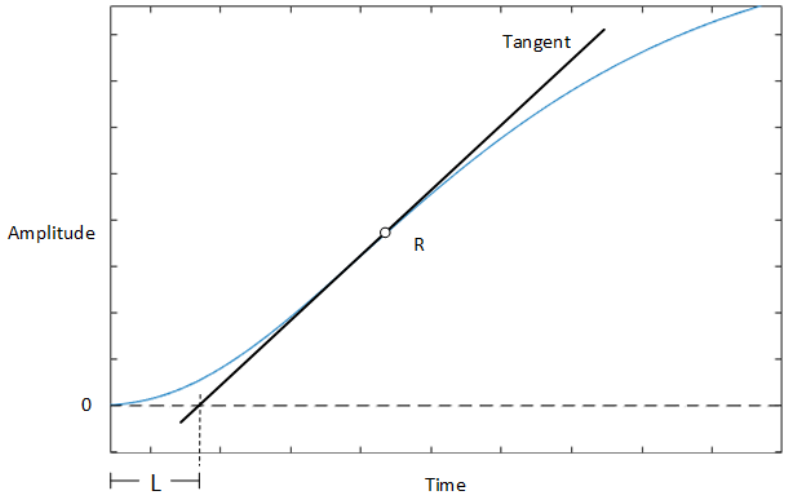


Figure 2.9: Lag, L, and Reaction rate, R, identified in a step response

FOPTD from step response

Time delay is found graphically from the step response data in the same way as L. The first order time constant is the time it takes y to reach 63.2 % of the final value, and can be found using formula 2.26. Gain is the change in y, relative to u, and for a unit step response is given by Δy. Matlab function dcgain or 2.25 is used to find K. The values are illustrated in figure 2.10.

$$K = \frac{Y_n - Y_0}{U_n - U_0} = Y_n \tag{2.25}$$

Where:

Y_n = Final value of Y
Y_0 = Initial value of Y
U_n = Final value of U
U_0 = Initial value of U

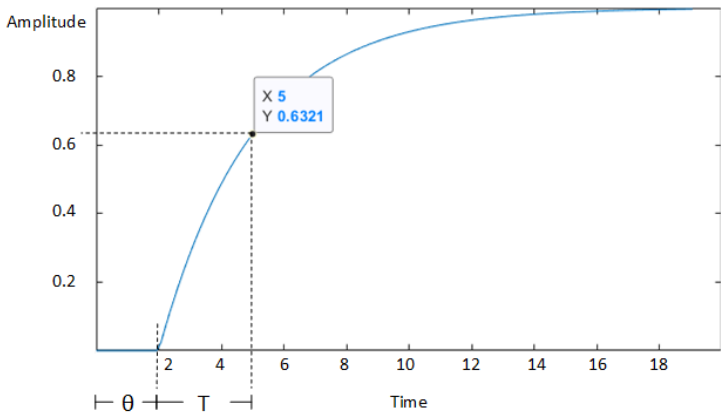


Figure 2.10: Time constant, T and time delay, θ identified in a unit step response

$$T = \frac{Y_n - Y_0}{R} = t_{0.632} - L \tag{2.26}$$

Where:

$t_{0.632}$ = Time instant when Y is 63.2% of final value

SOPTD from step response

For a second-order process, it is more difficult to find the parameters graphically. For under-damped models it is possible to find damping coefficient, ζ and second-order time constant, T_s, and then using the following relationship between second-order model formulations:

$$hp = K\frac{1}{(T_1 s+1)(T_2 s+1)}e^{-\theta s} = K\frac{1}{T_s^2 s^2 + 2\zeta T_s s + 1}e^{-\theta s}$$
$$T_1 T_2 = T^2$$
$$T_1 + T_2 = 2\zeta T$$

To find $\zeta$and T_s from a step response, the overshoot ratio, and the period is found graphically and then applied in formulas:

$$\zeta = \sqrt{\frac{ln(OS)^2}{\pi^2 + ln(OS)^2}}$$
$$T_s = \frac{\sqrt{1-\zeta^2}}{2\pi} P$$

Where:

OS = Overshoot ratio
P = Period

This only works with underdamped systems, with $\zeta < 1$, as other models do not give overshot or period. For this reason, this method has not been used in this thesis, and are only briefly explained here.

2.4.2 Model fitting using optimization

Another way of estimating time constants, gain and time delay is to use an optimization algorithm. This is done by choosing the values of x in 2.27, which minimizes the value of J in 2.28. The structure of x is chosen according to the desired model type. For each iteration of the optimization process, a step response is simulated, and the output is compared with the original model, using MSE. The values of x which give the best fit are chosen. The optimization is done using fmincon in Matlab. The optimization problem is given by:

$$\hat{x} \ = \ min \ J(x) \ s.t \begin{cases} l_b \leq x \leq u_b \\ T_2 < T_1 \end{cases}$$

where:

$$x = [K, \ T_1, \ \theta], \ x = [K, \ T_1, \ T_2, \ \theta], \ x = [K, \ \theta] \tag{2.27}$$

$$J = MSE = \frac{1}{N} \sum_{n=1}^{N} (y - y_{est})^2 \tag{2.28}$$

$$l_b = \begin{bmatrix} -\text{inf} \\ 0.01 \\ 0.01 \end{bmatrix}, \ u_b = \begin{bmatrix} \text{inf} \\ \text{inf} \\ 20 \end{bmatrix}$$

A third way of estimating a lower order model is to use Matlab function procest. This function also uses inputs Y, U, and t, the estimate a transfer function on a specified form. This function uses a combination of different search methods in sequence at each iteration for parameter estimation.

2.4.3 Comparison of estimation methods

This section gives a comparison of the estimation methods described in 2.4.1 and 2.4.2. The model fit and time usage for the different methods are compared and presented in table 2.1. For the comparison, 50 random 15th order SSM with 1 second time delay was used. The MSE value stated is the median of the 50 calculated MSE values. The graphical method stands out as the fastest, while the SOPTD optimization method is the slowest. The difference in MSE between FOPTD and SOPTD approximations are small.

Table 2.1: Comparison of model estimation methods, based on 50 random 15th order SSM

	Time(s)	MSE	MSE < 10
Optimization FOPTD	93.9	0.0108	45
procest FOPTD	94.5	0.0106	44
Graphical FOPTD	1.7	0.1099	39
Optimization SOPTD	186.8	0.0106	45
procest SOPTD	151.2	0.0095	43

Figure 2.11 shows the result of an experiment comparing the 5 different methods in terms of MSE values and the number of cases with MSE larger than 10, with increasing model order. 100 random models for each model order was used, and the MSE value in the plot is the median. The graphical estimation gives the worst fit but is close to the other methods until model order 8. The other methods give similar values for MSE. The exception is the SOPTD estimation using optimization, which gives a slightly higher value. The bottom plot is the number of systems where the MSE value is above 10, here the SOPTD optimization method is best. The graphically estimated models stand out as worst, with a maximum of 29 estimations with MSE larger than 10, for 15th order systems.

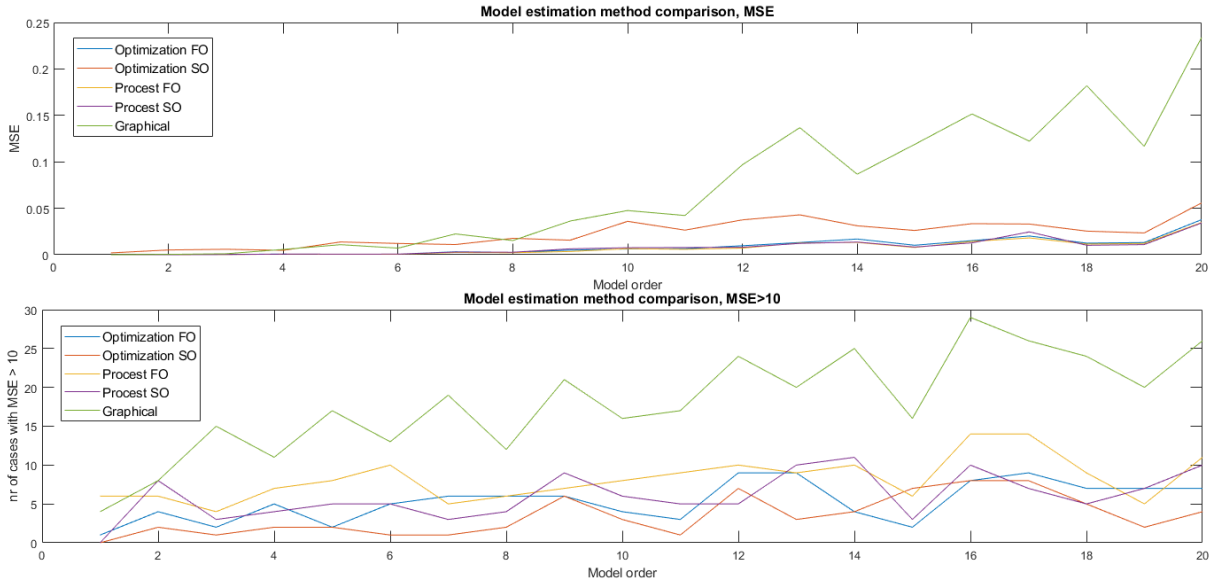


Figure 2.11: Comparison between model estimation methods in terms of median MSE, and the number of attempts with MSE>10, with increasing model order, based on 100 SSM per model order

3 Background theory on control engineering

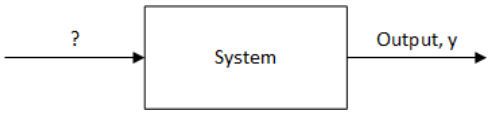


Figure 3.1: The control problem

This chapter introduces the PID controller, control loops, and various measures used to determine the performance and robustness of these control loops. The field of control engineering aims to solve a control problem, illustrated in 3.1. One way of achieving this is to use a PID controller, which continuously updates u as a function of e, where e is defined by 3.1. "Acceptable limits" mean that e should go towards zero as time goes towards infinity, as formulated in 3.2 [9]. The block diagram in figure 3.2 shows the basic negative feedback control loop used in PID control.

$$e = r - y \tag{3.1}$$

$$\lim_{t \to \text{inf}} e(t) = 0 \tag{3.2}$$

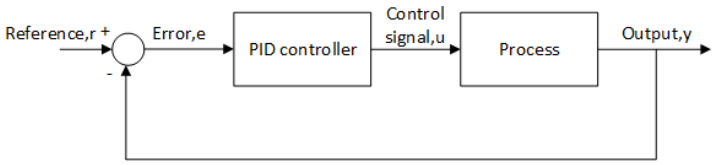


Figure 3.2: Feedback control loop, block diagram

3.1 History of control and current use of PID controllers

An estimated 95% of process control applications are of PI or PID type [10]. This is due to low cost and usability, as there are only 3 parameters to adjust. The formal

control law known as the PID controller has been around since 1922 [11], and due to its popularity, hundreds of different tuning rules have emerged. In the "Handbook of PI and PID Controller Tuning Rules" from 2009, 1731 different tuning rules are presented [12].

Classical control theory emerged in the 1930s and 1940s and was documented by Bode and others. It deals with LTI SISO systems using the Laplace transform and the s operator, utilizing the frequency domain. Tools in classical control theory for analyzing systems and design controllers include Nyquist plots, Bode plots, and root locus. Many methods for PID controller design relies on the frequency domain, and thus the classical control theory. Modern control theory, which had its beginning around 1960, is based on the state space representation of systems, in the time domain. With the modern control theory came new control strategies and different methods for synthesizing controllers, with a focus on optimal control. The modern control theory also deals with MIMO systems. However, the rise of modern control theory does not make the classical control theory obsolete; on the contrary, they fulfill each other. This makes it interesting to investigate how the tools of the classical control theory for tuning PID controllers can be utilized when the system is modeled using state space representation.

Despite the relative simplicity and few tuning parameters, it is believed that there is a fair amount of poorly tuned PID controllers operating in the industry. In an article published in 1993, it was found that 30 % of the control loops performed so badly that manual operation would be better. Another finding was that 25% of the controllers were operating at default factory settings [13]. Other problems with the control loops were poorly sized actuators and measurement problems like inadequate filtering. This illustrates that there is a lot to be gained by properly tuning the control loops and that it might be a neglected area of interest in many companies.

To further examine the state of the PID controllers in the industry today, a short questionnaire was sent to a selection of sizeable Norwegian process plants. They were asked how their control loops were tuned, by who and if they have a clear strategy for tuning. Although the number of answers was limited, the trend is the same. Ziegler Nichols or auto-tuning performed by technicians without any parent strategy is the most common method. Other methods that are mentioned are various approaches which are more or less guesswork. These methods are referred to as SWAG (Scientific Wild Ass Guess) and field tuning and are a collection of approaches used to adjust parameters manually until the closed loop response is within seemingly acceptable limits, as judged by the person performing the tuning. The complete results from the survey is in appendix E.

In an article from 2009 by Skogestad [14], it is argued that the advantages of simple feedback control need to be rediscovered periodically. It is claimed that feedback control may be discarded in many cases because of its simplicity, and the notion that since it is based on past measurements, it is not good enough. To contradict this, Skogestad points out 3 fundamental advantages for feedback control, being; it is the only way to fundamentally change the dynamics of a system; it is required for a system to adapt

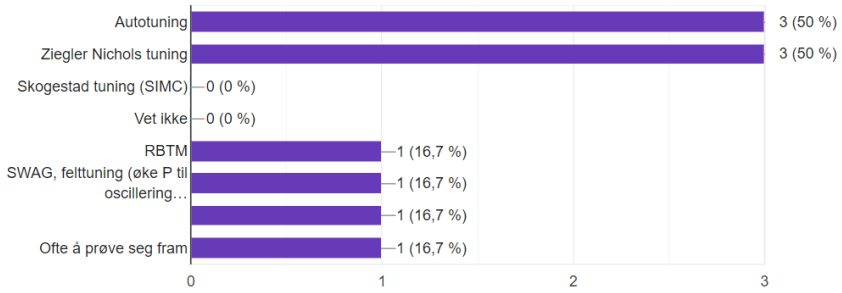


Figure 3.3: Survey result, controller tuning rules used in the industry

to new conditions; and it makes it possible to obtain tight control without an accurate model.

All of these points outline the importance of controller tuning and why it should be connected to the modern state space models.

3.2 PID controller parameters

The PID controller consists of 3 parts, the proportional term, integral term, and derivative term, which gives the abbreviation PID. The PID controller is the simplest form of controller which updates the control signal based on both past, current, and predicted future error. The proportional term contributes to the total output with a factor proportional to the current error. The integral term contributes with a factor proportional to the integral of the error, which is the sum of past errors. The derivative term contributes with a factor proportional to the derivative of the error, which is the predicted future error. This concept is illustrated in figure 3.4.

Tuning a PID controller is the process of adjusting how much each of the terms should contribute to the total control signal. This is done by using weighting coefficients, known as controller gains, which can be formulated in different ways, shown in section 3.3.

In many cases, it is not necessary to include all three terms of the controller, as they might reduce performance. When, for example excluding the derivative term, the controller is referred to as a PI-controller. Some common controller types and characteristics follow:

- P controller, pure proportional controllers are slightly more complex than an on/off controller and are not very common. A pure P controller always produces a steady state error and require high gain to reduce it. Too high gain may cause oscillations. This means that it can only be used in processes where a static offset can be accepted; in these cases, P controllers have an advantage due to simplicity and speed of response. In a control system with a cascade architecture, a P controller can be

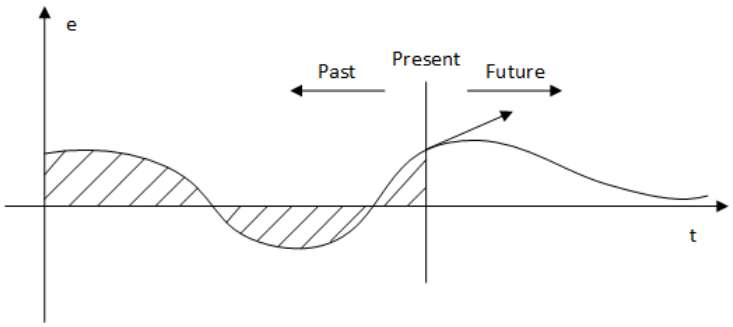


Figure 3.4: PID controller principle, a control signal, u, is calculated based on past, present and future value of the error, e

used in the inner control loop, as the offset is counteracted by the outer control loop.

- PI controllers are the most commonly used form of the controller. The integral term eliminates the steady state error but also makes the overall response slower. Used for pressure, level, and flow control.
- PID controllers are less used because the derivative term is sensitive to noise. Adding derivative action makes faster responses possible, as proportional and integral gain can be increased. This type of controller has an advantage in processes which are slow, with a high degree of inertia and non-linearity. An example of this is temperature control and conventional autopilot. Processes that are double integrating or oscillating needs derivative action to be stabilized.
- PD controller, well insulated thermal processes act as integrators, which makes the need for integral action disappear. This kind of processes allows for large proportional gain, eliminating the problem with steady state error and the need for integral action. PD controllers are also used for control of flying or underwater objects such as missiles or ships.

The following list gives some general statements about the effect of adjusting the controller gains, but might not be accurate in all cases. This is important to know when performing manual adjustments. A parallel controller is considered.

- P term - increase tuning parameter K_p
  - Faster tracking
  - More overshoot
  - Less stability
  - Decrease steady state error
- I term - increasing tuning parameter K_i

  - Faster tracking
  - More overshoot
  - Less stability
  - Decrease steady state error
- D term - increasing tuning parameter K_d
  - Faster tracking
  - Less overshoot
  - More stability

3.3 PID controller formulations

The PID controller can be formulated in a variety of ways, in both the continuous, discrete and Laplace domains. It is essential to know which formulation is being used when tuning and implementing the controller, as increasing K_i and T_i has the opposite effect. The choice of controller formulation has less importance and has little influence on the performance [15]. The different formulations are presented in this section.

In addition to the formulas presented here, it is common to implement some constraints, like anti-windup and bumpless transfer. The purpose of anti-windup is to prevent integral error to build up when the output saturates. Bumpless transfer is related to the switch between manual and automatic operating modes.

Equation 3.3 is the PID controller on standard form in the time domain. Another name used for this formulation is the ideal form. In this form, the gain, K_p, affects all the 3 controller terms.

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)d\tau + K_p T_d \frac{de}{dt} \tag{3.3}$$

By substituting $K_p/T_i = K_i$ and $K_p T_d = K_d$, the PID controller on parallel form in 3.4 is obtained. The parallel form is more intuitive to work with if the gains are to be adjusted manually, as they are independent of each other as the name suggests.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de}{dt} \tag{3.4}$$

The standard and parallel forms are mathematically equivalent, unlike the series form seen in 3.5, which resembles a pneumatic controller more closely. A series controller without derivative action is equivalent to the parallel form.

$$u(t) = K_p(e(t) + \frac{1}{T_i}\int_0^t e(t)dt)(1 + T_d\frac{d}{dt}) \tag{3.5}$$

Transfer functions The transfer function of the PID controller is found by performing the Laplace transformation of the time domain equations. The operator s is used, where $s = j\omega$. Equation 3.6 shows a PID controller on standard form in the Laplace domain.

$$u(s) = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{3.6}$$

The parallel form of the controller is found by inserting K_i and K_d, as in 3.4, and is shown in equation 3.7.

$$u(s) = K_p + \frac{K_i}{s} + K_d s \tag{3.7}$$

Equations 3.8 are the series form formulations, also called cascade formulation. The SIMC tuning rules give a controller on this form.

$$u(s) = \frac{k_p(T_i s + 1)(T_d s + 1)}{T_i s} \;=\; K_p(1 + \frac{1}{T_i s})(1 + T_d s) \tag{3.8}$$

If the process contains noise, and a controller with derivative action is used, it is often good practice to use a low pass filter on the derivative term. The noise gives a high derivative of the error signal e, which causes too much compensation from the controller. The filter helps with this problem and is added by replacing the derivative term with the following, for standard form:

$$\frac{T_d s}{\frac{T_d}{N}s + 1}$$

Where N is the filter constant, which needs to be chosen for each case.

Discrete form All electronic devices operate in the discrete time domain, and therefore the discrete time PID controller is important. Equation 3.9 shows the discrete time PID controller on the absolute form. The derivative term is discretized using forward Euler.

$$u_{i(k)} = u_{i(k-1)} + \frac{T_s K_p}{T_i} e(k) u(k) = K_p e(k) + u_{i(k)} + K_p T_d \frac{e(k) - e(k-1)}{dt} \tag{3.9}$$

Another discrete representation is given by 3.10. This is a controller on standard form and is the formulation that was used when performing simulations in this thesis. The controller state, z, needs to be initialized. These formulas are used together with equation 2.9, to simulate a closed-loop system with a SSM.

$$\begin{aligned} e_k &= r - y_k \\ u_k &= z_k + K_p e_k - K_p T_d \frac{y_k - y_{k-1}}{dt} \\ z_{k+1} &= z_k + dt \frac{K_p}{T_i} e_k \end{aligned} \tag{3.10}$$

Conversions The relationship between the parallel and ideal forms are given in the intro to section 3.3. When converting from serial to ideal form, the factor $f = 1 + T_d/T_i$, and the following formulas are used:

$$K_p^{ideal} = K_p f, \quad T_i^{ideal} = T_i f, \quad T_d^{ideal} = T_d / f \tag{3.11}$$

3.4 Control structures

Figure 3.5 shows the standard control loop, including input disturbance v, and all the symbolic names which are used when discussing control loops in this thesis. In some cases, it is beneficial to use different structures for the control system. This is useful if there is more than one measurement available. Two common control structures are cascade control and feedforward control. These control structures can also be combined in various ways.

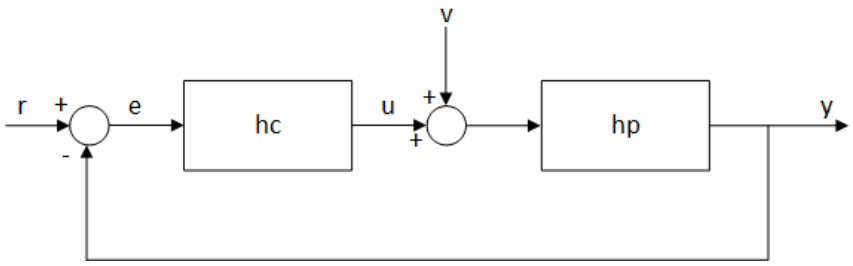


Figure 3.5: Standard negative feedback control loop, including input disturbance, v

3.4.1 Cascade loop

Figure 3.6 shows the block diagram for a cascade control loop. This type of set up can be useful when the system has large time constants or dead time. A cascade control loop consists of nested control loops, where the inner loop is referred to as the secondary loop. The inner loop acts as the actuator to the outer/primary control loop. This setup may give tighter control.

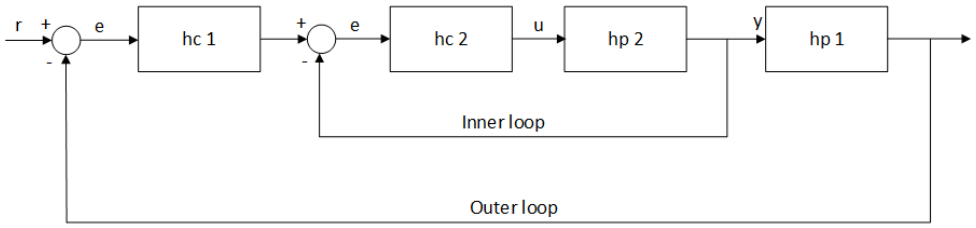


Figure 3.6: Cascade control loop, block diagram

3.4.2 Feedforward loop

In feedforward control loops, the process disturbance is measured and used to compensate, using a feedforward compensator, denoted h_{cff}. The goal of feedforward control is to compensate for the process disturbance before it creates a control error. An example of how a feedforward control loop can look is given in figure 3.7.

3.4.3 Two degrees of freedom PID controller

In 2 degree of freedom controllers r and y have different signal paths, as see in figure 3.8. This gives more flexibility to satisfy design compromises, like fast disturbance rejection without increased overshoot in setpoint tracking [16].

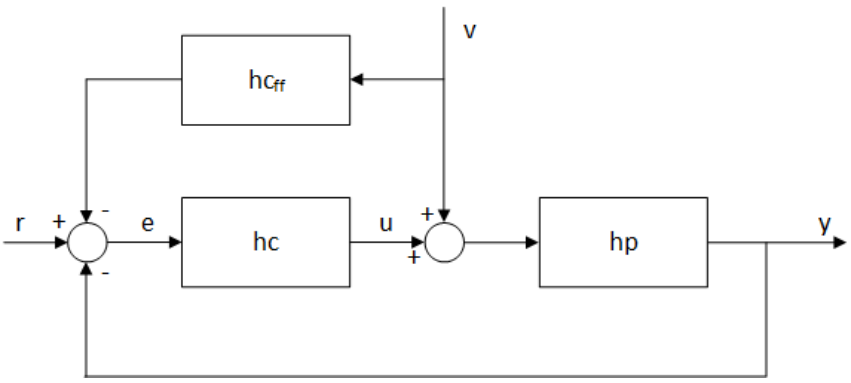


Figure 3.7: Feedforward control loop, block diagram

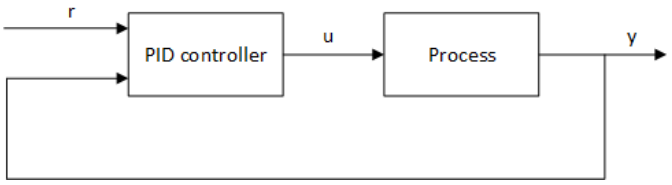


Figure 3.8: 2 degree of freedom controller principle

Equation 3.12 is the transfer function for the 2DOF PID controller on standard form. The parameters b and c are adjustable weight coefficients.

$$u(s) = K_p(be + \frac{1}{T_i s}e + \frac{T_d s}{\frac{T_d}{N}s+1}ce) \tag{3.12}$$

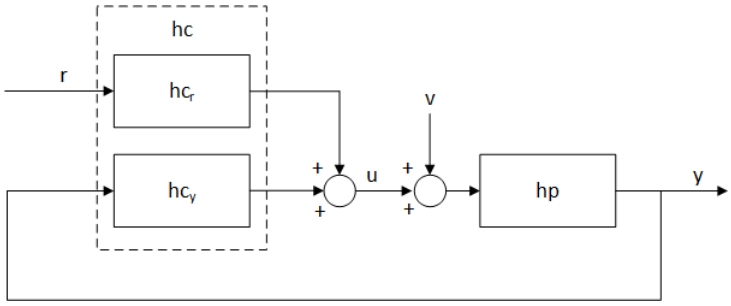


Figure 3.9: Feedback control loop with a 2DOF controller, block diagram

Equations 3.13 are the closed loop transfer function equations for setpoint tracking and disturbance rejection based on block diagram in figure 3.9 [17]. These transfer functions show how the extra flexibility is added, as the controller is divided into h_{cr} and h_{cy}.

$$\frac{y}{r} = \frac{h_p h_{cr}}{1 - h_p h_{cy}}, \quad \frac{y}{v} = \frac{h_p}{1 - h_p h_{cy}} \tag{3.13}$$

Figure 3.10 shows a comparison between a PID controller on standard form and a 2DOF PID controller. Both are tuned using pidtune and 3rd order plant model. The 2DOF controller has better setpoint tracking while maintaining the same disturbance rejection as the standard PID controller.

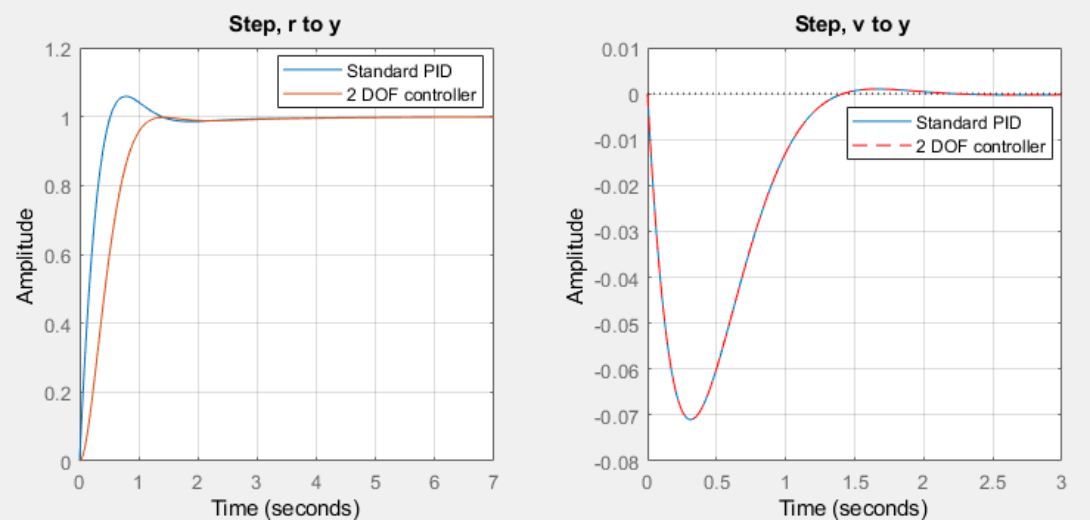


Figure 3.10: Step response, 2DOF controller compared to PID controller (left:SP tracking, right:disturbance rejection)

3.5 PID tuning goals

One of the reasons why there exist so many different methods for PID tuning, despite the low number of tuning parameters, is the fact that the choice depends on the desired characteristics of the system. A range of tuning parameters can be used to stabilize the system, but still give very different behavior. This means that there is not one choice of parameters which can be said to be correct. Different tuning methods can help to achieve different system behavior, and some tuning methods also have tuning parameters that can be chosen for this purpose. The main trade-off is between high controller gains for performance and low gains for robustness and less input usage. The following properties are desired to obtain in the tuned control loop:

- Setpoint tracking - This is the controller's ability to track a changing setpoint
- Disturbance rejection - The controller's ability to keep the output at the setpoint despite disturbances to the process
- Robustness - The ability to handle uncertainty

- Low input usage - High performance might demand a high degree of input usage, which can be expensive and wear out or damage the actuator

The transfer functions in 3.14 are used to simulate setpoint tracking, disturbance rejection, and controller effort. These transfer functions describe the feedback control loop in figure 3.5. Setpoint tracking is often used for testing, as changing the setpoint and observing the response is easy to do. However, good disturbance rejection is usually more important, as many processes operate at a fixed setpoint. Section 3.6 explains how tuning goals can be measured and quantified.

$$\frac{y}{r} = \frac{h_p h_c}{1 + h_p h_c}, \quad \frac{y}{v} = \frac{h_p}{1 + h_p h_c}, \quad \frac{u}{r} = \frac{h_c}{1 + h_p h_c} \tag{3.14}$$

3.6 Performance and robustness

Robustness and performance are contradictory qualities in a system, and high performance generally gives less robustness. Therefore it is important to quantify these terms, in order to tune a controller which satisfies both. There are numerous ways of quantifying robustness and performance, and the measures used in this report are explained here.

Robustness is the system's ability to handle uncertainties, i.e. how much the controlled process can vary from the nominal. Robustness is important because the models used to tune controllers do not always represent the real process accurately. Components degradation can also cause a change in process behavior over time.

Performance in control engineering is to make the output, y, behave in a desired manner. What the desired system behavior is, must be determined for each case. An ideal control loop has a fast response and no overshoot, requiring little controller effort.

3.6.1 Performance measures

Performance measurements based on a step response is explained in the list below and illustrated in figure 3.11. Settling time and rise time quantify the speed of response, while overshoot and steady state error give the quality of the response.

- Settling time - The time it takes the output, y, to settle within 5% of the final, steady state value
- Rise time - The time it takes the output, y, to travel from 10% to 90% of the final, steady state value

- Overshoot - The peak value of the step response divided by the final, steady state value. Given in percent
- Steady state error - The difference between the final steady state value and the desired final value

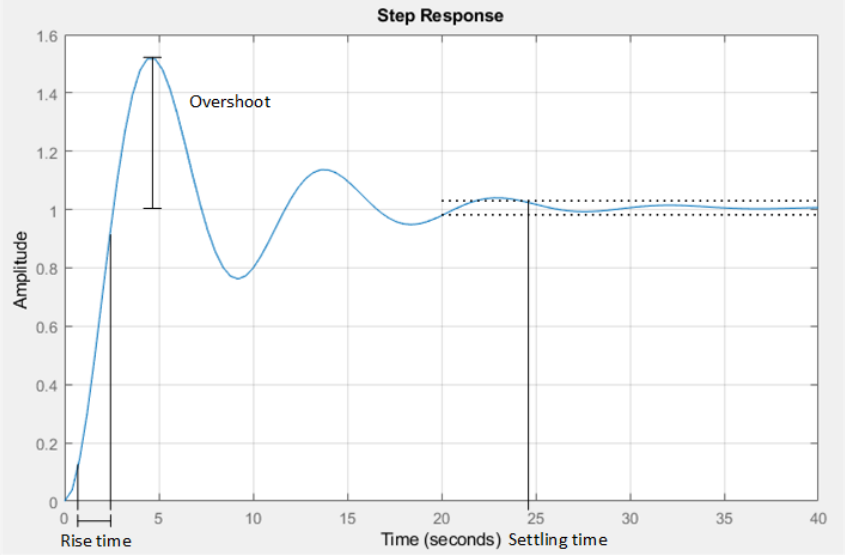


Figure 3.11: Performance measures read from step response

A performance measure which is extensively used in the report is the integrated absolute error, defined in 3.15. It is the sum of the error, e, over a time interval. A low value of IAE is desired [18]. The IAE value is calculated for step responses at both r and v, to measure SP tracking and disturbance rejection.

$$IAE = \int_0^\infty |e| dt \tag{3.15}$$

To measure input usage/controller effort, the total value defined by 3.16, is used. A high value of TV means that the controller uses more input to adjust the process, making it more aggressive. A high value of TV may also cause more stress on the actuator and be expensive in economic terms.

$$TV = \sum_{k=1}^{\infty} |\Delta u_k| \tag{3.16}$$

where $\Delta u_k = u_k - u_{k-1}$

3.6.2 Robustness measures

Robustness can be measured using margins, which says something about how much uncertainty in different parameters, the control loop can withstand without going unstable. Gain margin is the amount of gain the loop can be increased with before the system goes unstable. Typically, it is desired to have $GM > 2$. Phase margin is the amount of phase lag that can be added to the loop before the system goes unstable, and generally it is desired to have $PM > 30^o$ [19]. RHP zeros and time delays cause phase lag, and therefore PM is related to the maximum time delay error, $d\theta_{max}$, defined by 3.17. $d\theta_{max}$ is the maximum time delay that can be added to the system before causing instability.

$$d\theta_{max} = \frac{PM}{\omega_c} \tag{3.17}$$

Gain and phase margins are found from the bode plot of the loop transfer function, as shown in figure 3.12. If the magnitude is zero when the phase is -180^o, the closed loop is unstable. The gain margin is the distance between the actual magnitude and zero magnitude when the phase is -180^o, and the phase margin is the distance between the phase and -180^o when the magnitude of the gain is zero. The frequency when the gain is zero is called gain crossover frequency, ω_c, and the frequency when the phase is -180^o is called phase crossover frequency, ω_{180}.

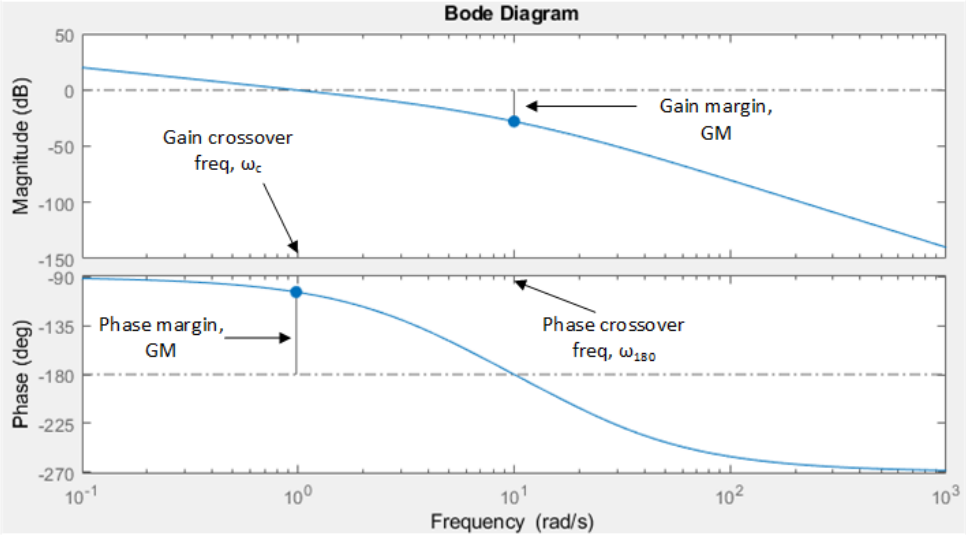


Figure 3.12: Gain and phase margin explanation using a Bode plot

Another robustness measure is M_s, which is the maximum peak of the sensitivity function, defined in 3.18. The magnitude of M_s should be less than about 2 (6dB). The smallest distance between $h_c h_p$ and the -1 point is M_s^{-1} and therefore smaller value of M_s gives more robustness [19]. A good value for M_s is 1.59, and this is around the point where

IAE values are the lowest [20]. M_s also says something about the performance. Another variant is M_{st}, which is given by $M_{st} = max(M_s,\ M_t)$, and 3.18.

$$M_s = max_\omega|S(j\omega)|, \quad M_t = max_\omega|T(j\omega)| \tag{3.18}$$

Where $S = 1/(1+h_p h_c)$ and $T = h_p h_c/(1+h_p h_c)$

3.7 Other types of controllers

Despite its popularity, PID controllers have some limitations. They are generally not suitable for MIMO control, and since they rely on fixed parameters, control of non-linear systems can be challenging. This section briefly mentions some of the alternative control strategies. It is worth mentioning that it is possible to use PID controllers for non-linear systems as well, if utilizing gain scheduling. The PID controller can be used with MIMO systems if the degree of interaction between the various inputs and outputs are low.

Pole placement A simple method for designing a controller in the state space domain is pole placement. If all state variables are known to the controller at all times, it is possible to place the closed-loop poles at any desired location. In practice, there are limits to how much the process dynamics can be changed. This means that unrealistic pole placement can cause the actuator to saturate. This can be done in Matlab using the place function to find k when $u = -kx$. The closed loop poles are equal to the eigenvalues of the matrix (A-Bk). Pole placement techniques can also be used in the frequency domain and can also be modified to produce a PI or PID controller [21].

LQR The LQR controller is optimal in the sense that it minimizes the cost function in 3.19. The cost function is used to find an optimal gain matrix, k, which minimizes J when $u = -kx$. Q and R are weights, used to favor performance or input usage. The state and input values are squared, which makes J a quadratic function with an absolute minimum. This type of controller works well with MIMO systems. [22].

$$J = \int_0^\infty (x^T Q x + u^T R u)dt \tag{3.19}$$

MPC The MPC uses a plant model to predict the best choice of u over a finite time period to reach the desired state. This controller also uses optimization to determine the best choice of u. When the algorithm has determined the best u for each time step in the prediction horizon, the first value for u is applied to the system. For each discrete time step, the optimal values for u over the horizon are predicted, then only the first value is applied. This controller works with MIMO systems, nonlinear systems and it is possible to impose constraints on both x and u.

Fuzzy controller The fuzzy controller utilizes fuzzy logic, where values are tied to linguistic variables. Each input is given a degree of membership to different properties, for example, "70% warm" based on their value and predefined membership functions. A set of powerful if then statements are then used to determine output values. This controller can deal with MIMO and nonlinear systems [23].

4 PI and PID controller tuning methods

This chapter presents some commonly used methods of tuning PID controllers. If nothing is specified, the plant models used in the examples are randomly generated state space models. Each section presents a set of tuning rules and contain a summary of the main advantages and disadvantages, as well as outlining how they can be applied to SSM. In many cases, tuning rules are controller parameters given as a function of the process describing variables. Suggestions for obtaining these variables from SSM are given in section 2.4. One goal for a set of tuning rules can be to give stable closed-loop systems for all processes. This is hard to achieve, but some methods are very versatile, for example, Matlab pidtune and mftun. A methods ability to work with many processes has been one of the main focuses when evaluating and comparing tuning rules.

The tuning rules can be categorized as in ”PID control in the third millennium” [10]. It is possible that some tuning rules fall under several of these categories. The rules in all categories generally require a process model of some sort, except ultimate cycle methods.

PRC tuning rules PRC methods uses a step response to identify process parameters, as explained in section 2.4.1. Examples of such tuning rules are the ZN PRC method in section 4.3.2 and the Cohen-Coon rules in section 4.4. These process parameters can be found from SSM, as suggested in section 2.4.

Ultimate cycle tuning rules These methods are based on recording process parameters when the system is brought to marginal stability. The closed-loop system is brought to marginal stability by using a P-controller and increasing the gain until the system starts to oscillate. The ZN closed loop method in section 4.3.1 uses the UC parameters, and auto-tuning explained in section 4.7 usually utilizes UC. When a system is marginally stable, the poles are on the imaginary axis.

Optimization based tuning rules These kinds of methods are used to find controller gains that minimize different performance and robustness criteria. This requires initial tuning parameters and an optimization algorithm. A weakness is that numerical optimization cannot guarantee to find a global minimum.

Direct synthesis tuning rules Controller design based on a process model and some specific closed loop criteria. These criteria might be pole placement, GM, and PM. DS design can, for example be used to favor SP tracking or disturbance rejection [24]. Note that this type of controller design does not always produce a controller with a PI or PID structure. IMC methods and DS methods are closely related and in many cases, identical.

Tuning rules for robustness Tuning rules that are formulated to achieve robust performance and/or robust stability.

4.1 Matlab pidtune

The Matlab function pidtune uses Mathworks own patented PID tuning algorithm [25]. The algorithm is very robust and can handle any type of discrete or continuous time model. The model can be both stable or unstable, integrating and contain any type of time delay. The function can also be used to tune multiple controllers at a time, by inputting an array of models. A graphical tool exists as well, utilizing the same tuning algorithm, which can be opened with the command pidTuner.

4.1.1 Mathworks PID tuning algorithm

The algorithm aims to stabilize the loop while balancing two measures of performance, set-point tracking, and disturbance rejection. The idea behind the algorithm is to make it easy for the user to specify known measures of stability and performance and tune the controller based on that. These are terms that can be difficult to relate directly to the controller gains. Another aspect is that many modeling applications let users create system models that are not restricted to a particular order. These higher order models may be difficult to work with, and Mathworks aims to make this easy.

Figure 4.1 is a flowchart found in the patent [25], describing the main steps used to obtain parameters. The flowchart is general, and all steps do not apply to all situations. For example, the linearization steps when working with transfer functions. The figure also shows that if the specification from the user cannot be met, the algorithm modifies the specification. Based on information in the patent, the following can be said about the process of tuning, which is step 6 in figure 4.1. To be able to determine the controller parameters based on specified PM and ω_c, the parameterization of the controller in 4.1 is used. ω_c is either user-specified or selected based on the natural frequency of the plant.

$$C(s) = \frac{\omega_c}{s}\left(\frac{sin\phi_z s + \omega_c cos\phi_z}{\omega_c}\right)\left(\frac{sin\beta s + \omega_c cos\beta}{sin\alpha s + \omega_c cos\alpha}\right) \tag{4.1}$$

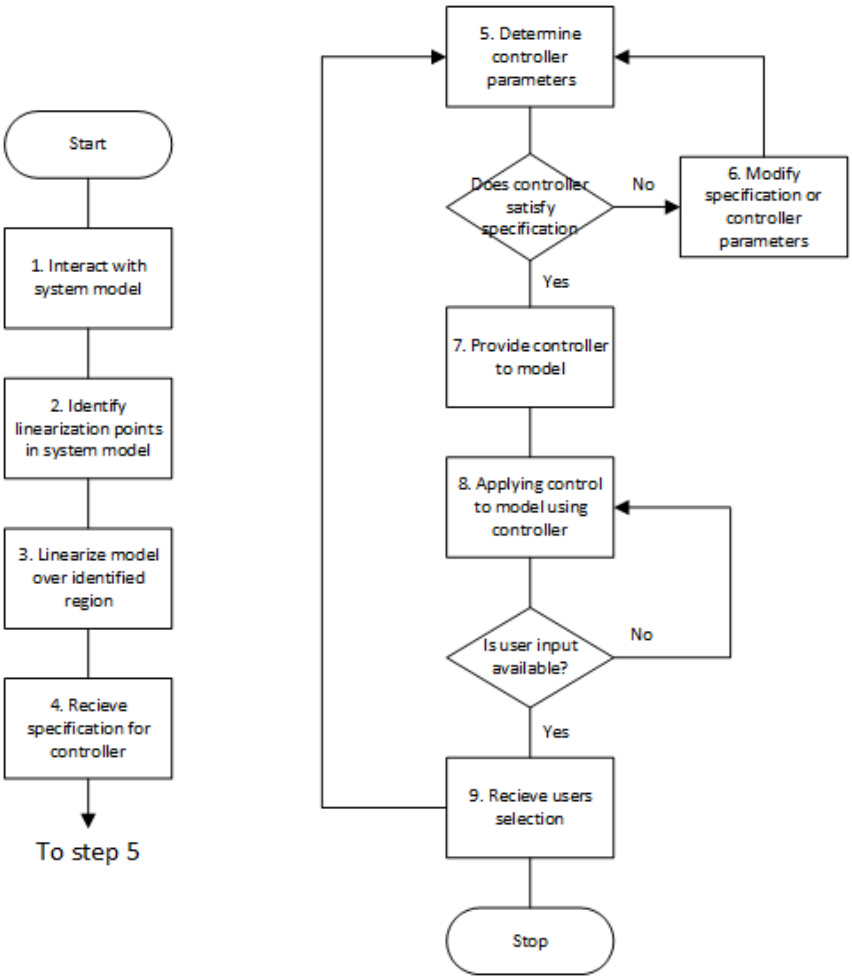


Figure 4.1: Flowchart for the pidtune algorithm, from patent [25]

Where crossover frequency ω_c and phase margin θ_m are fixed values and α and β are free variables in the range:

$$0 < \alpha < \beta < 90$$
$$\Delta_\phi - 90 < \beta - \alpha$$

At first, an initial guess for α and θ are used, then a plurality of the free parameters are identified. Gridding technique is used to discard values of α and β that violates the constraint or fail a Nyquist stability test. The patent states that gradient descent optimization technique is used to search through the free parameters, and choose the parameters that produce the smallest value of an objective function that includes a sensitivity function and a complementary sensitivity function. A known robustness measure based on

these transfer functions is the M_{st} value in 3.18. The optimization problem given in the patent is stated as:

$$min\ F = max_{\omega} max(|S(s)| - 2, |T(s)| - T_{max}, T_{min} - |T(s)|)$$

Subject to:

$$T_{min}(\omega) = \frac{1}{max(1, \omega/\omega_0/1.5))}$$
$$T_{max}(\omega) = \frac{1}{max(1, \omega_0/\omega/1.5))}$$

Based on this it can be said that the algorithm uses TF to tune controllers which minimizes a performance criteria related to M_{st}, which satisfy a stability test and gives prescribed PM.

4.1.2 Use of pidtune Matlab function

```
C = pidtune(sys, type/C0, (wc), (opts))
```

There are two mandatory arguments, sys, which is the plant model, and ”type” or c0, which is the specification of the desired controller. The controller can be specified with a text string, for example, ’PI’, or by passing an object, c0, which is an initial controller on the desired form. C0 can be a transfer function or a pid object, created using Matlab functions pid or pidstd. Pidtune can also be used to tune 2DOF controllers. By only specifying plant and controller the default settings are ”balanced” design focus and phase margin of 60 degrees.

To specify additional options, the pidtuneOptions function is used to create the object, opts. 3 options can be specified by the user:

- PhaseMargin - Used to specify the desired phase margin
- DesignFocus - Used to specify the desired design focus, ’reference-tracking’, ’disturbance-rejection’ or ’balanced’ which is the default setting
- NumUnstablePoles - Used to specify the number of RHP unstable poles in the plant

Besides, the desired crossover frequency, ω_0, can be specified directly in the pidtune handle. The following sections provide examples of different characteristics obtained using different options.

Design focus

Using the design focus option enables the user to prioritize reference tracking or disturbance rejection while keeping the same crossover frequency. Figure 4.2 shows the closed loop response using different design focus. The plant is a 5th order process, and a PID controller is used. The margins and controller parameters for the tuned controllers can be seen in table 4.1. The largest phase margin is obtained by favoring SP tracking.

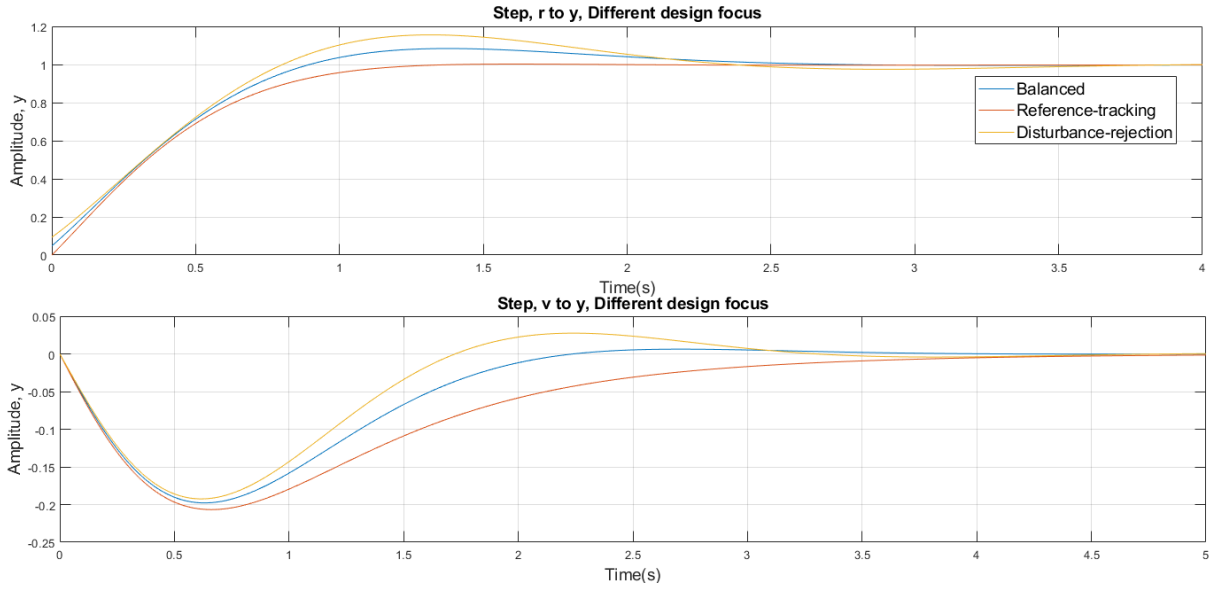


Figure 4.2: Comparison of SP tracking and disturbance rejection with different design focus, using pidtune

Table 4.1: Comparison of controller parameters when specifying design focus

Setting	K_p	T_i	Td	ω_c	PM	GM
”Balanced”	-2.32	0.511	0.0365	2.3750	69.2156	∞
”SP tracking”	-2.56	0.819	4.21e-05	2.3750	78.4312	∞
”Disturbance rejection”	-2.01	0.343	0.0858	2.3750	60.0	∞

Phase margin

Figure 4.3 shows the closed loop step responses when specifying different phase margin. Specifying a lower phase margin gives more aggressive controllers with higher controller gains. Resulting margins and parameters are given in table 4.2.

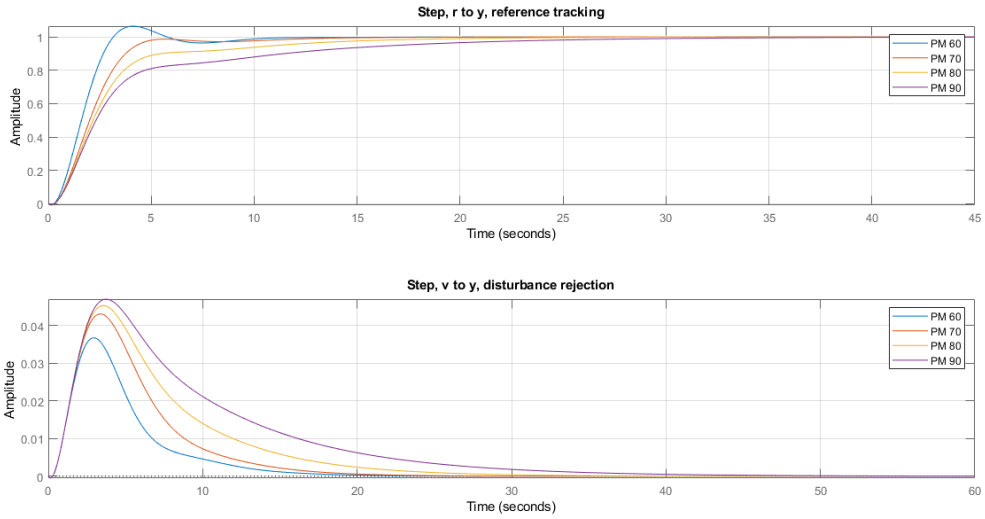


Figure 4.3: Comparison of SP tracking and disturbance rejection when specifying PM, using pidtune

Table 4.2: Comparison of controller parameters when specifying PM

Setting	K_p	T_i	ω_c	PM	GM
”PM 60”	17.8	3.31	2.73	60.0	8.71
”PM 70”	12.5	3.26	2.72	70.0	12.33
”PM 80”	11.7	4.14	2.81	80.0	13.93
”PM 90”	11.3	5.47	2.89	90.0	15.15

Crossover frequency, ω_c

Crossover frequency can be specified directly in the pidtune handle, shown in section 4.1.2. The gain crossover frequency, ω_c, is related to the control bandwidth of the system, which is the highest frequency where the system output tracks an input sinusoid in a satisfactory manner [26]. This means that bandwidth measures the speed of response and that higher ω_c gives a quicker response, while low ω_c favor robustness. The following example is with a PID controller tuned based on a 3rd order process. Figure 4.4 shows the closed loop step responses of 4 different systems with different crossover frequencies. Table 4.3 shows all margins and parameters for the systems. As expected the system with the highest ω_c has the highest gains and thus lowest margins.

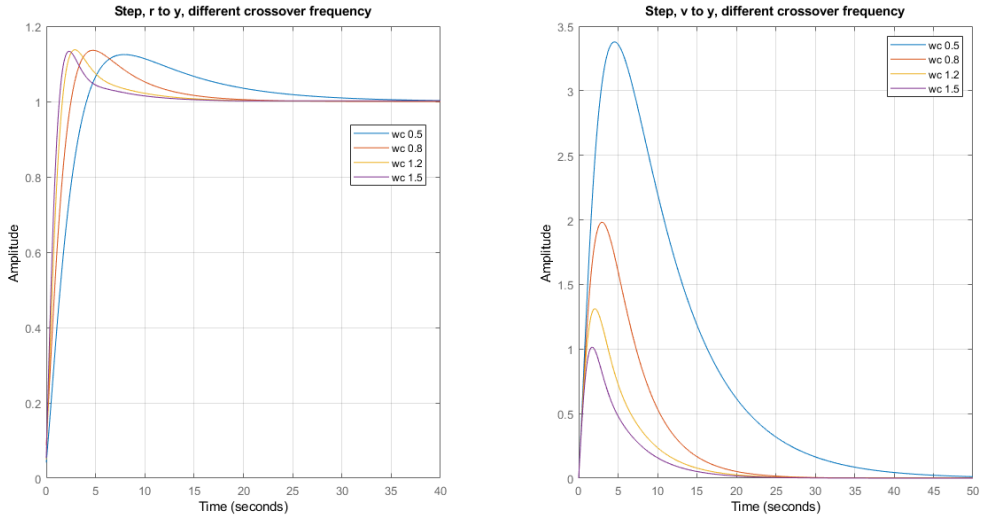


Figure 4.4: Comparison of SP tracking and disturbance rejection when specifying different ω_c

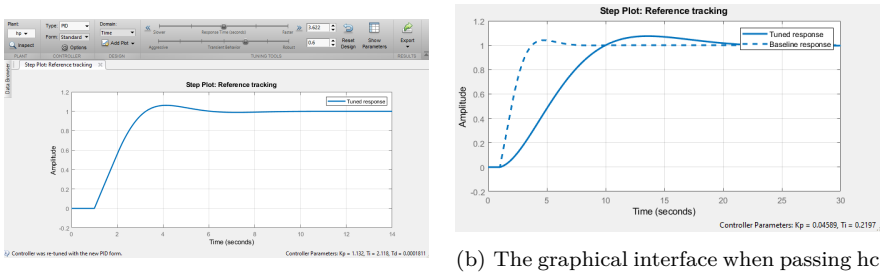
Table 4.3: Comparison of controller parameters when specifying ω_c

ω_c setting	K_p	T_i	Td	ω_c	PM	GM
0.5	0.236	10.3	0.16	0.5	74.2	∞
0.8	0.403	6.02	0.207	0.8	74.3	∞
1.2	0.659	5.35	0.0687	1.2	69.2	∞
1.5	0.878	5.08	0.056	1.5	69.5	∞

4.1.3 Graphical interface

Matlab also has a graphical tool for tuning PID controllers based on the same algorithm, the tool can be opened with the ”pidTuner” command. If the app is launched without arguments, a plant model must be imported later. Type of controller can be specified in the handle or the user interface. If a controller is passed as an argument when opening the app, this controller is used as a baseline. This means that the controller design is compared to hc, as shown in figure 4.5b. The layout of the pidTuner app can be seen in figure 4.5.

When opening the app, a controller is automatically tuned. This controller can be modified by manipulating 2 ”sliders”, response time and transient response. Adjusting the response time towards faster response gives larger K_p and more overshoot. The response time setting uses seconds as a unit and is approximately the time the system uses to reach 90% of the final value. Transient means that a system's response to an input signal is temporary before the system reaches steady state [4]. This means that adjusting the transient response changes how the system reacts to changes in inputs or disturbances.



(a) The graphical interface layout

(b) The graphical interface when passing hc argument

Figure 4.5: Matlab pidtuner graphical interface

The transient behavior can be adjusted to become more robust or more aggressive. The settings response time, and transient behavior are interconnected and changes in one might affect the other, and there are also limitations to what responses are possible. For example, with a first-order system with a PI controller, if the response time is set slow (10sec), the transient behavior setting barely changes the system dynamics.

4.1.4 Cases where pidtune fail

A test was performed to find models where pidtune fail to give a stabilizing controller. A loop generating different random stable state space models were used for this purpose. For each model, a PI controller was tuned using pidtune, and then allmargin was used to test for stability. This process was repeated until 30 unstable systems were found, for both 2nd, 3rd, 4th and 5th order systems. A total of 70 213 different systems was tested to find these 120 unstable plant models. Some characteristics of these plants are given here:

- 2nd order plants
  - 27 double integrating
  - 3 with large value complex poles close to the imaginary axis
  - All can be stabilized with pidtune, using PID controller
- 3rd order plants
  - 26 has a double integrator
  - 2 has an integrator and complex poles close to the imaginary axis
  - 2 has complex poles close to the imaginary axis
  - 2 can be stabilized with pidtune using PID controller
  - 1 can be stabilized with PI controller, using optimization tuning

- 4th order plants
  - 18 has a double integrator
  - 11 has a double integrator and complex poles
  - 1 has an integrator and complex poles
  - 3 with large value complex poles
  - 0 can be stabilized with pidtune and PID controller
  - 1 can be stabilized with PI controller, using mftun
- 5th order plants
  - 11 has a double integrator
  - 1 has a double integrator
  - 18 has a double integrator and complex poles
  - 1 can be stabilized with pidtune and PID controller
  - 1 can be stabilized with PI controller using mftun, delta tuning, Cohen coon, SIMC or ZN PRC
  - 2 can be stabilized with PI controller, using optimization tuning

Figure 4.6 shows the open loop step responses for 3 of these systems, as seen they have integrating and/or oscillating behavior. This test was performed to find weaknesses in pidtune, and it seems that integrating systems cause the most problems. However, a large amount of systems had to be tested to find these, making pidtune a very robust tuning method.

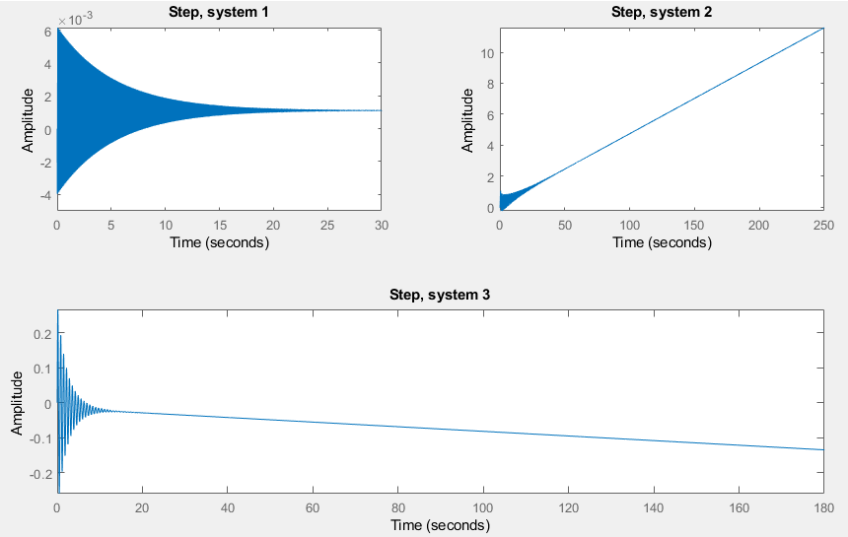


Figure 4.6: Examples of open loop step responses from systems where pidtune algorithm does not give a stabilizing controller

4.1.5 DIPTD systems and pidtune

As seen in section 4.1.4, pidtune may struggle with systems containing integrators and double integrators. When using pidtune with default settings on a DIPTD process the performance is less than that of other methods, as for example δ -tuning. Figure 4.7 shows setpoint tracking and disturbance rejection, for the default controller as well as attempts to increase performance by adjusting PM and ω_c. The plant has K=1 and $\theta = 2$, and the δ controller has default settings. As seen the performance can be improved but does not match δ -tuning.

Table 4.4 shows the results of the simulation. Lowering ω_c gives better disturbance rejection and high margins. The PM was set to 30, but the algorithm does not allow a lower PM than 48, which gives the best performance.

Table 4.4: Pidtune performance with DIPTD systems, using different settings

	PM	GM	M_s	IAEr	IAEv
Default settings	60	3.2972	1.5596	9.6188	1.2742e+05
PM 30	48	2.5315	1.8529	8.2691	7.7737e+03
ω_c 0.15	60	5.1940	1.3307	12.2790	8.2612e+04
delta tuning	30	2.1988	2.2790	9.3715	444.9121

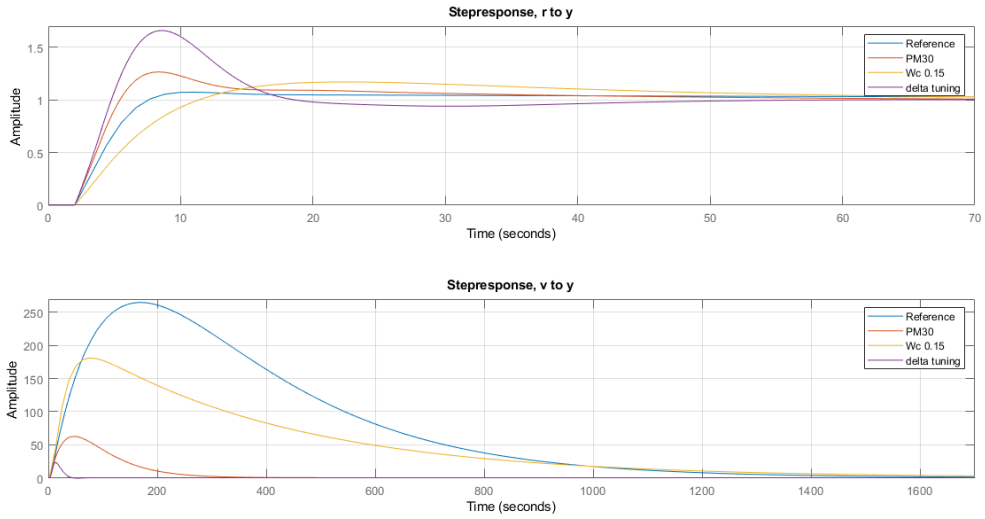


Figure 4.7: Step responses for SP tracking and disturbance rejection, comparing different settings for pidtune, delta tuning for reference

4.1.6 Summary pidtune

- Advantages
  - Simple to use, little knowledge required
  - Robust/flexible
  - Fast, despite optimization
- Disadvantages
  - A proprietary algorithm, requires a Matlab license
  - Require model
  - Bad performance with DIPTD processes, and other integrating systems
- Use with SSM
  - Pidtune works directly with state space models. The algorithm uses TF but pidtune has built-in capability for conversion and linearization

4.2 Delta tuning rules

The δ-tuning rules are presented in papers [27]–[30] by Di Ruscio and Dalen. The tuning rules ensure a prescribed maximum time delay error to time delay ration, denoted δ as in 4.2. The delta tuning rules contain two tuning parameters that must be specified, δ and $\bar{c}$. Method product parameter $\bar{c} = \alpha\beta$, is chosen based on desired robustness. Using values suggested in papers for δ and $\bar{c}$, gives tuning rules that are simple to use and remember, while still giving flexibility for more experienced users.

$$\delta = \frac{d\theta_{max}}{\theta} \tag{4.2}$$

The first paper from 2010 [27] presents tuning rules for a PI controller for IPTD processes, which has been extended to include tuning rules for PD/PID controllers for DIPTD processes [28]. Later, a PRC approach to the δ-tuning rules was presented [29], [30].

4.2.1 PI controller for integrating plus time delay

In the paper from 2010 where the delta tuning rules for IPTD are introduced, it is shown that all tuning rules for this type of plant can be formulated as in 4.3. An approach to obtain α and β, based on parameters δ and $\bar{c}$, are then presented.

$$K_p = \frac{\alpha}{K\theta}, T_i = \beta\theta \tag{4.3}$$

The following formulas are used to find α and β:

$$f = \frac{1+\sqrt{1+\frac{4}{(\bar{c})^2}}}{2} = \frac{1+\sqrt{1+\frac{4}{\bar{c}^2}}}{2}$$
$$a = \frac{arctan(\sqrt{f}\bar{c})}{\sqrt{f}} = \frac{arctan(\sqrt{f}\bar{c})}{\sqrt{f}}$$
$$\beta = \frac{\bar{c}}{a}(\delta+1)$$
$$\alpha = \frac{a}{\delta+1}$$

where:

$f =$ Factor used to find a
$a =$ Factor used to find β and α
$\bar{c} =$ Tuning parameter $1.5 \leq \bar{c} \leq 4$
$\delta =$ Prescribed delay margin $1.1 \leq \delta \leq 3.4$

Choosing $\bar{c} = 2$ and $\delta = 1.6$ as suggested by Di Ruscio, the PI controller for IPTD processes are given by 4.4.

$$K_p = \frac{0.4}{K\theta}, \quad T_i = 4.995\theta \tag{4.4}$$

4.2.2 PD and PID controller for double integrating plus time delay

In 2017 the δ tuning rules were extended to include PD and PID controllers for DIPTD systems [28]. As seen in 4.5, a new parameter, γ, is included.

$$K_p = \frac{\alpha}{K\theta T_d}, \quad T_i = \gamma T_d, \quad T_d = \beta\theta \tag{4.5}$$

Where γ is the relative integral derivative time ratio. γ should be chosen such that $1 \leq \gamma \leq \text{inf}$. Choosing $\bar{c} = 2.5$ and $\gamma = 2.1$, gives the tuning rules in table 4.5.

In addition, there are 2 different ways of specifying the tuning parameter, δ:

- $DM = d\theta_{max} = \delta\theta$, the delay margin varies relative to the time delay, and the tuning parameter δ should be chosen in each case to get acceptable responses. The margins PM, GM, and M_s are constant when using this option.
- $DM = d\theta_{max} = \delta$, the second option is to specify a fixed DM, independent of the dead time, which gives varying margins. This is achieved by choosing $\delta = \delta/\theta$.

Table 4.5: Delta DIPTD tuning rules, using suggested settings

	k_p	T_i	T_d
PD	$\frac{0.549}{k\theta T_d}$		4.551θ
PID	$\frac{0.549}{k\theta T_d}$	$2.1T_d$	4.551θ

4.2.3 Delta tuning PRC method

The delta PRC tuning rules were introduced in 2018 [30], and uses metrics R and L, explained in section 2.4.1, to approximate a DIPTD model. The δ-tuning rules in section 4.2.2 are then used to obtain a PID controller. The gain and time delay in the DIPTD model are found using the following formulas:

$$K = \zeta \frac{R}{L}$$
$$\theta = \eta L$$

where ζ and η are chosen to be:

$\zeta = 1$ or 6
$\eta = 1/2\ \pi$

In most cases, choosing $\zeta = 1$ gives the best results.

4.2.4 Approximating processes as (D)IPTD using optimization

Dalen and Di Ruscio also present an optimization-based approach for approximating a IPTD or DIPTD model based on a step response [31]. This approximation method is referred to as δ-optimization in this report. This model is then used to obtain PID parameters, based on the δ-tuning rules. Matlab optimization function, fmincon, is used to find $\rho = [K \quad \theta \quad T_f]$ which minimizes the objective function J:

$$J = \frac{1}{N}(y-\hat{y})^T I (y-\hat{y})$$

With the constraints $A\rho \leq b$ and $l_b \leq \rho \leq u_b$, and the following parameters:

$$lb = \begin{bmatrix} c_1 \\ -inf \\ -inf \end{bmatrix},\ ub = \begin{bmatrix} c_2 \\ inf \\ inf \end{bmatrix},\ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 1 \end{bmatrix},\ b = \begin{bmatrix} c_2 \\ 1 \\ 1 \end{bmatrix}$$

c_1 and c_2 are the lower and upper bound for the gain and must be selected for each case. In the implementation used in this thesis, the bounds are chosen by checking R, to automate the method. If $R < 0$, $c_1 = -10$ and $c_2 = -0.1$, in all other cases $c_1 = 0.1$ and $c_2 = 10$. These bounds are close to the ones used in the paper presenting the method.

4.2.5 Mftun and megatuner

Mftun and megatuner are extensions to the δ-tuning rules that have been developed by Di Ruscio and Dalen. The methods are relatively new, and no papers have been published on them at this time. Mftun is a PRC method whereas the megatuner source code is closed to the public, and the basis is therefore unknown. Both of these functions aim to stabilize a large number of process models, thus competing with Matlabs pidtune.

Mftun has 2 ”paths”, and the first step is to check if any of the model's eigenvalues are zero. This implies an integrator in the system, and the PI controller is calculated using the delta rules for integrating processes in section 4.2.1. If else, a set of process describing variables based on a step response is calculated. The values obtained from the step response can be seen in figure 4.8.

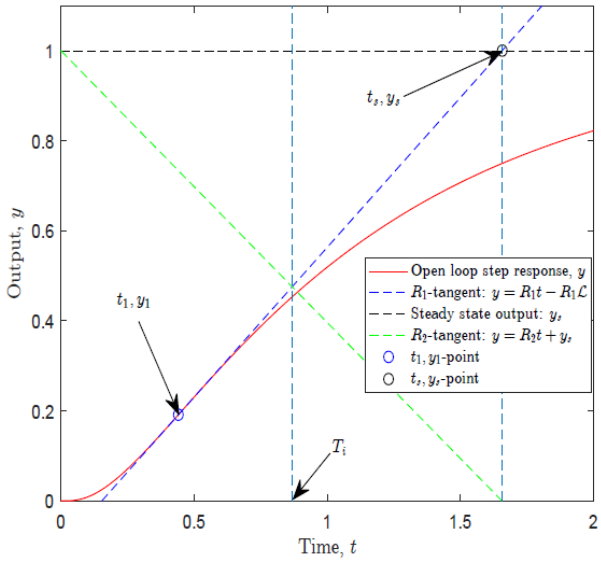


Figure 4.8: Graphically obtaned process describing variables obtained from step response and used by mftune [32]

The PI controller is then calculated using the following formulas:

$$K_p = \frac{1}{\rho R_1 T_1}$$

$$T_i = \frac{R_1 t_s}{R_1 - R_2}$$

Where ρ is a tuning parameter, and since there are no recommended values for ρ, a range of values are tested. A lower value of ρ gives more aggressive tuning.

Megatuner is currently available online in ver. 0.1, for Matlab [33]. The details of how the method work is not publicly available, but some assumptions can be made. As the method uses more time computing the controller parameters, with higher order models it can be assumed that it is not a PRC method. The time usage is displayed in figure 4.9. It is claimed that the method has been successfully tested on a million different random SSM, which gave rise to the name. The available version of Megatuner does not handle processes which contain integrators and can be seen as an alternative to the part of the mftun algorithm that handles the systems without integrators.

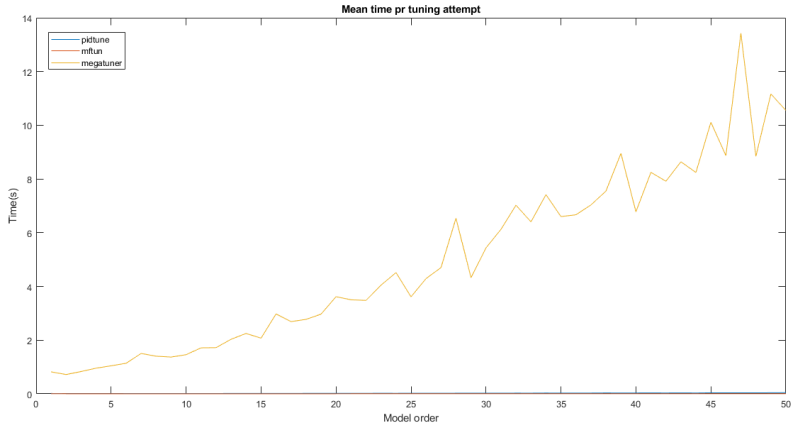


Figure 4.9: Evolution of Megatuner time usage for tuning PI controllers when model order gets higher

4.2.6 Summary delta tuning rules

The basis for the delta tuning rules is the PI controller for IPTD systems and PID/PD controller for DIPTD systems. In addition, 2 methods for approximating these types of systems based on step response data are suggested. One uses R and L to approximate a DIPTD model, the other uses optimization to approximate an IPTD or DIPTD model.

- Advantages
  - Simple and easy to remember if default tuning parameters are used
  - Tuning rules for integrating systems which can be hard to control
  - Gives good performance for DIPTD systems
  - mftun is very robust and only has one tuning parameter which makes it easy to use
- Disadvantages

  - Many approaches for implementation, can be hard to get an overview of the rules
  - Describing higher order systems using 2 parameters might not work in all cases
- Use with SSM
  - The SSM can be simulated in open loop to obtain Y, which then can be used with the δ-PRC method or the δ-optimization method
  - A IPTD model can be approximated using all 3 methods in section 2.4
  - A DIPTD model can be approximated using optimization described in 2.4

4.3 Ziegler and Nichols tuning rules

The Ziegler and Nichols tuning rules were published in the paper "Optimum Settings for Automatic Controllers", in 1942. 2 different methods for PID tuning was purposed, an ultimate gain method and a PRC method. These are some of the most popular tuning methods today, despite well-known disadvantages. There have been numerous suggestions to update the ZN tuning rules by for example Hegglund and Åstrøm [34], Tyreus and Luyben [35], and by Haugen [36]. The ZN tuning methods use two parameters to describe the process and provides simple formulas to calculate the parameters based on this. This makes the tuning simple, but it is not always enough to describe the process characteristics. ZN defined acceptable closed-loop stability using the amplitude ratio between peaks in y, following an input step. The ratio between the first and second overshoot/undershoot should be 1/4. This results in a controller with poor damping and robustness.

- Advantages
  - Simple and easy to remember
  - Useful when overshoot is no problem
- Disadvantages
  - No tuning parameter
  - Results in rather aggressive closed-loop response
  - Cannot be used for all processes, like DIPTD
- Use with SSM
  - Perform open loop simulation of SSM to obtain PRC values
  - For the closed loop method, find GM and $\omega_l 80$ to compute UG and UP

4.3.1 Ultimate gain method

The UG method is performed by bringing the closed-loop system to marginal stability and timing the oscillations. The procedure is to use a pure P controller and increase the gain until the system has sustaining oscillations. The gain required to make this happen is called ultimate gain. The ultimate period is the time of one period. These metrics are then used to calculate controller gains according to table 4.6.

Table 4.6: Ziegler and Nichols ultimate gain tuning rules

	k_p	T_i	T_d
P	$\frac{K_u}{2}$		
PI	$\frac{K_u}{2.2}$	$\frac{P_u}{1.2}$	
PID	$\frac{K_u}{1.7}$	$\frac{P_u}{2}$	$\frac{P_u}{8}$

In Matlab, the ultimate gain method can be implemented by finding the GM and ω_{180} by using margins. Ultimate gain and period are then:

$$K_u = GM$$
$$P_u = \frac{2\pi}{\omega_{180}}$$

4.3.2 PRC method

The original ZN PRC method uses the variables R and L to describe the process, but later the method has been converted to an alternative which use, gain, time constant and time delay instead i.e. a FOPTD approximation. The alternative using R and L has the advantage that it can be used to describe integrating processes. For self-regulating plants the FOPTD approximation may give a better description of the dynamics. The ZN PRC tuning rules for these two alternative methods are presented in tables 4.7 and 4.8.

Table 4.7: Ziegler and Nichols PRC method, using R and L

	K_p	T_i	T_d
P	$\frac{1}{LR}$		
PI	$\frac{0.9}{LR}$	$3.3L$	
PID	$\frac{1.2}{LR}$	$2L$	$0.5L$

Table 4.8: Ziegler and Nichols PRC method, using FOPTD approximation [12]

	K_p	T_i	T_d
P	$\frac{T}{k\Theta}$		
PI	$0.9\frac{T}{k\theta}$	3.33θ	
PID	$1.2\frac{T}{k\theta}$	2θ	0.5θ

4.4 Cohen-Coon tuning rules

The Cohen-Coon tuning rules was published in 1953 [37], 11 years after Ziegler and Nichols published their rules and are still used today. As with the ZN tuning rules, the Cohen-Coon rules are aimed to achieve a quarter decay ration. The Cohen-Coon tuning rules are a PRC method, which uses a FOPTD approximation. The controller is given by the formulas in table 4.9. Compared to ZN, this tuning method gives a faster response and works better on processes with long dead time compared to time constant. This difference is illustrated in figure 4.10.

Table 4.9: Cohen-Coon tuning rules [12]

	k_p	T_i	T_d
P	$\frac{1.03}{k}(\frac{\theta}{T}+0.34)$		
PI	$\frac{1}{k}(0.9\frac{T}{\theta}+0.083)$	$T(\frac{3.33(\theta/T)+0.31(\theta/T)^2}{1+2.22(\theta/T)})$	
PD	$\frac{1.24}{k}(\frac{\theta}{T}+0.129)$	$0.27T\frac{\theta-0.324T}{\theta+0.129T}$	
PID	$\frac{1}{k}(1.35\frac{T}{\theta}+0.25)$	$T(\frac{2.5(\theta/T)+0.46(\theta/T)^2}{1+0.61(\theta/T)})$	$\frac{0.37\theta}{1+0.19(\theta/T)}$

Summary of properties for Cohen-Coon tuning rules:

- Advantages
  - Useful when overshoot is no problem
  - Good in systems where θ is large compared to T
  - Tuning rule for a PD controller
- Disadvantages
  - No tuning parameter
  - Results aggressive closed loop response
  - Cannot be used for integrating processes
  - Complicated formulas
- Use with SSM

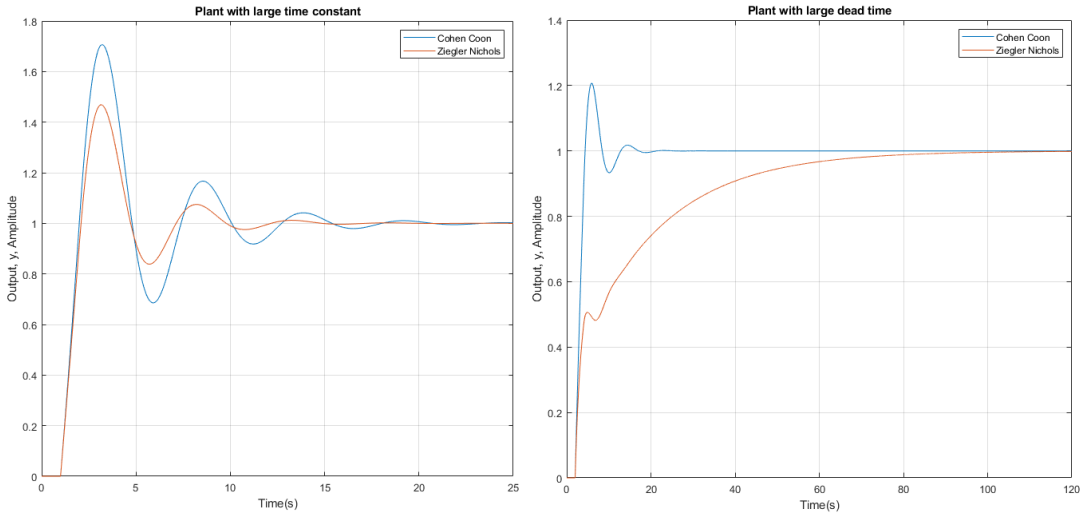


Figure 4.10: Input step response comparison between Cohen-Coon and ZN tuning, plant with large T to the left, plant with large θ to the right

– Simulate model and approximate values

4.5 Internal model control methods

IMC is a systematic method control system design. The IMC structure was introduced by Garcia and Morari in 1982 [38] and is sometimes referred to as lambda tuning. The IMC method uses a specific control structure to synthesize a controller mathematically, this structure is shown in figure 4.11. For most SISO systems the IMC methods lead to a PID controller [39]. Using the IMC method enables the user to specify the sensitivity function and the complementary sensitivity function, which directly determines the nature of the closed loop response.

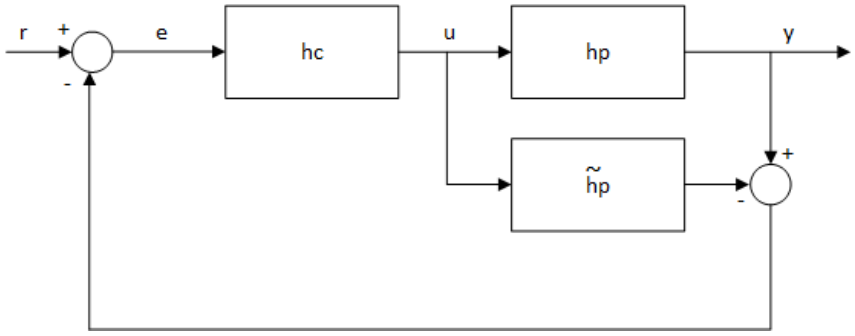


Figure 4.11: IMC control loop structure, block diagram

The first step is to factor the model $\tilde{hp}$ into two parts. One containing all the non minimum phase elements, being RHP zeros and time delays, denoted $\tilde{hp}_+$. The other is minimum phase and invertible, denoted $\tilde{hp}\_$. An IMC controller can then be specified as:

$$\tilde{h_c} = \tilde{hp}_-^{-1}$$

Step 2 is to add a filter, f(s), to $\tilde{h_c}$, such that $h_c = \tilde{hc}f(s)$, this is done to make the controller proper. The sensitivity function and complementary sensitivity function can now be express as functions of $\tilde{hp}$, h_c, and f. A common choice for f is:

$$f = \frac{1}{(\lambda s + 1)^n}$$

Depending on the factorization of h_p and the choice of f, the resulting controller parameters won't change for a given plant transfer function. For this reason, it is not necessary to do the math every time, and there are tables expressing controller gains as functions of λ, T, θ and other values, depending on process type [40].

4.5.1 SIMC

The simple internal model control tuning rules was presented by Skogestad in 2001[41] and has its roots in the IMC method from 1982. The motivation for developing SIMC was the poor margins given by the widely used Ziegler Nichols methods. This method has since become popular [42]. The SIMC tuning requires a FOPTD or a SOPTD plant model, which gives a PI controller or a PID controller, respectively. SIMC has later been extended for IPTD and DIPTD processes, and the iSIMC rules, presented in the following sections. The original SIMC rules for PI and PID controllers are given in table 4.10.

Table 4.10: SIMC tuning rules

	K_p	T_i	T_d
PI	$\frac{1}{K}\frac{T_1}{T_c+\theta}$	$min[T_1, 4(T_c+\theta)]$	
PID	$\frac{1}{K}\frac{T_1}{T_c+\theta}$	$min[T_1, 4(T_c+\theta)]$	$T_d = T_2$

Where T_c, the closed-loop time constant is the tuning parameter in the range: $-\theta < T_c <$ **inf**. Small values of T_c gives a fast response and disturbance rejection, while large values favor stability, robustness, and small input usage. In most cases, $T_c = \theta$ is a good choice, which gives a fast response with good stability margins. The SIMC rules give controllers on the series form.

iSIMC

In a paper from 2018 Skogestad suggest an improvement to the original SIMC rules, called iSIMC [15]. There are two suggested improvements in this paper, for a FOPTD process iSIMC suggests adding derivative action. For delay dominant processes, updated tuning rules for a PI controller is suggested. For processes where $T > \theta/2$ the benefits of iSIMC are marginal, and in some cases negative.

Table 4.11: iSIMC tuning rules

	K_p	T_i	T_d
PI	$\frac{1}{K}\frac{T+\theta/3}{T_c+\theta}$	$min[T+\theta/3, 4(T_c+\theta)]$	
PID	$\frac{1}{K}\frac{T_1}{T_c+\theta}$	$min[T_1, 4(T_c+\theta)]$	$T_d = \theta/3$

SIMC for double integrating processes

In 2016 Skogestad presented SIMC tuning rules for double integrating processes [43]. These rules for a PID controller are given by:

$$K_p = \frac{1}{K}\frac{1}{4(T_c+\theta)^2}$$
$$T_i = 4(T_c+\theta)$$
$$T_d = 4(T_c+\theta)$$

SIMC summary

- Advantages
  - Simple formulas to obtain controller parameters
  - Gives a controller with good performance and reasonable robustness
  - Tuning parameter for adjustment
- Disadvantages
  - Rely on FOPTD or SOPTD approximations and cannot be used for all systems
- Use with SSM
  - Convert SSM to TF, then use model reduction if needed and obtain parameters

  - Gain, time constants and θ can be obtained by simulating a step response from the SSM and then using techniques in section 2.4

4.6 Optimization based tuning

PID parameters can be chosen based on optimization. Two different methods are used in this thesis, and they are explained in 4.6.1 and 4.6.2. When tuning a PID controller based on optimization, the PID parameters are adjusted in such a way that an objective function is minimized. The objective function can be chosen freely to favor a certain type of system behavior. The optimization algorithm used in this thesis is fmincon in Matlab.

- Advantages
  - Can be made to work with any model
  - Closed loop characteristics can be determined by choosing an objective function and/or weights
- Disadvantages
  - Computational expensive
  - Global minimum of the cost function cannot be guaranteed
  - Optimization algorithm can fail
  - May require manual tuning of weights in each case to get good results
  - Require an accurate process model
  - Initial controller parameters must be guessed or found using a secondary tuning method
- Use with SSM
  - If the objective function is based on measures such as IAE and input usage, which can be found from a numeric simulation of SSM, this method can be used directly with SSM. In other cases, there might be a need to do conversions, for example if M_s is used.

4.6.1 Optimization tuning, using transfer functions

The first optimization technique tested in this thesis is referred to as Opt 1 or TF Opt. This implementation requires the SSM to be converted to a transfer function, as it uses the M_s value. It uses the Matlab step function to calculate IAE values. Defining $x = [K_p, \quad T_i, \quad T_d]$, the optimization problem can be stated as:

$$\hat{x} \; = \; min \; J(x) \; s.t \; l_b \leq x \leq u_b \tag{4.6}$$

where:

$$J \; = \; W_1 IAEr \; + \; W_2 IAEv \; + \; W_3 M_s$$

$$l_b = \begin{bmatrix} -\inf \\ 0.001 \\ 0 \end{bmatrix}, \; u_b = \begin{bmatrix} \inf \\ \inf \\ \inf \end{bmatrix}$$

$W_1 =$ Weight for SP tracking
$W_2 =$ Weight for disturbance rejection
$W_3 =$ Weight for robustness

4.6.2 Optimization tuning, based on SSM

In order to eliminate the use of transfer functions a second optimization approach, referred to as Opt 2 or SSM Opt, was tested. It was discovered during testing that TV and M_s values have a positive correlation. So in the second optimization method TV was used to improve robustness. The optimization problem is stated as in 4.6, but with the objective function:

$$J \; = \; W_1 IAEr \; + \; W_2 IAEv \; + \; W_3 TV$$

Another approach to increase the robustness could be to minimize the overshoot given by 4.7. This can also be calculated without involving TF.

$$OS \; = \; y_{max} - 1 \; = \; ||y||_\infty - 1 \tag{4.7}$$

4.6.3 Pareto optimal controller

A "Pareto-optimal" controller is often used as a reference controller. Pareto-optimality is used for multiobjective problems, performance, and robustness in this case, and means that no improvement can be made in performance without sacrificing robustness [20]. The optimization problem is given by:

$$\hat{x} \;=\; min\ J(x)\ s.t \begin{cases} l_b \leq x \leq u_b \\ M_s(x) - M_s^o = 0 \end{cases}$$

where:

$$J(x) = 0.5(\frac{IAEr(x)}{IAEr^o} + \frac{IAEv(x)}{IAEv^o}) \tag{4.8}$$

x = Controller parameters, $[K_p, T_i]$
M_s^o = Prescribed M_s (1.59)
$IAEr^o$ = Reference value from IAEr optimal controller
$IAEv^o$ = Reference value from IAEv optimal controller

4.7 Auto-tuning

Most commercial controllers have some sort of auto-tune capability. This is a "1 button push" approach to obtain controller parameters. This method requires the controller to be connected to the process. The auto-tuning algorithm executes some experiment to gain information on the process and then calculates the controller parameters.

A commonly used method used by auto-tuning algorithms is the relay tuning, as presented by Åström and Hägglund [44]. This method replaces the PID controller by an on/off controller, which will induce oscillations in the output. These oscillations have roughly the same period as in the Ziegler Nichols closed loop method, and ultimate gain can be calculated. This method is less time consuming than the Ziegler Nichols approach and thus, more suitable in an auto-tuning algorithm.

- Advantages
  - Easy to perform
- Disadvantages
  - Gives the user none, or very little information about the process, or the closed loop performance and robustness

- Use with SSM
  - Not applicable. This method is used on-line, with the real process
  - Relay tuning might be used with SSM by simulation, and switching between setting u=0 and u=1

5 Comparison of tuning methods

This chapter gives a comparison of the different tuning methods presented in chapter 4. Sections 5.1, 5.2, and 5.3 presents a comparison based on common system types, which are described in section 2.3. The values for K, θ, and T are taken directly from the transfer functions. This is done to give precise tuning based on the models and to illustrate the differences between the methods accurately. The optimization method used in these experiments is based on TF and is described as Opt 1 in section 4.6.1, the W_3 weight is set to zero. The results obtained from these tests can be compared to the results from sections 5.5 and 5.6, where random state space models are used.

5.1 Comparison of tuning methods based on first order plus time delay model

This section compares PI tuning rules based on the FOPTD model given by equation 2.17. Figure 5.1 shows the evolution of the tuning parameters K_p and T_i when the time constant, T, is increased, while K and θ are 1. These graphs demonstrate how differently controllers can be tuned for the same plant, while still giving a stable closed loop system. This illustrates why controller tuning is such a debated topic with numerous tuning rules. The most notable difference is that mftune and pidtune results in small values for K_p, which does not vary much when T is increased. The more aggressive setting for mftun shifts K_p higher but does not affect T_i. The other methods give increasing values for K_p as T are increasing. CC and ZN PRC method provide the highest, and also the same values for K_p. The values for T_i are steady for most of the methods, the exception being pidtune, mftun and megatuner which gives higher values for T_i with increasing T. The K_p curve for megatuner strengthens the notion that it is based on an iterating algorithm. The optimization tuning also results in nonlinear curves.

Figure 5.2 shows the M_s and TV values of the controllers. The methods with the steepest slope for K_p provides decreasing robustness. The graph also shows that the ZN UG method results in better robustness than the PRC method when T is more than 2. For faster systems, the ZN PRC method is best. The M_s graph shows that there are differences to which methods give the best robustness for systems with small time constants and larger time constants. Another key point from this graph is that SIMC offers a steady

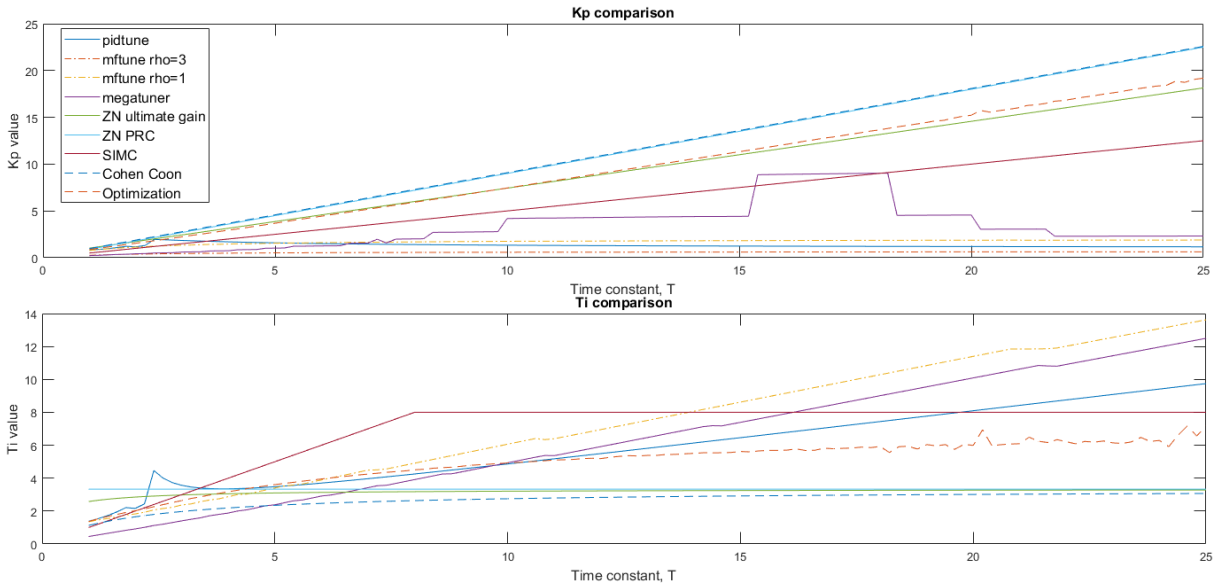


Figure 5.1: Evolution of controller parameters for different tuning methods when using FOPTD model, with an increasing time constant, T

value of M_s, which is close to the desired 1.59. The choice of ρ for mftune has less effect on robustness for systems with larger time constants. Comparing the graphs reveal a correlation between the input usage and the M_s value. They also show that megatuner is not designed to prescribe a specific value for M_s.

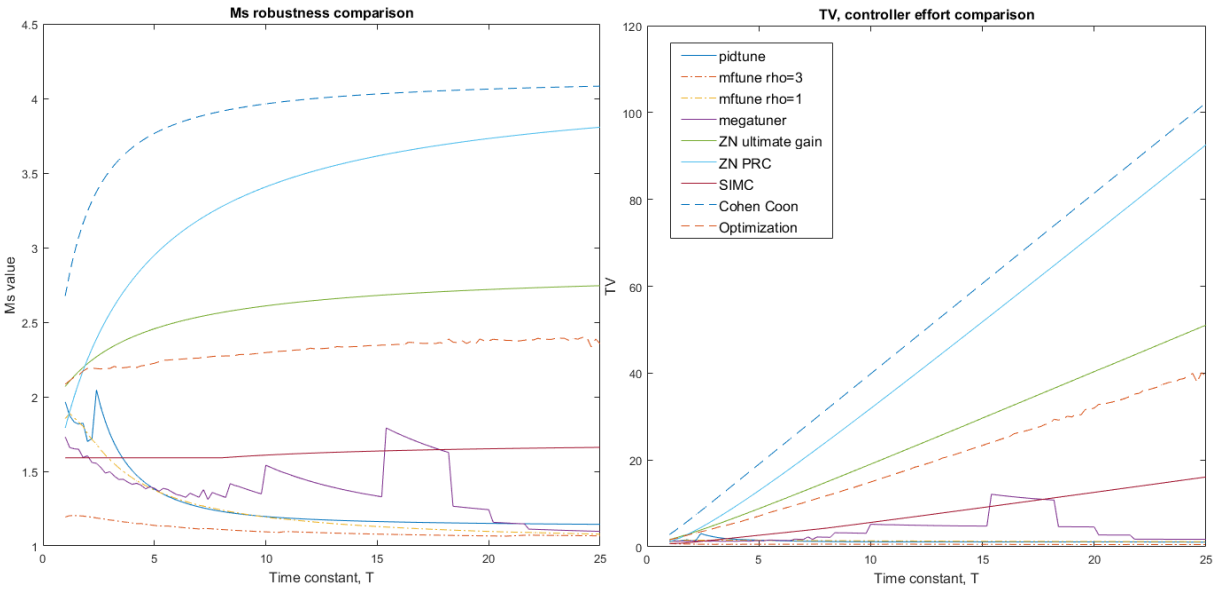


Figure 5.2: Comparison of M_s and TV for different tuning methods based on FOPTD model, with an increasing time constant, T

A common way of comparing tuning methods is to plot the performance, J, against the

robustness, M_s, where J is given by 4.8. Figure 5.3 shows the trade for some of the methods tested in this section. The SIMC curve is obtained by adjusting Tc between 0 and 3, and the default value of $\boldsymbol{Tc = \theta}$ is marked. Optimization gives the best performance while keeping M_s less than 2 in both cases, while SIMC is second best. The optimization curve is obtained by adjusting the M_s weight between 0 and 4. The plot also shows how ZN PRC changes characteristics from being robust and slow to being aggressive.

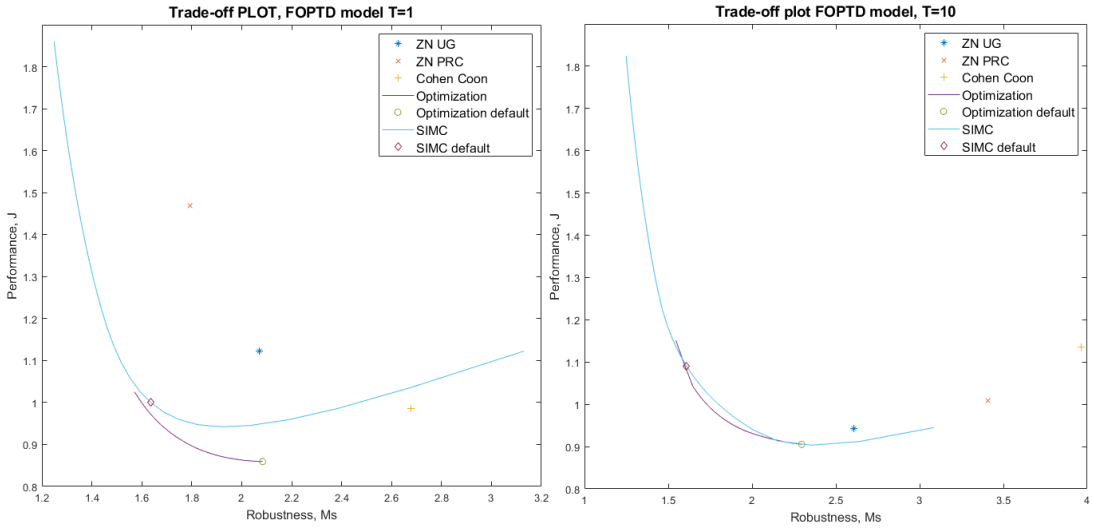


Figure 5.3: Trade-of plots between robustness and performance, comparing different methods based on a FOPTD model with T=1 and T=10

Figure 5.4 shows that the high robustness scores of pidtune and mftun come at the cost of performance, as both present higher values for IAE. The more aggressive setting for mftune give better performance without sacrificing robustness, especially with high values of T. The rest of the controllers give a similar performance for both disturbance rejection and SP tracking.

Figure 5.5 shows the step response from r to y and v to y for the different tuning methods for a FOPTD model with T=2. Cohen-Coon and both the Ziegler Nichols methods give aggressive dynamics, with oscillations. SIMC provides good balance between robustness and performance, with good responses for both SP tracking and disturbance rejection. Mftune give the best disturbance rejection in a sense that it is the fastest without giving any oscillations.

Table 5.1 contains the mean values of the calculated scores presented in this section. The main point here is that pidtune and mftune, favors a high level of robustness, while ZN PRC and Cohen-Coon give a $\boldsymbol{M_s}$ value higher than recommended. The SIMC settings results in a controller which balances all the measures, which explains its popularity.

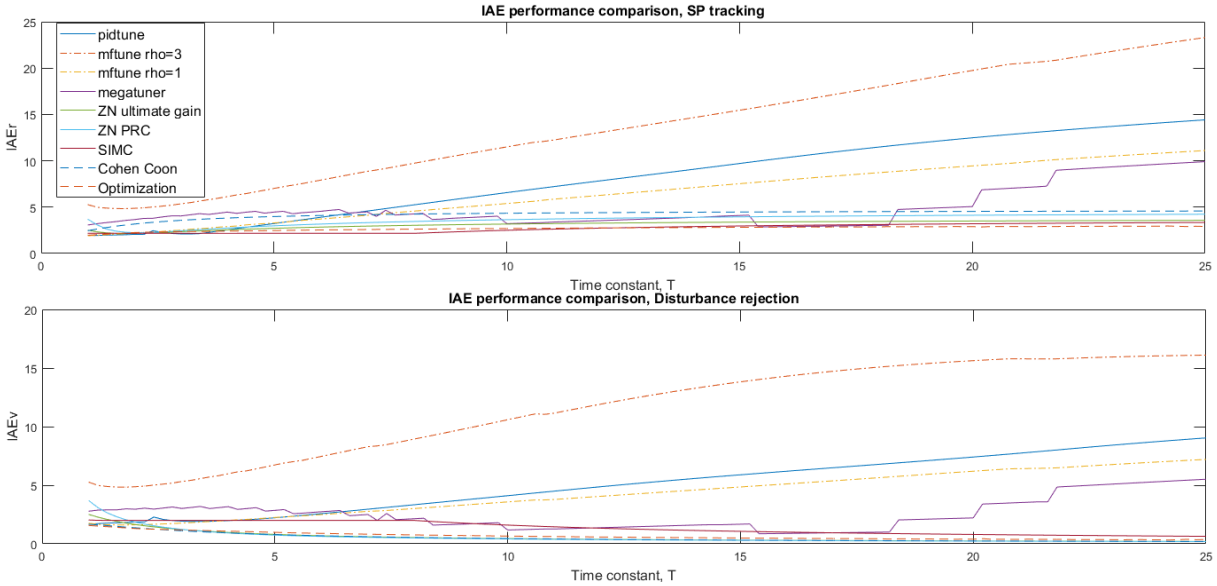


Figure 5.4: Comparison of IEA for SP tracking and disturbance rejection using different tuning methods for FOPTD model, with an increasing time constant, T

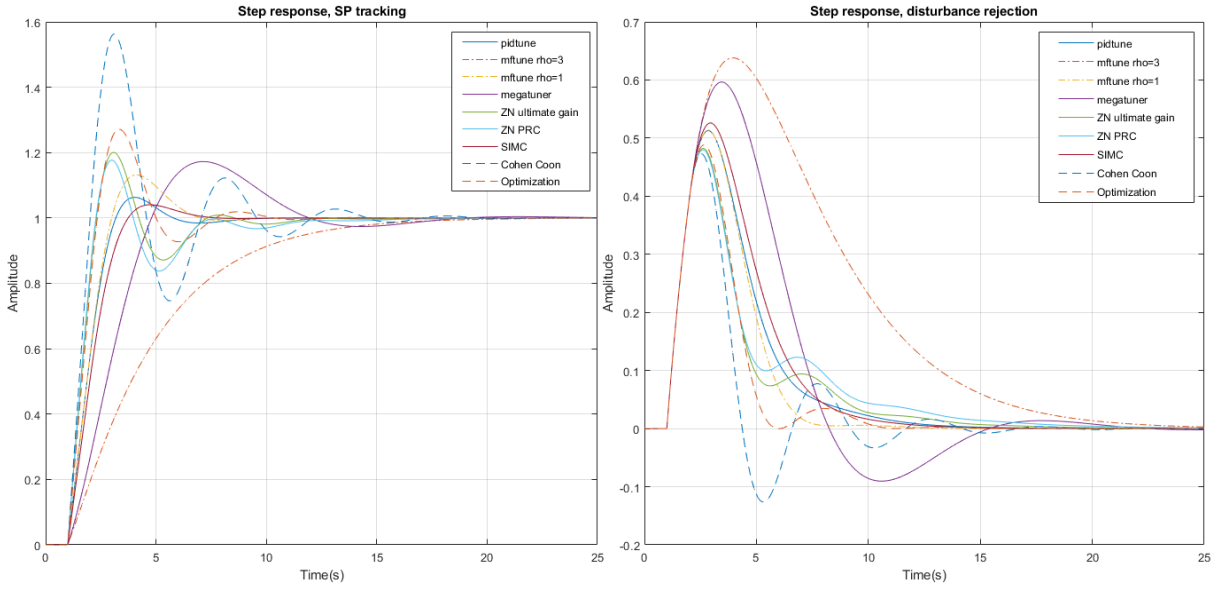


Figure 5.5: Comparison of step response for SP tracking and disturbance rejection using different tuning methods for a FOPTD model with T=2

Table 5.1: Mean measurements of performance and robustness, FOPTD

	GM	PM	M_s	IAEr	IAEv	TV
pidtune	19.4	60	1.25	9.45	5.91	1.23
mftune $\rho = 3$	39.1	72	1.09	15.73	12.42	0.56
mftune $\rho = 1$	13.0	66	1.21	7.50	4.90	1.30
megatuner	8.8	59	1.33	5.85	3.11	3.57
SIMC	3.1	56	1.63	2.85	1.21	9.59
ZN UG	1.9	31	2.62	3.24	0.48	30.93
ZN PRC	1.6	26	3.45	3.75	0.48	54.56
Cohen-Coon	1.5	21	3.93	4.32	0.45	62.72
Optimization	2.0	39	2.32	2.75	0.60	24.53

5.2 Comparison of tuning methods based on integrating plus time delay model

In this section, tuning rules for an IPTD model are compared. The Cohen-Coon tuning rules cannot be applied to this type of system and mftun is equal to the δ-PI rules, and therefore not included. In this section, θ is increased from 1 to 5, while K is kept constant at 1.

Figure 5.6 shows how controller parameters K_p and T_i change when θ is increasing. The K_p values are small and close together, suggesting that there is little room for adjustment before the system goes unstable. Larger values for θ gives smaller gains, as this makes the system more difficult to control. ZN and optimization results in the most aggressive controllers. The ZN UG method fails when $\theta > 3.5$.

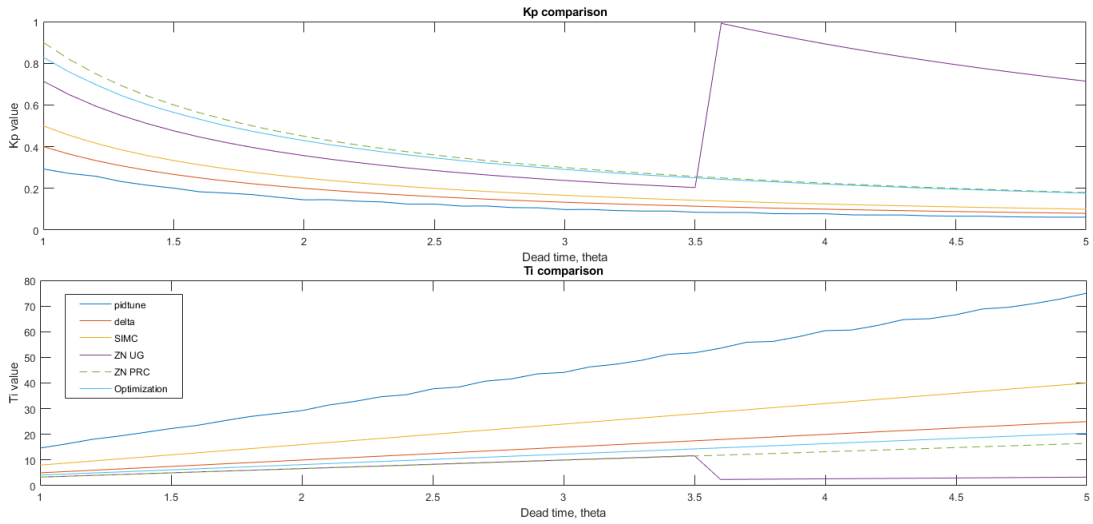


Figure 5.6: Evolution of K_p and T_i using different tuning methods for a IPTD model with increasing θ

The same correlation between TV and M_s as for the FOPTD test are seen in figure 5.7. In this case, the robustness of the different methods is more steady than for the FOPTD example. Pidtune gives the most robust controller with the lowest input usage, while SIMC and delta provide similar controllers. The trade-off plot in 5.8 also shows that delta and SIMC are similar but that the default choice of delta favors robustness over performance compared to SIMC. The δ-tuning curve is obtained by adjusting δ between 1.1 and 2.1. The other methods give good performance but poor robustness.

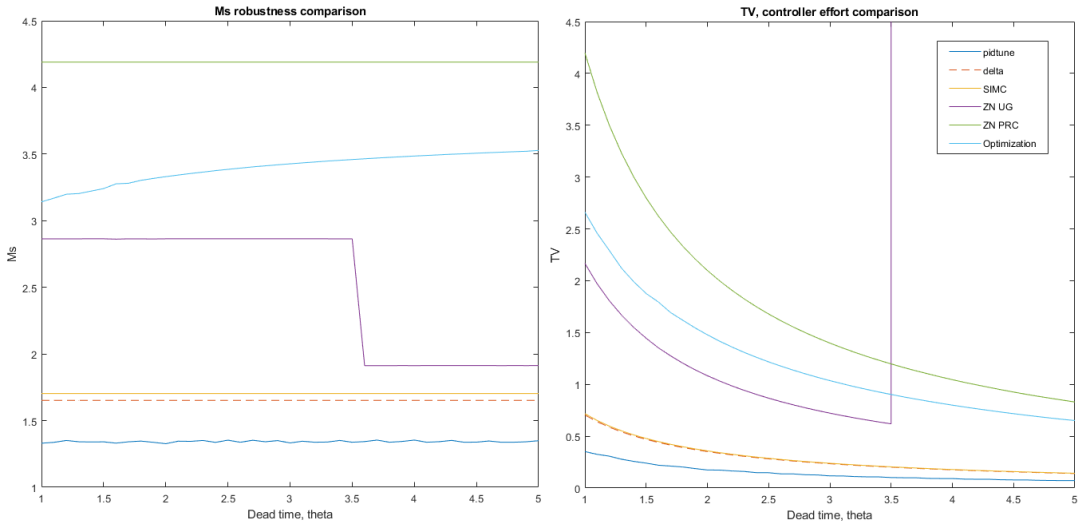


Figure 5.7: Comparison of M_s and TV for different tuning methods based on a IPTD model with increasing θ

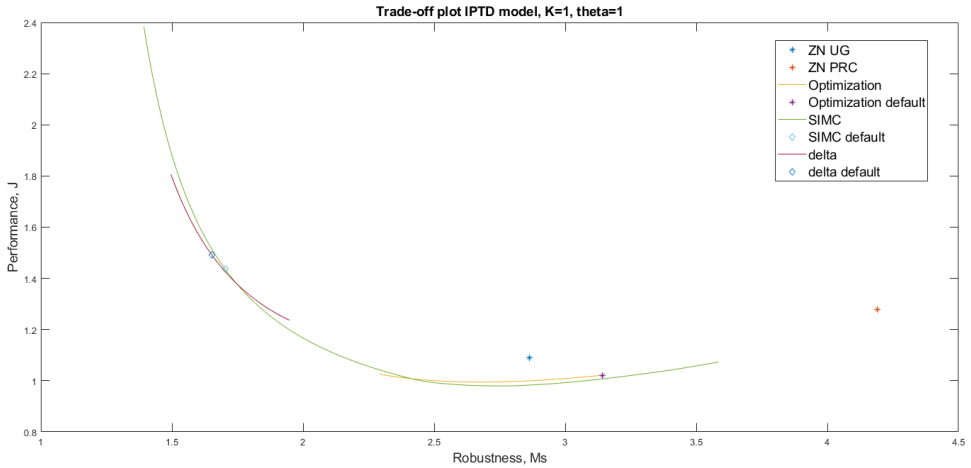


Figure 5.8: Trade-off plot between robustness and performance, comparing different methods based on a IPTD model with θ=1 and K=1

The IAE values for SP tracking show that the different methods give similar performance, with SIMC and optimization at the lower end and pidtune at the higher end. The differences are more significant regarding disturbance rejection, where pidtune performs considerably worse than the other methods. ZN and optimization produce the lowest IAE values.

Figure 5.10 shows the step responses for SP tracking and disturbance rejection for the different methods. This example shows the difference between the more aggressive methods being ZN and optimization, with M_s larger than 2, and the slower methods like pidtune. The aggressive tuning methods give much faster rise time but also shows oscillating

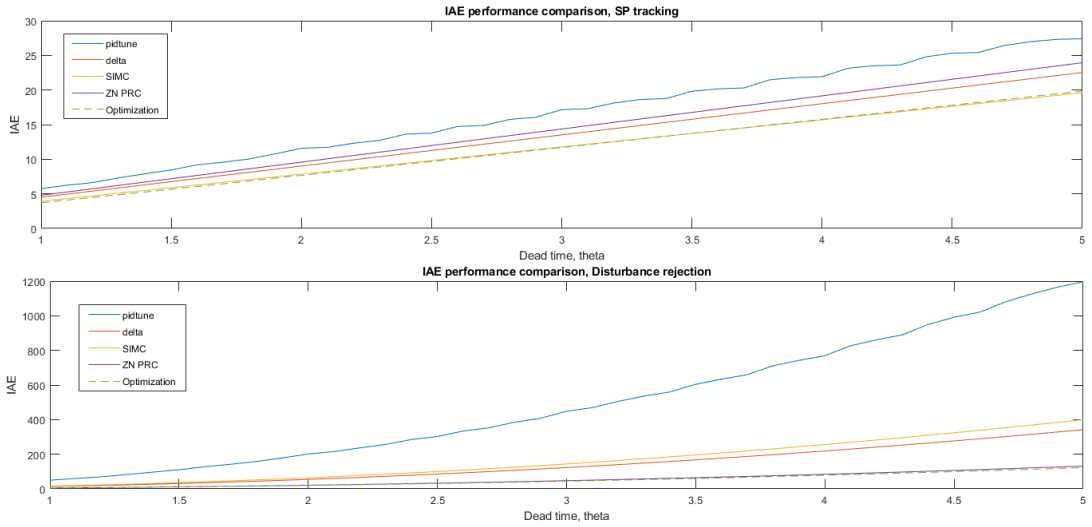


Figure 5.9: Comparison of IEA for SP tracking and disturbance rejection using different tuning methods for a IPTD model with increasing θ

behavior.

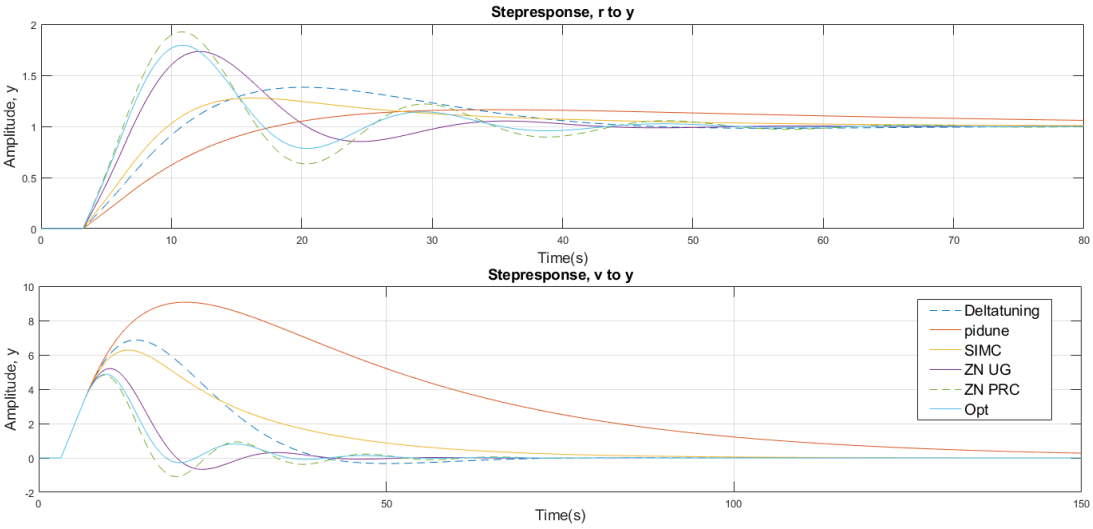


Figure 5.10: Comparison of step response for SP tracking and disturbance rejection using different tuning methods for a IPTD model with K=1 and $\theta = 3$

Table 5.2 summarizes the results from the tests in this section. Note that the values for the ZN UG method is only the mean values for θ between 1 and 3.5, as the method fails with larger values for θ.

Table 5.2: Mean measurements of performance and robustness, IPTD

	GM	PM	M_s	IAEr	IAEv	TV
pidtune	5.0	60	1.34	15.4	438.8	0.1465
delta	3.5	40	1.65	13.5	141.9	0.2867
SIMC	2.96	47	1.70	11.7	163.2	0.2921
ZN UG*	1.8	25	2.86	8.8	313	1.0986
ZN PRC	1.5	18	4.19	14.3	54.5	1.7078
Optimization	1.6	23	3.40	11.7	51.1	1.2121

5.3 Comparison of tuning methods based on double integrating plus time delay model

The DIPTD model requires a PID controller to be stabilized and for that reason mftun are not included in this test. ZN and CC tuning rules are also unable to stabilize a DIPTD process. Like in section 5.2, K is kept constant, and θ is increased from 1 to 5. Figure 5.11 shows how the controller parameters are chosen based on different methods. The differences in K_p are small, with the optimization method standing out when θ is small. For higher readability, the pidtune parameters are left out of the plots displaying T_i and T_d, as the values are very high.

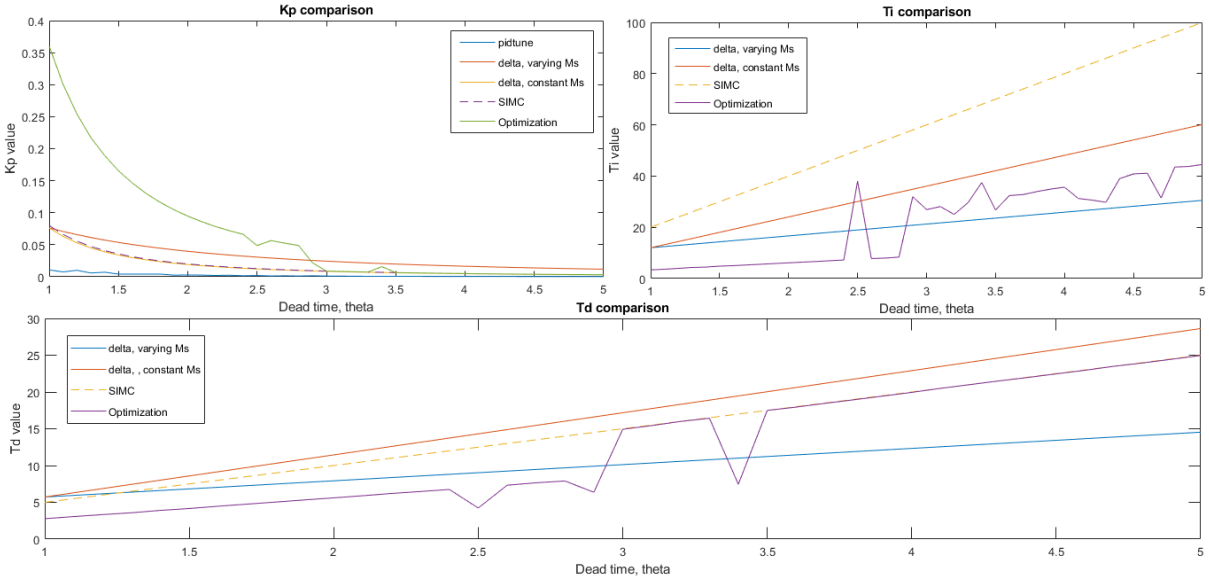


Figure 5.11: Evolution of controller parameters using different tuning methods and a DIPTD process with increasing θ

Figure 5.12 presents the M_s values for the different methods. Both pidtune, SIMC, and δ-tuning gives good robustness based on this measure. The optimization method gives

poor robustness for small θ, but acceptable values when dead time reaches 3 seconds. The reason for this might be that when θ is small, there is more room to increase gains to obtain low IAE values, while for high θ, this introduces instability. The TV plot in figure 5.12 shows that the performance of the optimization controller comes at the cost of very high input usage and that when θ is larger than 3, the difference between the controllers is small. Looking at figure 5.13, which compares IEA shows that the optimization method achieves the lowest values for both SP tracking and disturbance rejection. In this case, pidtune stands out as considerably worse than the other methods and is left out of the disturbance rejection plot. When using the setting δ-tuning with varying M_s, superior performance for models with high dead time is achieved.

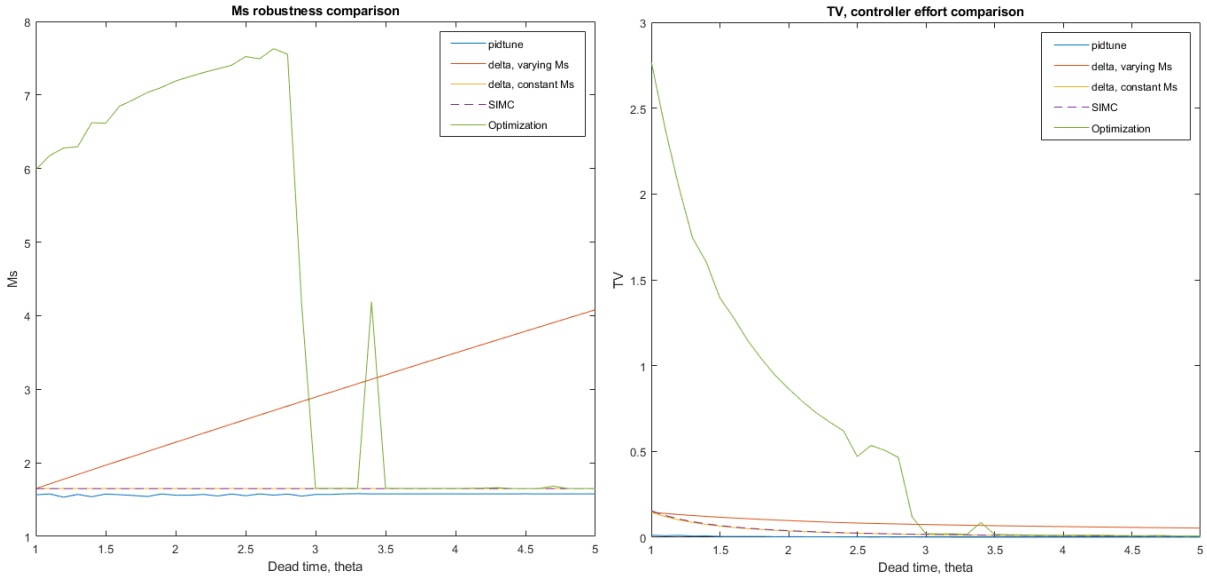


Figure 5.12: Comparison of M_s and TV for different tuning methods based on a DIPTD model with increasing θ

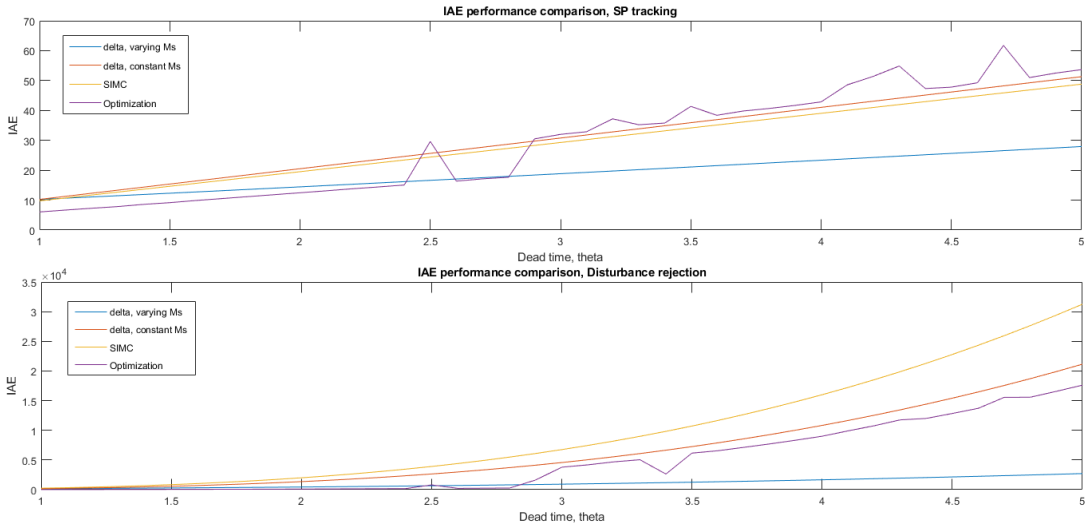


Figure 5.13: Comparison of IEA for SP tracking and disturbance rejection using different tuning methods for DIPTD model with increasing θ

Figure 5.14 shows an example of step responses for a DIPTD plant and the different tuning methods. These plots show that pidtune gives slow SP tracking and bad disturbance rejection compared to the other methods. The responses from SIMC and delta are similar, with SIMC favoring SP tracking and delta favoring disturbance rejection. The optimization algorithm sacrifices SP tracking for disturbance rejection and increasing W_1 should be considered.

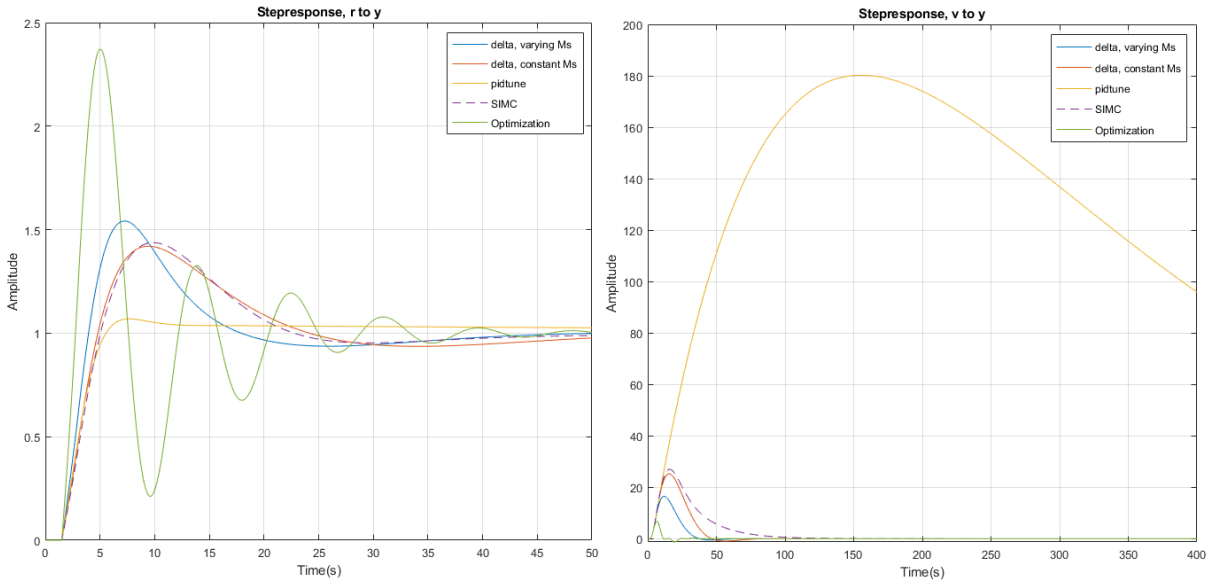


Figure 5.14: Comparison of step response for SP tracking and disturbance rejection using different tuning methods for a DIPTD model with k=1 and $\theta = 1.5$

Figure 5.3 summarizes the results of the current section.

Table 5.3: Mean measurements of performance and robustness, DIPTD

	GM	PM	M_s	IAEr	IAEv	TV
pidtune	3.3	60	1.57	240.3	1 068 500	0.0026
delta, varying M_s	2.0	51.8	1.52	52.9	15 152	0.0168
delta, constant M_s	3.31	41.5627	1.65	30.8	6 699	0.0303
SIMC	3.6	39.2	1.65	29.3	9 900	0.0320
Optimization	0.8	23.3	4.25	29.4	5 205	0.5480

5.4 Comparison between pidtune, mftune, and megatuner

To investigate the robustness of pidtune and mftun, they were used to tune 1000 controllers of order 1 until 70, a total of 70 000, to see how many that could be stabilized. Tuning PID controllers for 70th order models are not common, and the reason for this high number is to illustrate the trend in the data. A time delay of 1 second is added to each model. A modified version of mftun was also included. This version uses the δ-optimization to approximate a DIPTD model and tune a PID controller when the model has poles at the origin. The results of this experiment are shown in figure 5.15, and the dotted lines are the straight lines which fit the data best. Pidtune can stabilize most systems of order 1 until 10 before the success rate starts dropping, ending up at 90% stable systems at order 70. The success rate of mftun starts dropping from the beginning, ending up at about 60% at order 70, with the modified version performing worse. A comparison of speed was also made between the two methods, using 100 6th order SSM. Pidtune uses 3.1 seconds and mftun 1.5, which are impressive as pidtune uses iterations to find a controller that suits a set of constraints.

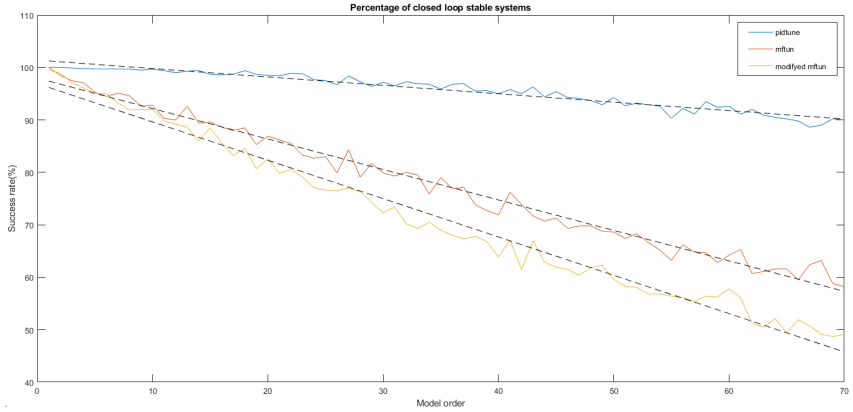


Figure 5.15: Comparison of mftun and pidtune, percentage of stabilized closed loops versus model order

A test like the one performed on mftun and pidtune was conducted to test megatuner. In this test, only models without integrators were used. The result is displayed in 5.16, and as seen, megatuner does not fail a single time, while pidtune fails two times. By comparing the results in figure 5.16 and 5.15, it becomes clear that the models with integrators are the most challenging systems to tune controllers for. Taking this into account, mftun method stabilizes slightly fewer processes than megatuner but uses a fraction of the time.

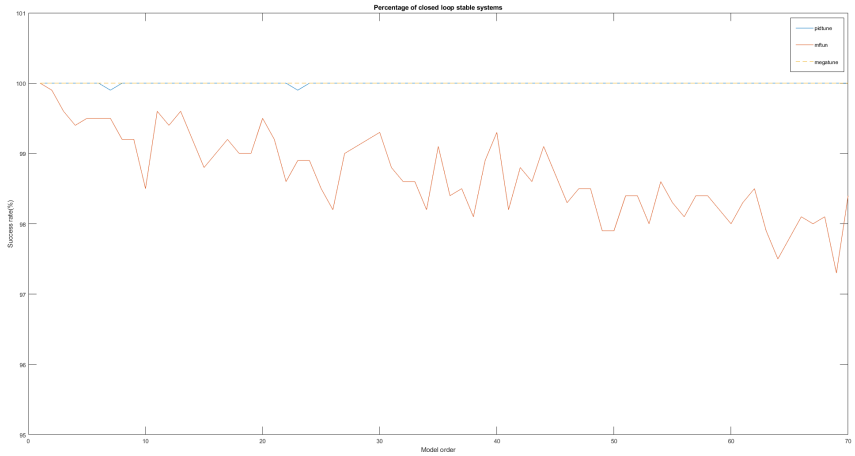


Figure 5.16: Success rate plotted against model order, comparison between mftun, megatuner and pidtune (no integrating models)

Figure 5.17 shows the trade-off between the methods compared in this section. The PO-optimal controller described in section 4.6.3 is used for reference. Mftun is close to optimal in all cases, while megatuner gives less performance than the other methods. To make the pidtune curves, ω_c is used as the tuning parameter, and the graph shows that it is possible to achieve close to optimal performance when M_s is between 1.7 and 2.2 for the 2 first systems. The graph also shows that the default pidtune controller is not the best choice in these cases. For the IPTD model mftune ($\delta - PI$) is best, while for the DIPTD system they are similar in the region with M_s between 1.5 and 2. For the FOIPTD system in equation 5.1, pidtune is slightly better, but both are close to optimal. However, the default choice of pidtune gives high robustness at the cost of performance.

$$hp = K\frac{1}{s(Ts+1)}e^{-\theta s} \tag{5.1}$$

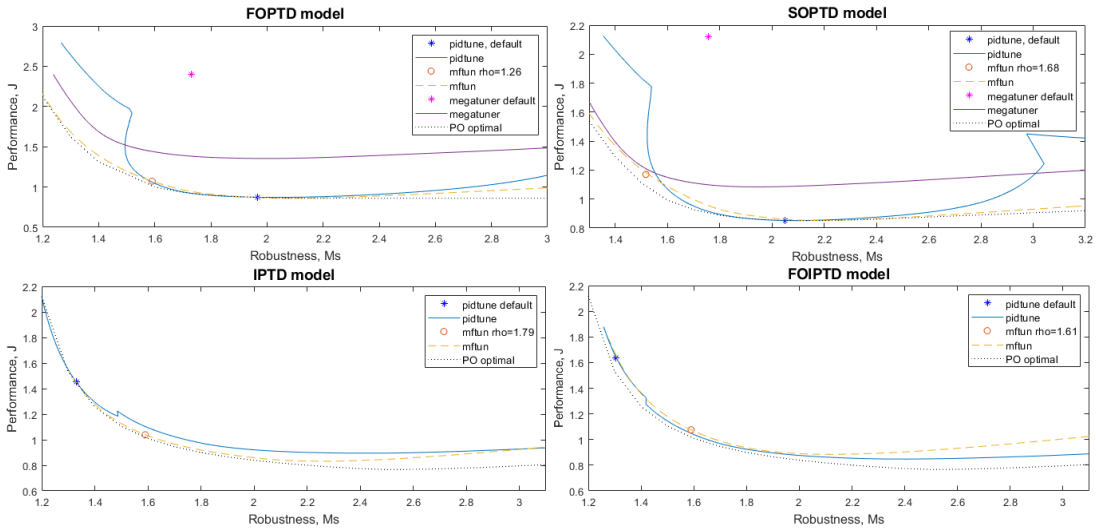


Figure 5.17: Performance vs robustness trade-off curves, comparison between pidtune, mftun, and megatuner

5.5 Comparison of PI controller tuning methods based on randomly generated SSM

This section presents the results from tuning PI controllers based on randomly generated SSM, using the Matlab function rss. The controllers are tuned using the different methods explained in chapter 4, and the Matlab code in appendix D. 2 different aspects are compared; the success rate, performance, and robustness of the methods; and the performance of the different methods for obtaining process parameters. The 3 different methods for calculating model parameters from SSM are explained in section 2.4. This experiment generated a large amount of data, which are provided in appendix B. This section gives a discussion of the results, some examples, and a summary.

5.5.1 Description of the experiment used for method comparison

To compare the different tuning and estimation methods to each other, 100 random SSM of order 3, 6, 9, 12, and 15 were generated, a total of 500 models. All the models have a dead time of 1 second. It is the same 500 models that are used for all testing in section 5.5, and 5.6. All methods were used to tune PI controllers based on these models, using the 3 methods for parameter estimation. For example, this means that the SIMC method has been tested 3 times for each model, a total of $3 * 100 * 5 = 1500$ times. The methods pidtune, mftun, megatuner, δ-optimization, and optimization tuning does not rely on the model estimation methods, as they are used directly on the SSM. This means that they have been used once for each of the models. The performance and robustness measures

presented are the median values based on the tuning attempts that gave a stable closed loop system, i.e. the successful tuning attempts. The standard deviation of the M_s value is included in all cases to illustrate the consistency of the method. The Ziegler Nichols PRC tuning rules which rely on R and L has only been used once for each model.

Abbreviations used in plots:

- ZN UG - Ziegler Nichols Ultimate Gain method used directly on the generated SSM
- ZN UG est - Ziegler Nichols Ultimate Gain method used on the estimated model
- ZN PRC L and R - Ziegler Nichols PRC method using reaction rate R and Lag, L
- ZN PRC - Ziegler Nichols PRC method using time constant, gain and time delay
- Opt 1 - Optimization tuning which minimizes IAEr, IAEv and M_s, i.e. it uses transfer functions see section 4.6.1
- SSM Opt - Optimization tuning using SSM exclusively, minimizes IAEr, IAEv and TV. See section 4.6.2

About the models:

- 3rd Order, 12 has a pole at the origin
- 6th Order, 15 has a pole at the origin
- 9th Order, 15 has a pole at the origin
- 12th Order, 29 has a pole at the origin
- 15th Order, 11 has a pole at the origin

5.5.2 Results for tuning methods using estimated model parameters

The δ tuning rules for the IPTD system was included in the test, but only for models with an integrator, which is the reason for the low success rate. δ tuning works best with parameters from the optimization method, which give margins and M_s closest to the ones obtained with a pure IPTD in section 5.2. The exception is for the 3rd order models where the graphical parameter estimation is best. This test also shows that for higher order models with integrators, it might be insufficient to use an IPTD approximation. For the 30, 12th order models with an integrator, half could be stabilized using optimization estimation. The measures for performance and robustness are inconsistent between estimation methods, with M_s above 2 in many cases.

The SIMC and the iSIMC tuning rules produces the highest number of stable closed loops when used together with graphical estimation. In general, the differences between iSIMC and SIMC is small. iSIMC used together with procest stands out as the least effective

combination in terms of success rate, but the best in terms of IAE measures. In general, SIMC results in low values for IAE compared to the other methods in most cases.

The best overall combination in terms of success rate is ZN PRC together with graphical estimation, which is a surprising result given that it is a method which results in small margins. This combination also gives a small standard deviation of M_s compared to the other methods. The ZN PRC method which uses the R and L values, provide similar robustness but succeed far less than the rules based on K, T and, θ, which illustrates the problem with using only 2 variables to describe the process. Using ZN UG based on estimated data produce poor results for both performance and robustness, for all estimation methods, this indicates that ZN UG needs an accurate model. The CC tuning gives controllers with the lowest margin and middle to large values of IAE. Similar to what was seen in section 5.1.

5.5.3 Results for methods using SSM directly

The first optimization tuning approach, Opt 1, works well with lower order models but starts to fail when the model order is larger than 9. For the 15th order models, it only succeeds 25 times. The problem is not that the closed loop becomes unstable, but instead that the method fails. This error is caused because the SSM is converted to a TF, and then a large amount of simulations is carried out to find the best parameters. For the more complex TF, the chance of one of the simulations to fail becomes high. The other optimization based tuning approach does not have this problem. The method gives good, consistent values for performance and robustness.

Matlab pidtune with default settings only fail 4 times and are by that criteria the best of all tuning methods considered. This number can be lowered to 2, by changing the PM setting. For the lower order models, this method is among the best for SP tracking and disturbance rejection. When increasing ω_c for increased performance, the success rate drops and the standard deviation of the results increase.

Megatuner gives stable closed-loop systems in all cases where there are no integrators. The results show that megatuner provides a PM close to 51.6 in all cases, making it likely that it is using a similar approach as pidtune to obtain controller parameters. Megatuner gives average results for SP tracking and disturbance rejection compared to the other methods.

ZN UG used directly on the SSM works well for lower order systems, but fail in the majority of attempts when the model order is 9 or higher. This method results in poor performance measured by IAE.

The δ-optimization based IPTD approximation cannot be said to work well in this test, with the implementation used. The method is successful approximately half of the cases,

Table 5.4: Optimization tuning settings

	W_{sp}	W_{dr}	W_{tv}	Goal
Set 1	1	1	1	Balance
Set 2	1	1	2	Robustness
Set 3	2	2	1	Performance
Set 4	1	2	1	Disturbance rejection

and the results are varying. The method seems to give tuning with high margins, and M_s below 2.

The optimization based tuning method based on SSM was tested using 4 different settings for the weight coefficients, these are listed in table 5.4. Figure 5.18 shows how the different settings affect the performance and robustness, these are the mean values for all 500 models. Judging by this, the settings push the controller towards the intended goal stated in table 5.4. In general, this method works well, but need improvement to compete with pidtune, mftun, and megatuner in terms of success rate.

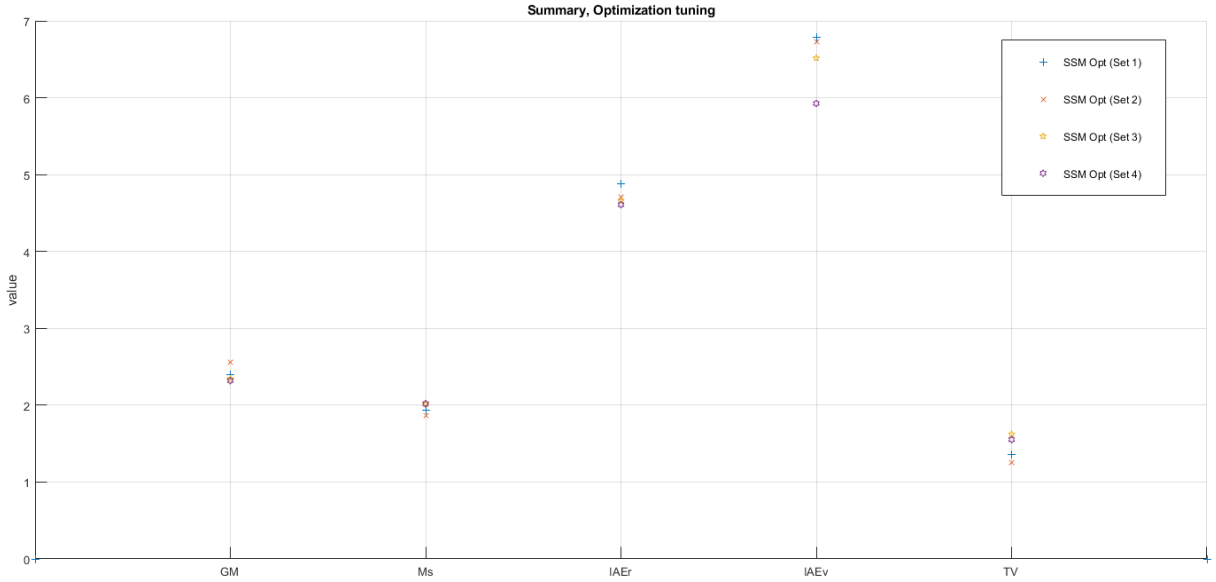


Figure 5.18: Summary of performance and stability measures for optimization-based PI tuning, 4 different settings

Since there are no suggested value for the tuning parameter ρ for mftun, a plurality of values was tested. The total averages for robustness and performance for these values are shown in figure 5.19. The large spread in values indicates that adjusting ρ is an effective way of getting desired closed loop characteristics. For the lower order models, ρ can be selected as low as 1 without affecting the success rate but overall, a value $\rho = 2$ seems like a good starting point.

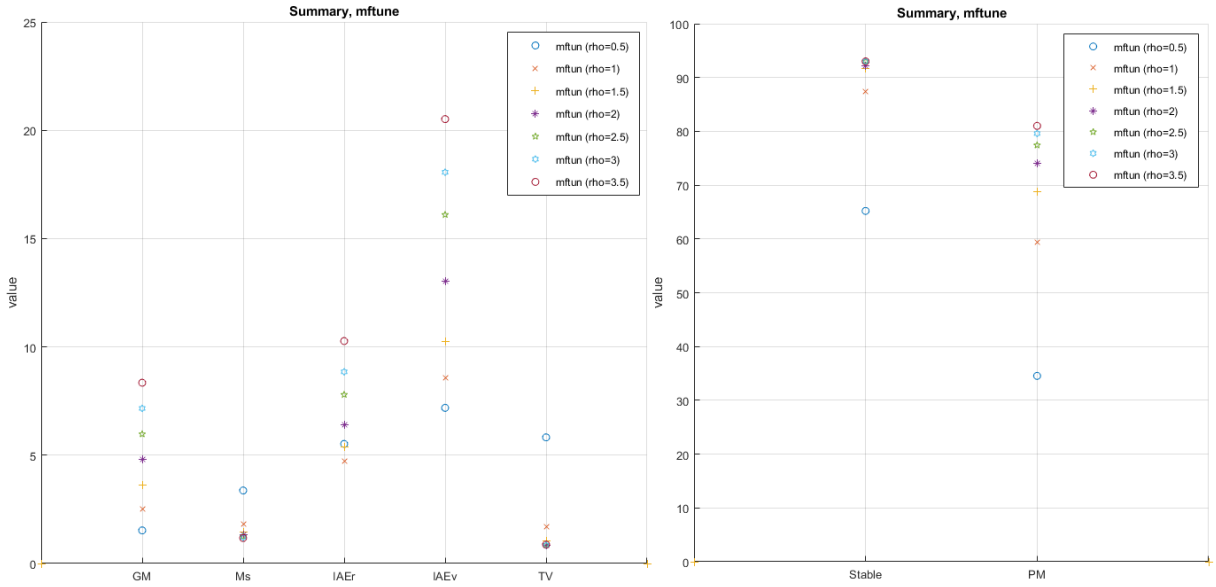


Figure 5.19: Summary of performance and stability measures for mftun, using different values for ρ, based on all models

5.5.4 Summary of PI controller tuning results

This section summarizes the results for the PI controller tuning experiments. Figure 5.20 and table 5.5 contains the mean values for all tuning methods. This mean is calculated using the median values for each tuning experiment and then calculating the average of all these median values. The data basis for the methods using estimation is more extensive, as each experiment with 100 random SSM of a given order is carried out 3 times. This is because there are 3 different estimation methods, as explained in section 2.4.

The summary shows that the methods which present the most stable systems are pidtune, mftun, SSM optimization, and megatuner. Of these methods, mftune gives the highest margins and lowest value for M_s. SSM optimization is best for disturbance rejection and pidtune is best for SP tracking. The SIMC methods and ZN PRC also work with a large number of models but SIMC gives higher GM and lower IAE values than ZN PRC. Pidtune, δ-optimization and ZN PRC RL provide the mean M_s values closest to 1.59.

There are tables with mean values per estimation method in appendix B. The simple graphical method from estimating process parameters gives a higher success rate for all the methods except for delta. The two other methods are similar to each other in terms of success rate and seem to result in better controllers. A likely reason for this is that the graphical method results in reasonable values, also for processes which cannot be approximated well by a FOPTD model. This results in a stable closed loop but might not give the desired performance. The other methods seem to produce ”unusable” numbers when a model can’t be approximated well enough, this results in model parameters which

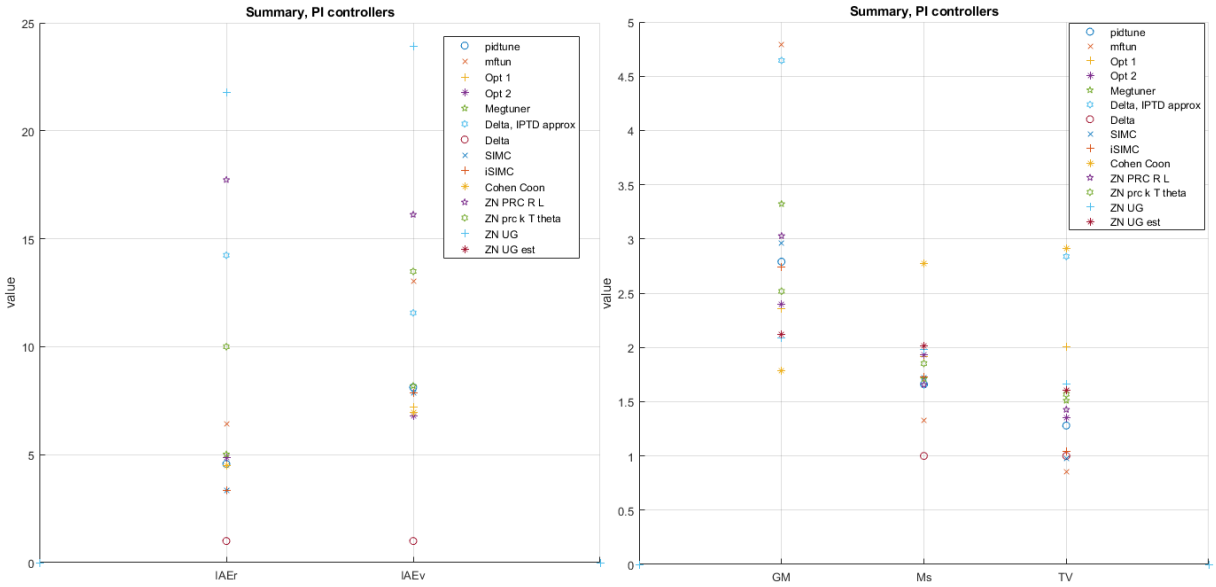


Figure 5.20: Mean performance and stability measures for PI controller tuning methods, tuned using all 500 random SSM

are far from the real process and then instability. However, in the cases where they succeed the model is closer to the original process. It is hard to draw any conclusions based on the standard deviation of M_s, for the different estimation methods and all methods give varying results.

Table 5.5: Average robustness and performance values for all PI tuning methods

	Stable	PM	GM	M_s	IAEr	IAEv	TV
pidtune	99	60	2.79	1.66	4.57	8.12	1.28
mftun(ρ1.5)	91.6	68.72	3.625	1.4621	5.3854	10.2371	1.0577
mftun(ρ2)	92.2	74.0499	4.7920	1.3241	6.4158	13.0460	0.8566
Opt 1 (TF)	72	53.332	2.356	1.917	4.867	7.222	2.009
SSM Opt (Set 1)	88	53.623	2.401	1.934	4.8842	6.789	1.354
Megatuner	83.4	51.587	3.323	1.71	5.017	8.185	1.511
Delta, IPTD approx	47.2	79.607	4.645	1.653	14.232	11.567	2.838
Delta	6.6	34.70	3.078	2.713	13.338	81.701	2.272
SIMC	72.667	59.948	2.96165	1.71885	3.365	7.8799	0.9775
iSIMC	69.533	62.394	2.7377	1.724	3.3501	7.8533	1.038
Cohen-Coon	59.4	38.184	1.785	2.773	4.484	6.9697	2.915
ZN PRC RL	49.4	89.644	3.028	1.656	17.73	16.116	1.427
ZN PRC K T θ	70.53	84.95	2.5178	1.852	9.999	13.486	1.5673
ZN UG	56.6	79.953	2.08579	1.985	21.759	23.8988	1.662
ZN UG est	39.5	74.6502	2.0134	2.2457	58.1964	30.9186	2.1631

5.6 Comparison of PID controller tuning methods based on randomly generated SSM

In this section, PID controller tuning methods have been tested on randomly generated SSM. The setup for the experiment is the same as explained in section 5.5.1. A difference worth mentioning is that the SIMC PID tuning rules use the T_2 time constant, so a 2nd order model must be estimated. The graphical method for the 2nd order parameter estimation is not used, meaning there is no SIMC tuning attempt in those cases. The complete results of the tests are provided in appendix C.

5.6.1 Results for methods using estimated model parameters

For PID controllers the iSIMC method, using graphical estimation presents the highest number of stable closed-loop systems, for all model orders. The difference between SIMC and iSIMC for PID controllers is that iSIMC uses a FOPTD approximation while SIMC uses SOPTD. For the two other estimation methods, optimization and procest, the SIMC method is more successful than iSIMC. With a few exceptions iSIMC give controllers with higher values of M_s and TV, which mean that it is more aggressive than SIMC. The SIMC methods are the only approach which gives M_s below 2 in all cases except for 15th order

models using procest. In that case, ZN PRC results in $M_s = 1.5$, but also very high IAE compared to SIMC.

ZN UG is the least effective method when used on estimated models, also for PID controllers. As in section 5.5, the 2 variable ZN PRC method is less successful than the one using 3 variables. The CC method gives the controllers with the highest input usage and high IAE values, which indicate aggressive responses with significant overshoot and oscillations.

The parameter estimation based methods are in general more effective when using PI controllers than PID controllers. However, the PID controllers give lower IAE values in most cases for both SP tracking and disturbance rejection.

5.6.2 Results for methods using SSM directly

Pidtune fails 4 times in total using default settings, making it the best method in that regard. Regarding disturbance rejection, pidtune produces higher IAE values than most of the other methods. Attempting to specify a lower PM than 60, is not possible in all cases and the median value only drops slightly. This is also the case for PI tuning and is likely due to the parameter selection criteria of the algorithm. MathWorks pidtuning algorithm uses a gridding technique and the system must satisfy a Nyquist stability criterion. If this is not possible with the desired PM, the settings are changed.

Both the δ PRC and the δ-optimization method were tested. The PRC method does not score well in these tests. The success rate is low, and the results have a high variance. The optimization approach to the delta PID controller is better and it succeeds fare more often than in the case with a PI controller based on this approach. It works especially well with the lower order models, up to 6. On the downside, it provides a conservative controller with too high margins and low M_s. This can likely be improved using the tuning parameters available in the δ PID rules.

ZN UG fails often and shows high TV and IAE values. For models of order 9 and lower it gives consistent margins.

The TF optimization method is successful for many models but gives high M_s and TV values. The balance between SP tracking and disturbance rejection is good. For PID controllers, this method does not produce the same large amount of errors as it does for PI controllers.

For the SSM optimization method, the same 4 settings as used for PI controllers and are listed in table 5.4 was used. The overall mean results are given in figure 5.21. For the PID controller setting 1 is the best for almost all measures. The exception is that the robustness setting, provides higher GM and lower M_s, but both measures are acceptable for setting 1. The method also gives a reasonable success rate for all model orders.

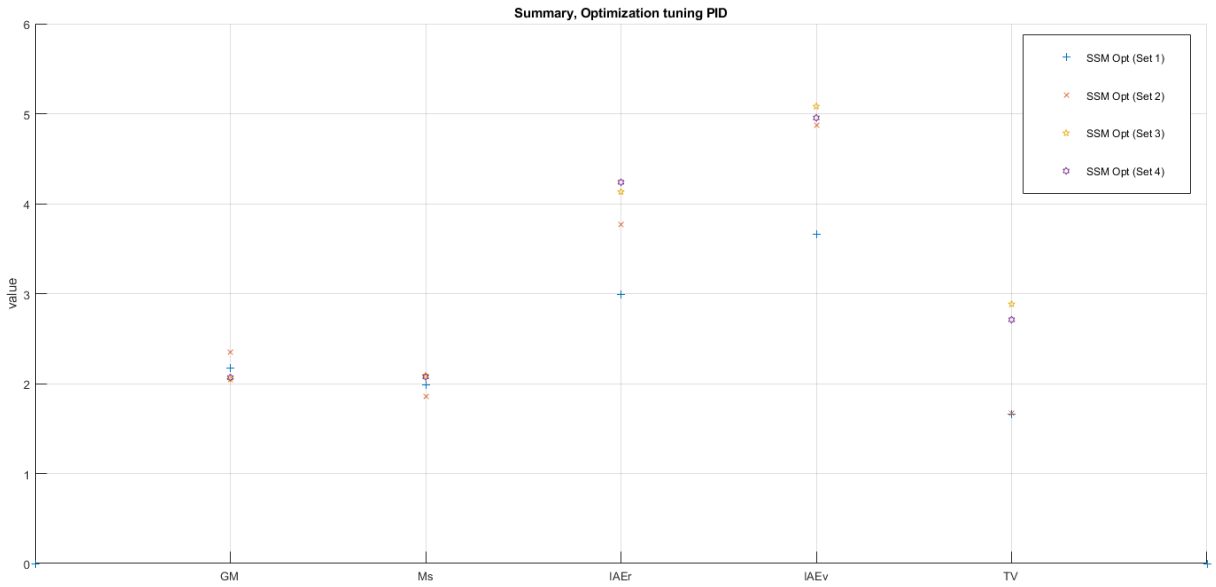


Figure 5.21: Summary of performance and stability measures for optimization based PID tuning, 4 different settings

5.6.3 Summary of PID controller tuning results

This section summarizes the results for the PID controller tuning experiments. The results in figure 5.22 and table 5.6 are calculated as in section 5.5.4. In terms of the number of stable closed-loop systems, pidtune is best, followed by the optimization methods. The delta DIPTD optimization approximation also gives decent success. The plot shows that ZN UG provides the best SP tracking and disturbance rejection, but also small margins and high TV. The ZN PRC methods give the highest IAE values together with Cohen-Coon.

For the methods using parameter estimation, the success rate is lower for PID controllers than for PI controllers.

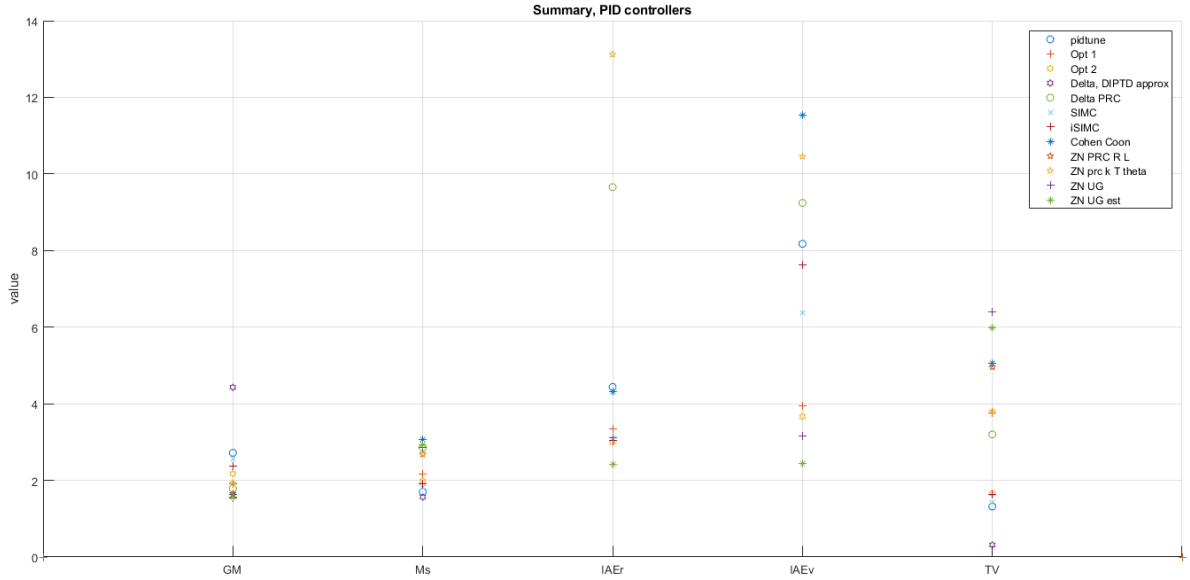


Figure 5.22: MeanSummary of performance and stability measures for PID controllers tuned using different methods based on random SSM

Table 5.6: Average robustness and performance values for all PID tuning methods

	Stable	PM	GM	M_s	IAEr	IAEv	TV
pidtune	99.2	60	2.7198	1.7026	4.4371	8.1752	1.3179
opt 1 (TF)	90.4	52.185	1.9157	2.1607	3.3499	3.9486	3.7509
SSM Opt (set 1)	88.4	54.8963	2.1726	1.9838	2.9945	3.6648	1.6622
DeltaOpt	64.4	90.8102	4.4273	1.5636	76.043	105.187	0.3133
DeltaPRC	23	64.6578	1.7877	2.8503	9.6553	9.2436	3.1964
ZN R L	40	78	1.65	2.68	43.5	200	4.96
SIMC	60.3	60.46959	2.55117	1.80733	3.09	6.37241	1.41635
iSIMC	60.3	62.349	2.377	1.904	3.0473	7.631	1.624
CC	44.93	53.333	1.640	3.063	4.3197	11.539	5.0658
ZN prc	49.27	79.407	1.933	2.699	13.121	10.453	3.805
ZN UG	25.2	54.31092	1.550	2.847	3.115	3.169	6.3987
ZN UG est	15.8	52.981	1.554	2.924	2.415	2.438	5.985

6 Discussion and further work

The possible weaknesses that have been discovered in this thesis are explained in this paragraph. Input usage, TV, have only been calculated for SP tracking and it could have been beneficial also to calculate TV for disturbance rejection. δ-PID tuning for DIPTD systems were not tested for PID controllers based on SSM. When testing with random SSM, the performance and robustness measures are based on stable results only. This means that the methods with a high success rate, have a better data basis. It is possible that this makes the performance of the less successful methods seem better or worse than in reality. In the total average results presented in chapter 5, using values for just one of the estimation methods might have been better. Using average instead of the median for PM would have been a better choice. Another approach to solving the project goals might have been to test fewer tuning methods more thoroughly, i.e. finding the best tuning parameters for the methods. In general, it might have been better to produce less data, in order to be able to examine the results more closely. In many cases, it would have been better to use the PID controller on parallel form, especially when comparing parameters between controllers. There is also some decimal point round off inconsistencies in the report.

Most of the testing in this report is done using the recommended settings for method tuning variables. In some cases, the stated performance might be improved by adjusting the tuning variables. This has not been extensively examined, as it was prioritized to cover a broader scope of methods, as well as the fact that simplicity is vital. By this, it is meant that for a method to be widely implemented in industry, it should be easy to use.

Since most of the results in the report are based on a large amount of randomly generated SSM, the focus of the work has been to use tuning and estimation methods which are possible to automate. This means that some analytic methods for tuning, such as IMC, has been left out. Another strategy that falls under this category is to covert the SSM to TF and use model reduction to obtain model parameters for controller tuning.

When generating random SSM in Matlab, the resulting models might may have a non zero D matrix. This is problematic for control, and therefore all D matrices are set to zero. The time delay was also added after the generation of the SSM. This is to specify that the models used are slightly modified from the generated model.

The optimization based controller tuning method which uses TF and M_s in the objective function often fails for higher order models. It is possible that this problem can be solved, but due to time limitations, this was not investigated thoroughly. The problem might lie in the step size of the simulations of in the constraints. The $\boldsymbol{\delta}$-optimization approach to estimating a DIPTD model works well on the lower order models. Using more aggressive values for $\bar{c}$ and $\boldsymbol{\delta}$ would likely make this an interesting approach to investigate further.

It was presented in this thesis that many of the control loops in the industry are poorly tuned. Many of the control loops are tuned by lower level technicians with insufficient training in the topic. One way of bringing more complex tuning methods to the industry would be an application for phones and tablets. This tool should be based on computer vision techniques to convert a picture of a graph to discrete time data points, which then can utilize the model estimation techniques tested in this thesis. This is possible by marking some points on the graph, and manually assigning x and y values, the data points along the entire graph can them be labeled. This again should be used to provide the user with controller parameters. These parameters could be based on several different methods, and estimated step responses using different methods could be presented to the user, who can decide which to use. This app could also save the plant model together with controller parameters and relevant loop data such as margins and performance in a database, helping the user to keep track and document the control loops. A search on Google play store shows that no are apps incorporating this functionality.

If the goal of mftun and megatuner is to be able to work with all kinds of processes, support for derivative action should be added. The results in this thesis show that the plant which contains integrators is the weakness for both mftune and pidtune, while megatuner does not work with these systems at all. This is where work needs to be done to create a tuning method which can stabilize all LTI stable systems.

7 Conclusions

Controllers can be tuned based on SSM in a variety of different ways, outlined in figure 2.8. The methods tested in this thesis are tuning methods that work directly on SSM and different methods of parameter estimation to obtain a lower order model. The parameter estimation methods utilize that an open loop simulation of the SSM always can be performed.

- Conclusion about tuning methods based directly on state space models
  - Pidtune, mftun, megatuner and optimization tuning can be used directly with SSM and give good results, also for higher order models
  - TV can be used in the objective function of optimization methods to increase robustness, however, it does not give the same flexibility as M_s
  - ZN UG can also be used directly on SSM by finding GM and ω_{180}, this method works well for lower order models
  - The tuning methods that work directly on SSM gives better results than the methods that use approximations
  - The standard deviation of the results in terms of robustness and performance are low for these methods
- Conclusion about model parameter estimation methods
  - Process describing variables K, θ, T, R, L can be found from SSM using graphical estimation or optimization
  - The Graphical method gives the highest number of successful tuning attempts for all tested PI and PID tuning methods
  - The Optimization method gives the best results in terms of performance for the resulting closed loop systems
- Conclusion about tuning methods based on estimated model parameters
  - In many cases, the higher order SSM can be approximated well with a lower order model, making it possible to use tuning methods based on these

  - SIMC stands out as the method which gives the best trade-off between robustness and performance
  - SIMC and ZN PRC gives the highest amount of stable systems
  - The results from controllers tuned based on model estimation generally have a high standard deviation

- Delta tuning rules
  - δ-PI/PD/PID can be used with SSM by estimating an IPTD or DIPTD model from a step response, as indicated in section 2.4. A zero eigenvalue in system matrix A indicates an integrator in the system, and that such models can give a good fit. These model types can also be approximated by using δ-optimization described in 4.2.4. δ-optimization work with non integrating systems as well, and the DIPTD approximation is best. δ-PRC can also be used directly with a SSM since it only requires open loop step response data.
  - Mftun is the second best method in terms of success rate, behind pidtune. Mftune generally gives more robustness than pidtune. 2 is a good initial choice for tuning parameter ρ. values for ρ that give $M_s = 1.59$ for some example systems are given in figure 5.16
  - Megatuner does not seem to fail at all with stable systems without poles at the origin. This method gives less performance than mftune and pidtune, for the same value of M_s.

- Matlab pidtune
  - Is easy to use and rarely fails
  - ω_c is the best tuning parameter for adjusting the trade-off between performance and robustness
  - When tuning for systems with integrators, some manual adjustment might be necessary to get tight control
  - The default settings do not always give the best trade-off between robustness and performance

Bibliography

[1] T. Katayama, *Subspace methods for system identification*. Springer Science & Business Media, 2006, pp. 7–9.

[2] L. Ljung, *System Identification: Theory for the User*, ser. Prentice Hall information and system sciences series. Prentice Hall PTR, 1999, ISBN: 9780136566953. [Online]. Available: https://books.google.no/books?id=nHFoQgAACAAJ.

[3] B. Friedland, *Control system design: an introduction to state-space methods*, ser. McGraw-Hill series in electrical engineering: Control theory. McGraw-Hill, 1986, ISBN: 9780070224414. [Online]. Available: https://books.google.no/books?id=2M1SAAAAMAAJ.

[4] J. R. Carstens, *Automatic Control Systems and Components*. Prentice-Hall inc., 1990, pp. 205–227, ISBN: 0130542970.

[5] D. D. Ruscio, 'Subspace system identification theory and applications', *Telemark Institute of Technology*, pp. 13–23, Jan. 2014.

[6] MathWorks, *Estimate state-space model using subspace method*, Webpage. [Online]. Available: https://se.mathworks.com/help/ident/ref/n4sid.html.

[7] ——, *Estimate state-space model using time or frequency domain data*, Webpage. [Online]. Available: https://se.mathworks.com/help/ident/ref/ssest.html.

[8] D. D. Ruscio, *D-sr toolbox for matlab*, Webpage. [Online]. Available: http://davidr.no/d-sr/d-sr_e.html (visited on 10/04/2019).

[9] F. Haugen, *PID control*. Tapir academic press Trondheim, 2004, vol. 238.

[10] R. Vilanova and A. Visioli, *PID Control in the Third Millennium: Lessons Learned and New Approaches*, ser. Advances in Industrial Control. Springer London, 2012, ISBN: 9781447124252. [Online]. Available: https://books.google.no/books?id=1uB73y89NagC.

[11] N Minorsky, 'Directional stability of automatic steered bodies', *Journal of the American Society for Naval Engineers*, vol. 34, May 1922. DOI: 10.1111/j.1559-3584.1922.tb04958.x.

[12] A. O'Dwyer, *Handbook of PI and PID Controller Tuning Rules*. Imperial College Press, 2009, ISBN: 9781848162426. [Online]. Available: https://books.google.no/books?id=zivrLUbIMgUC.

[13] D. B. Ender, 'Process control performance: Not as good as you think', *Control Engineering*, pp. 180–190, 1993.

[14] S. Skogestad, 'Feedback: Still the Simplest and Best Solution', *Modeling, Identification and Control*, vol. 30, no. 3, pp. 149–155, 2009. DOI: 10.4173/mic.2009.3.5.

[15] C. Grimholt and S. Skogestad, 'Optimal pi and pid control of first-order plus delay processes and evaluation of the original and improved simc rules', *Journal of Process Control*, vol. 70, pp. 36–46, Oct. 2018. DOI: 10.1016/j.jprocont.2018.06.011.

[16] V. Alfaro and R. Vilanova, *Model-Reference Robust Tuning of PID Controllers*. Apr. 2016, pp. 7–19, ISBN: 978-3-319-28211-4. DOI: 10.1007/978-3-319-28213-8.

[17] MathWorks, *Analyze design in pid tuner*, Webpage. [Online]. Available: https://se.mathworks.com/help/slcontrol/ug/analyze-the-design-in-the-pid-tuner.html.

[18] K. Åström and T. Hägglund, 'Pid controllers : Theory, design, and tuning / karl j. astrom and tore hagglund', *SERBIULA (sistema Librum 2.0)*, Mar. 2019.

[19] S. Skogestad and I Postlethwaite, 'Multivariable feedback control: Analysis and design', in. Jan. 2005, vol. 2.

[20] S. Skogestad and C. Grimholt, 'The simc method for smooth pid controller tuning', in *PID Control in the Third Millennium*, Springer, 2012, pp. 147–175.

[21] G. C. Goodwin, S. F. Graebe, M. E. Salgado *et al.*, *Control system design*. Prentice Hall New Jersey, 2001, vol. 240.

[22] C. Chin, *Computer-Aided Control Systems Design: Practical Applications Using MATLAB® and Simulink®*. Taylor & Francis, 2012, ISBN: 9781466568518. [Online]. Available: https://books.google.no/books?id=dL3ITAO6V0sC.

[23] W. Pedrycz, *Fuzzy control and fuzzy systems*, ser. Electronic & electrical engineering research studies: Control theory and applications studies series. Research Studies Press, 1993, ISBN: 9780863801310. [Online]. Available: https://books.google.no/books?id=uupSAAAAMAAJ.

[24] D. Chen and D. E. Seborg, 'Pi/pid controller design based on direct synthesis and disturbance rejection', *Industrial & engineering chemistry research*, vol. 41, no. 19, pp. 4807–4822, 2002.

[25] P. G. C. Eryilmaz, 'Automated pid controller design', pat. US8467888B2, (to appear), 2013. [Online]. Available: https://patents.google.com/patent/US8467888.

[26] *Feedback Control of Dynamic Systems*. Pearson Education, 2008, ISBN: 9788131721421. [Online]. Available: https://books.google.no/books?id=EFgOQABEBjIC.

[27] D. Ruscio, 'On tuning pi controllers for integrating plus time delay systems', *Modeling, Identification and Control*, vol. 31, Oct. 2010. DOI: 10.4173/mic.2010.4.3.

[28] D. Di Ruscio and C. Dalen, 'Tuning PD and PID Controllers for Double Integrating Plus Time Delay Systems', *Modeling, Identification and Control*, vol. 38, no. 2, pp. 95–110, 2017. DOI: 10.4173/mic.2017.2.4.

[29] C. Dalen and D. Di Ruscio, 'A Semi-Heuristic Process-Reaction Curve PID Controller Tuning Method', *Modeling, Identification and Control*, vol. 39, no. 1, pp. 37–43, 2018. DOI: 10.4173/mic.2018.1.4.

[30] ——, 'A Novel Process-Reaction Curve Method for Tuning PID Controllers', *Modeling, Identification and Control*, vol. 39, no. 4, pp. 273–291, 2018. DOI: 10.4173/mic.2018.4.4.

[31] ——, 'PD/PID controller tuning based on model approximations: Model reduction of some unstable and higher order nonlinear models', *Modeling, Identification and Control*, vol. 38, no. 4, pp. 185–197, 2017. DOI: 10.4173/mic.2017.4.3.

[32] D. Di Ruscio, private communication, 2019.

[33] C. Dalen, *Megatuner*, Webpage. [Online]. Available: http://megatuner.no/ (visited on 16/05/2019).

[34] T. Hägglund and K. Åström, 'Revisiting the ziegler-nichols tuning rules for pi control', *Asian Journal of Control*, vol. 4, pp. 364 –380, Dec. 2002. DOI: 10.1111/j.1934-6093.2002.tb00076.x.

[35] M. L. Luyben and W. L. Luyben, *Essentials of process control*. McGraw-Hill, 1997.

[36] F. Haugen and B. Lie, 'Relaxed ziegler-nichols closed loop tuning of pi controllers', 2013.

[37] G. H. Cohen and G. A. Coon, 'Theoretical consideration of retarded control', *Transactions of the ASME*, vol. 75, pp. 827–834, May 1953.

[38] C. E. Garcia and M. Morari, 'Internal model control. a unifying review and some new results', *Industrial & Engineering Chemistry Process Design and Development*, vol. 21, no. 2, pp. 308–323, 1982.

[39] D. E. Rivera, M. Morari and S. Skogestad, 'Internal model control: Pid controller design', *Industrial & engineering chemistry process design and development*, vol. 25, no. 1, pp. 252–265, 1986.

[40] D. E. Rivera, 'Internal model control: A comprehensive view', *Arizona State University*, 1999.

[41] S. Skogestad, 'Probably the best simple pid tuning rules in the world', in *AIChE Annual Meeting, Reno, Nevada*, vol. 77, 2001.

[42] E. FLADBERG, 'Prof. skogestad er vår internasjonale kybernetiker', *Teknisk ukeblad*, pp. 180–190, 2014. [Online]. Available: https://www.tu.no/artikler/prof-skogestad-er-var-internasjonale-kybernetiker/218878.

[43] C. Grimholt and S. Skogestad, 'Optimal pid control of double integrating processes', *IFAC-PapersOnLine*, vol. 49, pp. 127–132, Dec. 2016. DOI: 10.1016/j.ifacol.2016.07.228.

[44] K. J. Åström and T. Hägglund, 'Automatic tuning of simple regulators with specifications on phase and amplitude margins', *Automatica*, vol. 20, no. 5, pp. 645–651, 1984.

Appendix A

Task description

The original task description upon which the thesis is based.

University of
South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: State Space Model Based PID Controller Tuning

USN supervisor: David Di Ruscio

External partner: None

Task background:

Effective algorithms and software for building State Space Models (SSM) from measured plant input and output observations exists. Such methods is subspace system identification methods as DSR or the Prediction Error Method PEM. The resulting SSM are often low or higher order dynamic models representing the most important dynamic properties of the plant, i.e. the gain, the zeroes and the poles are represented in the model. Using PID controller tuning algorithms as the SIMC method only low order, i.e. first and second order models are used to tune PI or PID controllers and a modelling reduction step is used to reduce possibly higher order models to models suitable for the SIMC tuning method. The model reduction step, i.e. the half rule is not a trivial step in case of higher order oscillating and unstable models. Methods for tuning PID controllers directly from general single-input and single-output linear dynamic state space models would be of interest. One existing algorithm is the MATLAB pidtune.m algorithm.

Task description:

1. Give an overview of methods for tuning PID controllers. Comment upon advantages and shortcomings about the methods, and how these methods may be used based on SSM.
2. Give a more detailed user specified description about the MATLAB pidtune.m algorithm/function and how this method may be used based on SSM.
3. Recently algorithms for tuning PI and PID controllers from an integrator plus time delay model and a double integrating plus time delay model was published. Discuss the possibility to extend these methods to be based on general linear SSM.
4. Compare the different methods by simulation experiments using MATLAB or similar (Octave/Python). Random SSM generated from the MATLAB rss.m or drss.m functions may be used. A laboratory physical process may also be used for the experiments.

Student category: IIA students

Practical arrangements: -

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

Signatures:

Supervisor (date and signature):

Student (write clearly in all capitalized letters):

Student (date and signature):

Appendix B

Tuning PI controllers, results

Results from tuning PI controllers based on SSM. These results are discussed in section 5.5.

Tuning PI controllers using SSM, results

Preben Sandve Solvang

June 12, 2019

This documents was created as an appendix to the master thesis "State Space Model Based PID Controller Tuning", by Preben Sandve Solvang. The document contains measurements of performance and robustness for PI controllers tuned using a selection of different methods. The PI controllers are tuned based on randomly generated state space models of order 3, 6, 9, 12 and 15. 100 models of each order was used, and the values represented here are median values. The median gives the best measure, as some measurements contain outliers which gives inaccurate mean values. Table 1 contains the different values for the tuning weights for the optimization method.

Table 1: Optimization tuning settings

	W_{sp}	W_{dr}	W_{tv}
set 1	1	1	1
Set 2	1	1	2
Set 3	2	2	1
Set 4	1	2	1

1 3rd order SSM, PI controller

This section gives the result from tuning controllers based on random 3rd order SSM.

Table 2: Median robustness and performance values, no estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
pidtune (PM=60)	99.0	60.0	2.678	1.710	2.467	2.063	2.449	0.24
pidtune (PM=50)	99.0	50.0	2.902	1.717	3.050	1.805	2.723	0.18
pidtune (PM=40)	99.0	55.6	3.265	1.675	5.566	4.683	2.689	0.22
pidtune ($\omega_c = 0.5$)	94.0	60.0	2.905	1.631	2.345	2.083	2.422	15.15
pidtune ($\omega_c = 0.8$)	91.0	60.0	1.841	2.305	2.509	2.498	6.507	39.44
pidtune ($\omega_c = 1$)	84.0	60.0	1.548	3.012	3.570	4.386	10.195	3.48
mftun ($\rho = 0.5$)	79.0	31.9	1.459	3.451	4.227	1.196	9.617	6.85
mftun ($\rho = 1$)	97.0	61.3	2.533	1.770	3.053	2.082	3.085	0.94
mftun ($\rho = 1.5$)	98.0	70.7	3.766	1.424	3.836	3.083	1.600	0.74
mftun ($\rho = 2$)	98.0	75.1	5.022	1.294	4.717	4.015	1.347	0.30
mftun ($\rho = 2.5$)	99.0	78.0	6.277	1.226	5.822	5.098	1.419	0.19
mftun ($\rho = 3$)	98.0	80.3	7.532	1.186	6.823	5.977	1.505	0.15
mftun ($\rho = 3.5$)	98.0	81.7	8.788	1.160	7.913	6.924	1.553	0.12
Opt 1	99.0	55.4	2.567	1.783	3.180	1.875	2.867	0.76
SSM Opt (Set 1)	96.0	62.2	2.928	1.722	3.429	1.981	1.874	0.60
SSM Opt (Set 2)	96.0	65.6	3.183	1.620	3.796	2.306	1.663	0.55
SSM Opt (Set 3)	96.0	59.3	2.819	1.750	3.363	1.915	2.510	0.68
SSM Opt (Set 4)	95.0	61.3	2.858	1.729	3.427	1.634	2.133	0.74
Megatuner	88.0	51.6	3.382	1.709	3.506	2.175	2.554	0.20
ZN UG	82.0	98.5	2.149	1.870	13.090	5.567	2.914	0.23
Delta tuning, IPTD approx	65.0	88.0	9.685	1.211	13.965	6.231	3.466	3.18

Table 3: Median robustness and performance values, graphical estimation, 3 order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	8.0	52.1	2.672	1.748	3.739	5.967	1.667	1.01
SIMC	90.0	61.1	3.067	1.600	2.478	2.344	2.175	2.21
iSIMC	90.0	64.3	2.784	1.652	2.345	2.386	1.722	5.39
Cohen Coon	87.0	41.3	1.748	2.572	2.943	1.517	4.879	2.80
ZN prc R L	48.0	87.8	2.270	1.838	5.516	2.291	4.552	1.01
ZN prc k T theta	97.0	87.2	2.284	1.846	4.807	2.835	3.616	0.87
ZN UG est	77.0	100.1	2.200	1.833	14.138	6.829	2.558	1.15

Table 4: Median robustness and performance values, optimization estimation, 3rd order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	6.0	29.5	3.953	2.389	11.732	8.744	2.880	1.05
SIMC	80.0	61.6	2.973	1.618	2.311	1.664	2.221	2.82
iSIMC	81.0	65.1	2.649	1.693	2.324	2.122	2.116	1.88
Cohen Coon	71.0	32.8	1.619	3.150	3.324	1.902	6.649	4.54
ZN prc k T theta	77.0	75.2	1.990	2.192	4.213	2.673	4.116	2.87
ZN UG est	72.0	Inf	2.188	1.853	167.996	11.831	2.907	5.94

Table 5: Median robustness and performance values, procest, 3rd order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	3.0	16.4	1.980	3.663	6.517	2.460	9.134	1.01
SIMC	83.0	60.7	3.042	1.617	2.293	2.008	1.835	10.06
iSIMC	73.0	64.9	2.654	1.701	2.071	1.389	2.270	0.68
Cohen Coon	67.0	34.4	1.641	2.916	2.867	0.974	7.007	4.31
ZN prc k T theta	74.0	86.5	2.228	1.887	4.597	3.095	3.580	3.84
ZN UG est	34.0	71.0	1.925	2.205	3.412	1.207	8.473	2.38

2 6th order SSM, PI controller

This section gives the result from tuning controllers based on random 6th order SSM.

Table 6: Median robustness and performance values, no estimation, 6th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
pidtune (PM=60)	100.0	60.0	2.836	1.618	3.795	3.904	1.277	0.25
pidtune (PM=50)	100.0	50.0	2.855	1.696	3.988	3.727	1.542	0.19
pidtune (PM=40)	100.0	55.7	3.174	1.703	6.804	7.685	1.627	0.25
pidtune ($\omega_c = 0.5$)	86.0	60.0	2.491	1.794	2.718	3.760	1.945	35.19
pidtune ($\omega_c = 0.8$)	77.0	60.0	1.655	2.701	3.178	4.456	3.780	15.77
pidtune ($\omega_c = 1$)	72.0	60.0	1.414	3.667	5.515	9.498	6.833	20.05
mftun ($\rho = 0.5$)	61.0	36.1	1.458	3.479	4.556	5.926	3.956	16.34
mftun ($\rho = 1$)	91.0	60.7	2.191	1.970	3.603	4.257	1.762	1.20
mftun ($\rho = 1.5$)	97.0	70.3	3.138	1.540	4.714	5.749	1.230	3.31
mftun ($\rho = 2$)	97.0	77.2	4.099	1.369	5.788	6.824	1.047	0.48
mftun ($\rho = 2.5$)	97.0	79.7	5.124	1.280	7.198	9.073	1.060	0.27
mftun ($\rho = 3$)	97.0	81.5	6.134	1.225	8.285	11.301	1.032	0.19
mftun ($\rho = 3.5$)	97.0	82.7	7.142	1.189	9.665	13.404	1.064	0.15
Opt 1	97.0	53.3	2.329	1.938	4.222	3.136	2.034	0.76
SSM Opt (Set 1)	90.0	57.5	2.351	1.841	4.189	3.018	1.541	0.77
SSM Opt (Set 2)	91.0	60.6	2.449	1.805	4.231	3.264	1.450	0.81
SSM Opt (Set 3)	91.0	52.4	2.458	1.939	4.047	3.275	1.752	0.83
SSM Opt (Set 4)	91.0	54.2	2.417	1.911	4.222	3.399	1.733	0.87
Megatuner	85.0	51.6	3.072	1.730	4.127	3.638	1.684	0.22
ZN UG	72.0	80.9	2.092	1.939	7.725	6.959	1.325	0.49
Delta tuning, IPTD approx	46.0	87.6	5.681	1.333	15.215	9.738	1.526	1.77

Table 7: Median robustness and performance values, graphical estimation, 6th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	6.0	44.2	3.390	1.601	5.220	17.522	1.526	1.71
SIMC	76.0	61.1	2.859	1.687	3.015	3.966	0.970	9.11
iSIMC	77.0	63.6	2.678	1.742	2.967	5.067	0.930	6.23
Cohen Coon	65.0	41.4	1.823	2.552	3.367	3.663	2.352	19.13
ZN prc R L	53.0	89.8	2.270	1.875	6.697	6.986	1.492	1.44
ZN prc k T theta	84.0	82.1	2.324	1.861	6.438	6.742	1.549	2.21
ZN UG est	60.0	74.7	2.198	1.927	5.602	8.061	1.358	7.66

Table 8: Median robustness and performance values, optimization estimation, 6th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	7.0	40.7	5.921	1.553	10.883	67.820	0.455	0.13
SIMC	76.0	60.8	2.531	1.752	3.025	3.584	1.331	8.75
iSIMC	75.0	63.8	2.350	1.821	2.795	4.117	1.421	5.91
Cohen Coon	57.0	38.8	1.675	2.791	3.764	4.224	3.449	20.83
ZN prc k T theta	68.0	93.2	2.480	1.744	9.168	8.804	1.881	6.31
ZN UG est	55.0	86.3	2.019	2.107	10.071	10.140	2.012	1.81

Table 9: Median robustness and performance values, procest, 6th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	4.0	26.9	1.638	2.658	5.216	9.402	9.632	0.98
SIMC	73.0	60.5	2.787	1.739	2.900	4.393	0.966	2.94
iSIMC	63.0	64.8	2.435	1.809	2.521	3.922	1.220	2.84
Cohen Coon	49.0	41.0	1.692	2.852	3.420	3.542	3.621	8.11
ZN prc k T theta	62.0	93.0	2.837	1.579	12.976	12.308	1.170	4.52
ZN UG est	25.0	80.0	1.734	2.553	3.815	2.767	3.039	1.93

3 9th order SSM, PI controller

This section gives the result from tuning controllers based on random 9th order SSM.

Table 10: Median robustness and performance values, no estimation, 9th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
pidtune (PM=60)	98.0	60.0	2.824	1.662	5.062	9.204	0.774	0.25
pidtune (PM=50)	98.0	50.0	3.017	1.703	5.340	9.255	1.018	0.21
pidtune (PM=40)	98.0	56.4	4.867	1.552	9.444	20.895	0.880	0.23
pidtune ($\omega_c = 0.5$)	74.0	60.0	2.728	1.719	2.777	7.443	1.003	7.21
pidtune ($\omega_c = 0.8$)	62.0	60.0	1.785	2.408	3.637	6.615	2.146	7.36
pidtune ($\omega_c = 1$)	61.0	60.0	1.485	3.251	8.648	18.858	3.840	20.29
mftun ($\rho = 0.5$)	65.0	35.5	1.593	2.985	5.465	7.617	5.516	57.22
mftun ($\rho = 1$)	87.0	60.7	2.620	1.733	4.706	9.105	0.836	5.33
mftun ($\rho = 1.5$)	89.0	69.6	3.820	1.420	6.124	12.651	0.464	2.65
mftun ($\rho = 2$)	89.0	74.2	5.006	1.297	7.146	16.567	0.443	0.37
mftun ($\rho = 2.5$)	91.0	77.8	6.179	1.234	9.246	20.590	0.476	0.99
mftun ($\rho = 3$)	91.0	80.0	7.415	1.191	10.321	24.600	0.438	0.42
mftun ($\rho = 3.5$)	92.0	81.0	8.616	1.166	12.087	28.479	0.481	4.22
Opt 1	83.0	51.4	2.242	2.011	6.810	12.040	1.785	1.51
SSM Opt (Set 1)	87.0	49.3	2.171	2.078	6.669	9.830	1.131	2.89
SSM Opt (Set 2)	87.0	53.3	2.316	2.020	4.958	9.099	1.078	2.62
SSM Opt (Set 3)	87.0	48.5	2.024	2.174	5.346	8.215	1.415	2.87
SSM Opt (Set 4)	87.0	47.6	2.033	2.196	5.389	7.020	1.366	2.81
Megatuner	85.0	51.4	3.514	1.675	5.605	12.080	1.009	0.21
ZN UG	49.0	85.5	2.142	1.956	56.367	76.095	1.107	0.92
Delta tuning, IPTD approx	44.0	66.8	2.541	2.093	13.691	13.272	3.811	9.26

Table 11: Median robustness and performance values, graphical estimation, 9th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	4.0	18.0	2.368	2.447	25.397	114.901	1.177	6.54
SIMC	76.0	58.9	3.636	1.544	3.934	9.843	0.450	1.92
iSIMC	77.0	63.9	3.066	1.591	3.855	10.666	0.426	0.81
Cohen Coon	71.0	45.5	2.111	2.192	4.892	8.725	1.194	4.09
ZN prc R L	48.0	90.2	2.958	1.542	10.815	20.045	0.426	2.32
ZN prc k T theta	85.0	86.7	2.703	1.696	6.372	12.598	0.765	9.72
ZN UG est	41.0	72.0	2.200	1.963	15.062	11.370	1.073	3.30

Table 12: Median robustness and performance values, optimization estimation, 9th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	7.0	40.3	3.910	1.580	25.690	290.297	0.305	0.09
SIMC	67.0	58.3	2.848	1.706	3.146	8.223	0.607	3.85
iSIMC	64.0	60.8	2.584	1.724	3.008	8.364	0.615	5.75
Cohen Coon	53.0	28.2	1.562	3.225	4.523	5.860	2.422	4.58
ZN prc k T theta	63.0	64.2	2.001	2.390	6.469	11.477	1.527	8.03
ZN UG est	47.0	81.0	2.087	2.084	174.861	69.766	1.197	5.51

Table 13: Median robustness and performance values, procest, 9th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	0							
SIMC	70.0	60.7	3.173	1.640	3.354	10.135	0.361	3.39
iSIMC	60.0	63.3	2.668	1.709	2.849	8.259	0.487	9.00
Cohen Coon	52.0	36.1	1.609	2.898	5.279	5.982	1.655	2.93
ZN prc k T theta	62.0	90.4	2.431	1.902	6.979	11.106	0.652	1.43
ZN UG est	18.0	59.6	1.890	2.442	2.679	4.594	1.483	1.57

4 12th order SSM, PI controller

This section gives the result from tuning controllers based on random 12th order SSM.

Table 14: Median robustness and performance values, no estimation, 12th order model

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
pidtune (PM=60)	99.0	60.0	2.890	1.648	6.470	11.701	0.852	0.26
pidtune (PM=50)	100.0	50.0	2.909	1.720	6.433	13.107	0.942	0.22
pidtune (PM=40)	100.0	54.5	2.996	1.739	9.525	22.403	0.807	0.25
pidtune ($\omega_c = 0.5$)	69.0	60.0	2.615	1.781	3.624	11.894	0.881	15.14
pidtune ($\omega_c = 0.8$)	60.0	60.0	1.713	2.580	4.527	15.531	2.370	12.94
pidtune ($\omega_c = 1$)	60.0	54.9	1.384	3.773	16.685	41.458	6.136	20.70
mftun ($\rho = 0.5$)	59.0	31.2	1.562	3.693	7.768	11.286	5.740	15.67
mftun ($\rho = 1$)	75.0	55.9	2.588	1.824	5.187	12.227	1.466	3.45
mftun ($\rho = 1.5$)	80.0	66.3	3.584	1.491	5.949	14.424	0.819	1.13
mftun ($\rho = 2$)	82.0	71.9	4.765	1.346	7.280	17.927	0.615	2.66
mftun ($\rho = 2.5$)	83.0	75.8	5.952	1.271	8.244	21.482	0.624	2.27
mftun ($\rho = 3$)	84.0	78.0	7.132	1.222	8.680	23.983	0.606	0.67
mftun ($\rho = 3.5$)	83.0	79.7	8.310	1.195	9.870	25.879	0.624	0.41
Opt 1	56.0	52.1	2.175	1.936	5.569	9.962	1.678	1.63
SSM Opt (Set 1)	88.0	47.6	2.272	2.030	5.373	10.489	1.140	5.47
SSM Opt (Set 2)	85.0	49.5	2.363	1.974	5.738	10.413	1.112	8.12
SSM Opt (Set 3)	85.0	48.6	2.261	2.092	5.706	10.415	1.274	4.25
SSM Opt (Set 4)	87.0	47.2	2.259	2.133	5.351	9.053	1.324	5.86
Megatuner	70.0	51.7	3.141	1.735	5.544	10.628	1.026	0.24
ZN UG	39.0	64.6	2.038	2.090	21.598	13.122	1.767	2.03
Delta tuning, IPTD approx	47.0	78.6	3.063	1.735	13.330	12.127	1.389	2.49

Table 15: Median robustness and performance values, graphical estimation, 12th order models

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	10.0	21.8	3.178	2.327	22.041	63.358	0.325	11.16
SIMC	68.0	56.5	3.077	1.689	4.274	15.836	0.519	8.86
iSIMC	72.0	62.4	3.216	1.623	4.751	16.587	0.385	8.97
Cohen Coon	66.0	40.5	2.253	2.302	4.371	11.265	0.887	4.62
ZN prc R L	41.0	85.7	3.508	1.538	35.121	24.286	0.374	2.11
ZN prc k T theta	80.0	79.0	2.819	1.637	7.372	17.393	0.658	1.48
ZN UG est	29.0	74.5	2.200	1.846	173.878	59.701	0.902	0.92

Table 16: Median robustness and performance values, optimization estimation, 12th order models

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	15.0	39.8	3.132	1.621	25.946	198.932	0.431	5.09
SIMC	59.0	59.4	2.690	1.770	3.266	9.431	0.862	2.47
iSIMC	59.0	62.5	2.511	1.791	3.537	10.031	0.899	2.10
Cohen Coon	49.0	34.4	1.696	2.731	4.829	10.185	2.431	3.03
ZN prc k T theta	56.0	86.4	2.512	1.809	13.833	24.355	1.233	5.40
ZN UG est	30.0	57.8	2.196	2.112	117.036	94.774	1.455	6.52

Table 17: Median robustness and performance values, procest, 12th order model

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	4	1.8	61991	338.7	33926	4.5e+14	2.5661	
SIMC	69.0	58.3	2.986	2.049	3.875	12.460	0.519	3.34
iSIMC	54.0	62.9	2.405	1.873	3.267	8.394	1.110	4.08
Cohen Coon	41.0	34.9	1.768	2.861	4.058	8.620	2.555	3.67
ZN prc k T theta	51.0	93.9	2.866	1.616	17.283	24.303	0.709	14.13
ZN UG est	13.0	54.0	2.017	2.158	2.945	4.243	1.903	1.35

5 15th order SSM, PI controller

This section gives the result from tuning controllers based on random 15th order SSM.

Table 18: Median robustness and performance values, no model estimation, 15th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
pidtune (PM=60)	100.0	60.0	2.724	1.665	5.075	13.707	1.063	0.25
pidtune (PM=50)	100.0	50.0	2.915	1.672	6.285	13.825	1.042	0.18
pidtune (PM=40)	100.0	56.4	4.055	1.562	8.831	19.424	0.946	0.22
pidtune ($\omega_c = 0.5$)	72.0	60.0	2.282	1.890	4.015	14.533	1.370	2.11
pidtune ($\omega_c = 0.8$)	60.0	60.0	1.544	3.003	5.519	23.520	3.067	5.56
pidtune ($\omega_c = 1$)	54.0	60.0	1.479	3.319	15.225	57.672	5.162	9.07
mftun ($\rho = 0.5$)	62.0	37.9	1.563	3.242	5.586	9.908	4.286	65.04
mftun ($\rho = 1$)	87.0	58.3	2.582	1.734	7.012	15.243	1.442	4.50
mftun ($\rho = 1.5$)	94.0	66.7	3.817	1.437	6.305	15.278	1.177	1.72
mftun ($\rho = 2$)	95.0	71.9	5.068	1.315	7.147	19.898	0.831	0.80
mftun ($\rho = 2.5$)	95.0	75.7	6.334	1.247	8.460	24.260	0.684	0.34
mftun ($\rho = 3$)	95.0	78.2	7.601	1.205	10.126	24.450	0.630	0.23
mftun ($\rho = 3.5$)	95.0	79.9	8.868	1.175	11.808	27.885	0.642	0.18
Opt 1	25.0	54.5	2.316	1.913	4.554	9.096	1.683	0.95
SSM Opt (Set 1)	79.0	51.4	2.283	1.997	4.761	8.627	1.084	0.78
SSM Opt (Set 2)	78.0	53.7	2.489	1.909	4.835	8.576	0.970	0.74
SSM Opt (Set 3)	79.0	51.8	2.154	2.118	4.831	8.761	1.168	1.15
SSM Opt (Set 4)	78.0	50.4	2.025	2.133	4.655	8.523	1.202	1.19
Megatuner	89.0	51.6	3.506	1.701	6.301	12.404	1.280	0.22
ZN UG	41.0	70.3	2.008	2.068	10.015	17.751	1.200	2.38
Delta tuning, IPTD approx	34.0	77.0	2.254	1.892	14.960	16.469	3.998	3.38

Table 19: Median robustness and performance values, graphical estimation, 15th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	5.0	56.9	2.667	1.647	4.773	27.467	0.405	0.58
SIMC	80.0	59.4	3.469	1.566	4.461	14.565	0.389	4.03
iSIMC	83.0	61.3	3.194	1.591	6.046	18.041	0.375	3.21
Cohen Coon	72.0	46.6	2.339	2.100	7.318	15.691	0.752	19.62
ZN prc R L	57.0	94.6	4.136	1.424	30.500	26.973	0.292	2.40
ZN prc k T theta	86.0	91.5	3.108	1.506	8.177	18.388	0.532	2.00
ZN UG est	38.0	80.7	2.051	2.041	14.996	24.416	0.928	3.85

Table 20: Median robustness and performance values, optimization estimation, 15th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	8.0	35.7	3.438	1.881	21.683	246.489	0.205	0.90
SIMC	57.0	60.8	2.513	1.815	3.745	7.601	0.911	2.45
iSIMC	57.0	62.2	2.300	1.969	4.150	11.155	0.870	8.59
Cohen Coon	46.0	31.0	1.584	3.086	5.282	8.945	2.015	14.45
ZN prc k T theta	55.0	75.6	2.293	1.934	13.598	19.142	1.095	10.48
ZN UG est	38.0	86.1	1.873	2.408	161.545	142.859	1.241	3.43

Table 21: Median robustness and performance values, procest 15th order

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
delta	3.0	28.6	1.772	3.768	4.554	8.756	1.391	1.05
SIMC	66.0	61.1	2.772	1.841	4.406	12.144	0.548	7.64
iSIMC	57.0	63.6	2.418	1.856	4.064	12.464	0.698	2.89
Cohen Coon	46.0	42.9	1.649	3.108	7.249	14.168	1.946	5.01
ZN prc k T theta	58.0	90.6	2.891	1.778	27.695	27.390	0.707	22.41
ZN UG est	15.0	43.2	1.430	4.127	4.917	11.222	1.919	4.64

6 Average values by estimation method

Table 22: Mean robustness and performance values, using graphical parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	6.6	38.6126	2.855	3.106	12.234	45.843	1.02	4.2
SIMC	78	59.3824	3.2216	1.6228	3.6322	9.311	0.9006	5.2
iSIMC	79.8	63.088	2.9878	1.6454	3.9926	10.5494	0.7676	4.9
CC	72.2	43.0632	2.055	2.3606	4.5782	8.172	2.0128	10.1
ZN prc	86.4	85.3066	2.6476	1.7172	6.6332	11.5912	1.424	3.3
ZN UG est	49	80.3908	2.1698	1.9244	44.7354	22.0754	1.36366	3.4

Table 23: Mean robustness and performance values, using optimization parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta	8.6	37.2149	4.07088	1.8048	19.18704	162.45636	0.85518	1.5
SIMC	67.8	60.19294	2.71112	1.73422	3.09834	6.10066	1.18634	4.1
iSIMC	67.2	62.87174	2.47884	1.80936	3.16274	7.15776	1.18422	4.8
CC	55.2	33.03716	1.62706	2.99708	4.34422	6.22324	3.39318	9.5
ZN prc	63.8	78.8979	2.25494	2.02156	9.45622	13.29008	1.97052	6.6
ZN UG est	48.4	82	2.07242	2.11282	126.3018	65.87382	1.7623	4.6

Table 24: Mean robustness and performance values, using procest for parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV	std dev Ms
Delta*	3.33	23.9717	1.7965	3.5716	5.4289	6.8724	6.7193	1.0
SIMC	72.2	60.26874	2.95224	1.79954	3.36548	8.22802	0.84584	5.5
iSIMC	61.6	63.90064	2.51404	1.79214	2.95938	6.90982	1.16026	3.9
CC	50.8	38.45222	1.67248	2.96144	4.52914	6.51378	3.33978	4.8
ZN prc	61.4	90.64662	2.65062	1.81844	13.90612	15.576	1.30736	9.3
ZN UG est	21	61.572	1.7993	2.701	3.5537	4.5067	3.3634	2.3

Appendix C

Tuning PID controllers, results

Results from tuning PID controllers based on SSM. These results are discussed in section 5.6.

Tuning PID controllers using SSM, results

Preben Sandve Solvang

June 12, 2019

This documents was created as an appendix to the master thesis "State Space Model Based PID Controller Tuning", by Preben Sandve Solvang. The document contains measurements of performance and robustness for PID controllers tuned using a selection of different methods. The PID controllers are tuned based on randomly generated state space models of order 3, 6, 9, 12 and 15. 100 models of each order was used, and the values represented here are median values. The median gives the best measure, as some measurements contain outliers which gives inaccurate mean values. Table 1 contains the different values for the tuning weights for the optimization method.

Table 1: Optimization tuning settings

	W_{sp}	W_{dr}	W_{tv}
set 1	1	1	1
Set 2	1	1	2
Set 3	2	2	1
Set 4	1	2	1

1 3rd order SSM, PID controller

This section gives the result from tuning controllers based on random 3rd order SSM.

Table 2: Median robustness and performance values, No model estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
pidtune (PM=60)	100.0	60.0	2.486	1.777	2.352	2.023	2.585
pidtune (PM=50)	100.0	59.3	2.448	1.809	2.436	1.984	2.701
pidtune (PM=40)	100.0	58.6	2.372	1.849	2.395	2.046	3.031
pidtune ($\omega_c = 0.5$)	92.0	60.0	3.052	1.614	2.340	2.083	2.410
pidtune ($\omega_c = 0.8$)	88.0	60.0	2.009	2.087	2.350	1.656	5.364
pidtune ($\omega_c = 1$)	82.0	60.0	1.699	2.470	2.866	1.977	10.056
Opt 1	91.0	57.5	2.121	1.920	2.110	0.944	5.212
SSM Opt (Set 1)	96.0	63.2	2.627	1.682	2.454	1.880	2.043
SSM Opt (Set 2)	96.0	66.0	2.874	1.597	2.900	2.227	1.922
SSM Opt (Set 3)	96.0	62.4	2.501	1.726	2.695	1.471	3.507
SSM Opt (Set 4)	97.0	61.3	2.538	1.760	2.859	1.417	3.034
ZN UG	36.0	58.8	1.619	2.621	2.755	0.684	11.952
Delta PRC	36.0	90.9	2.137	1.880	16.242	7.230	2.355
Delta tuning, DIPTD approx	87.0	90.5	10.505	1.136	81.454	40.814	0.273

Table 3: Median robustness and performance values, graphical estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	91.0	63.2	2.649	1.665	2.157	1.901	2.637
Cohen Coon	79.0	51.4	1.459	3.239	2.763	1.078	10.197
ZN prc R L	43.0	71.0	1.500	3.039	3.375	1.071	15.889
ZN prc k T theta	87.0	69.4	1.536	2.875	2.993	1.561	10.618
ZN UG est	33.0	58.2	1.612	2.699	2.320	0.769	16.113

Table 4: Median robustness and performance values, optimization estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	81.0	61.3	2.985	1.609	2.105	1.621	2.891
iSIMC	74.0	63.8	2.444	1.762	2.050	1.594	3.710
Cohen Coon	47.0	46.2	1.423	3.501	2.520	1.182	12.149
ZN prc k T theta	49.0	65.9	1.470	3.198	2.729	1.883	10.470
ZN UG est	21.0	48.6	1.581	2.731	2.692	0.762	10.553

Table 5: Median robustness and performance values, procest, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	74.0	60.8	2.989	1.614	2.151	1.282	2.424
iSIMC	70.0	62.8	2.535	1.722	1.910	1.012	4.079
Cohen Coon	56.0	50.9	1.499	3.139	2.529	0.700	14.713
ZN prc k T theta	60.0	85.0	1.638	2.694	3.043	1.642	9.086
ZN UG est	22.0	59.5	1.631	2.617	1.975	0.537	14.902

2 6rd order SSM, PID controller

Table 6: Median robustness and performance values, no model estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
pidtune (PM=60)	99.0	60.0	2.688	1.706	3.496	3.659	1.471
pidtune (PM=50)	100.0	59.3	2.541	1.778	3.603	3.882	1.597
pidtune (PM=40)	100.0	58.5	2.458	1.813	3.575	4.008	1.656
pidtune ($\omega_c = 0.5$)	78.0	60.0	2.691	1.701	2.558	2.987	1.499
pidtune ($\omega_c = 0.8$)	67.0	60.0	1.926	2.162	2.907	2.630	2.845
pidtune ($\omega_c = 1$)	66.0	60.0	1.612	2.731	3.796	4.606	5.107
Opt 1	92.0	56.5	2.024	2.014	2.757	2.172	3.556
SSM Opt (Set 1)	92.0	57.9	2.238	1.862	2.700	2.470	1.729
SSM Opt (Set 2)	90.0	60.3	2.387	1.802	3.298	3.008	1.680
SSM Opt (Set 3)	90.0	56.9	2.101	1.972	3.687	2.804	3.031
SSM Opt (Set 4)	91.0	55.8	2.098	1.954	3.836	2.816	2.876
ZN UG	38.0	60.2	1.650	2.548	2.644	2.342	3.198
Delta PRC	34.0	86.6	1.836	2.330	8.077	6.359	2.396
Delta tuning, DIPTD approx	73.0	90.8	3.273	1.551	77.473	56.937	0.520

Table 7: Median robustness and performance values, graphical estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	72.0	63.5	2.429	1.830	2.846	4.287	1.315
Cohen Coon	52.0	50.5	1.457	3.228	3.252	3.710	3.992
ZN prc R L	45.0	72.4	1.526	3.054	4.544	4.685	3.593
ZN prc k T theta	64.0	64.5	1.450	3.303	3.356	4.038	4.471
ZN UG est	28.0	49.7	1.487	3.109	2.661	2.318	5.486

Table 8: Median robustness and performance values, optimization estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	64.0	60.9	2.532	1.805	2.560	3.011	1.355
iSIMC	56.0	63.4	2.243	1.990	2.150	3.201	1.983
Cohen Coon	38.0	57.1	1.512	3.090	2.901	2.686	4.940
ZN prc k T theta	43.0	93.9	1.650	2.553	4.831	5.912	4.176
ZN UG est	22.0	55.9	1.510	3.097	2.632	1.591	5.328

Table 9: Median robustness and performance values, procest, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	64.0	61.1	2.595	1.715	2.460	3.933	1.204
iSIMC	53.0	62.7	2.309	1.849	2.188	3.037	1.485
Cohen Coon	43.0	65.3	1.622	2.840	3.121	3.430	3.927
ZN prc k T theta	49.0	91.1	2.151	2.280	7.168	9.381	1.438
ZN UG est	22.0	54.1	1.520	2.947	2.625	1.601	5.545

3 9th order SSM, PID controller

Table 10: Median robustness and performance values, no model estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
pidtune (PM=60)	98.0	60.0	2.811	1.685	4.822	8.355	0.743
pidtune (PM=50)	99.0	59.3	2.548	1.753	5.288	8.784	1.049
pidtune (PM=40)	99.0	58.5	2.605	1.789	5.352	9.609	0.937
pidtune ($\omega_c = 0.5$)	71.0	60.0	2.833	1.652	2.802	7.207	0.914
pidtune ($\omega_c = 0.8$)	55.0	60.0	1.904	2.153	2.764	5.730	1.736
pidtune ($\omega_c = 1$)	49.0	60.0	1.597	2.712	3.314	6.588	2.780
Opt 1	97.0	49.3	1.832	2.285	3.768	4.468	3.491
SSM Opt (Set 1)	86.0	51.6	2.033	2.092	2.990	4.042	1.525
SSM Opt (Set 2)	85.0	54.4	2.171	1.968	3.450	5.120	1.598
SSM Opt (Set 3)	84.0	49.0	1.825	2.258	3.906	5.402	2.968
SSM Opt (Set 4)	85.0	48.6	1.874	2.237	4.008	5.289	2.747
ZN UG	24.0	58.2	1.551	2.822	3.005	3.862	3.694
Delta PRC	21.0	62.2	1.731	2.368	6.239	5.437	2.217
Delta tuning, DIPTD approx	60.0	91.0	3.307	1.526	67.690	126.412	0.285

Table 11: Median robustness and performance values, graphical estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	74.0	62.1	2.736	1.698	3.495	8.987	0.736
Cohen Coon	63.0	56.3	1.654	2.748	4.785	5.983	2.815
ZN prc R L	39.0	90.0	1.654	2.530	10.746	28.263	1.212
ZN prc k T theta	62.0	80.5	1.556	2.899	4.194	7.090	2.040
ZN UG est	19.0	53.8	1.458	3.221	2.406	3.415	4.327

Table 12: Median robustness and performance values, optimization estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	57.0	59.2	2.561	1.802	3.798	6.380	0.780
iSIMC	47.0	63.0	2.317	1.874	2.680	6.791	1.008
Cohen Coon	32.0	42.2	1.364	4.077	3.432	4.455	4.359
ZN prc k T theta	32.0	65.9	1.414	3.762	3.488	4.858	3.063
ZN UG est	12.0	53.3	1.464	3.236	2.218	2.823	4.330

Table 13: Median robustness and performance values, procest, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	59.0	62.0	2.898	1.628	3.120	8.052	0.549
iSIMC	52.0	63.6	2.584	1.761	2.881	8.060	0.711
Cohen Coon	42.0	51.1	1.565	2.999	4.388	6.790	2.600
ZN prc k T theta	49.0	90.0	1.850	2.317	25.390	8.570	1.496
ZN UG est	12.0	51.5	1.472	3.184	2.243	2.751	4.304

4 12th order SSM, PID controller

Table 14: Median robustness and performance values, no model estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
pidtune (PM=60)	99.0	60.0	2.790	1.659	5.951	13.512	0.800
pidtune (PM=50)	99.0	59.3	2.619	1.741	6.050	12.390	0.928
pidtune (PM=40)	99.0	58.5	2.649	1.749	6.659	13.524	0.888
pidtune ($\omega_c = 0.5$)	60.0	60.0	2.696	1.716	3.292	11.299	0.781
pidtune ($\omega_c = 0.8$)	49.0	60.0	1.951	2.146	3.033	7.967	1.958
pidtune ($\omega_c = 1$)	45.0	60.0	1.639	2.665	3.616	11.916	2.658
Opt 1	89.0	47.3	1.735	2.420	3.730	5.891	3.460
SSM Opt (Set 1)	87.0	49.1	1.938	2.201	3.246	5.125	1.563
SSM Opt (Set 2)	88.0	51.9	2.113	2.000	4.789	7.403	1.725
SSM Opt (Set 3)	87.0	47.1	1.881	2.273	5.429	8.624	2.662
SSM Opt (Set 4)	86.0	47.3	1.875	2.278	5.524	8.110	2.609
ZN UG	13.0	46.7	1.520	2.924	2.379	2.955	5.097
Delta PRC	11.0	70.4	1.823	2.222	4.156	3.580	2.439
Delta tuning, DIPTD approx	56.0	90.7	2.593	1.808	72.072	176.261	0.234

Table 15: Median robustness and performance values, graphical estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	66.0	61.3	2.657	1.808	4.648	15.753	0.789
Cohen Coon	53.0	45.1	1.480	3.204	4.393	10.951	3.243
ZN prc R L	32.0	69.4	1.656	2.704	99.383	184.537	2.217
ZN prc k T theta	60.0	65.8	1.545	3.062	6.812	10.387	3.625
ZN UG est	8.0	47.8	1.732	2.421	2.483	2.379	4.140

Table 16: Median robustness and performance values, optimization estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	48.0	59.4	2.237	1.899	3.183	8.082	1.051
iSIMC	44.0	61.5	1.868	2.167	2.890	6.900	1.467
Cohen Coon	29.0	51.0	1.476	3.216	3.696	5.661	4.837
ZN prc k T theta	30.0	93.0	1.588	2.708	7.221	10.458	2.029
ZN UG est	8.0	55.1	1.710	2.455	2.124	2.495	3.559

Table 17: Median robustness and performance values, procest, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	49.0	58.9	2.137	2.153	3.448	9.945	1.116
iSIMC	45.0	60.7	2.230	2.116	2.943	8.895	1.488
Cohen Coon	33.0	51.5	1.684	2.840	3.795	7.455	3.620
ZN prc k T theta	37.0	91.4	2.057	2.188	12.383	14.069	1.085
ZN UG est	8.0	53.7	1.705	2.468	2.137	2.460	3.618

5 15rd order SSM, PID controller

Table 18: Median robustness and performance values, no model estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
pidtune (PM=60)	100.0	60.0	2.824	1.686	5.564	13.326	0.991
pidtune (PM=50)	100.0	59.2	2.583	1.760	4.800	12.307	1.108
pidtune (PM=40)	100.0	58.5	2.770	1.719	4.985	12.345	1.140
pidtune ($\omega_c = 0.5$)	69.0	60.0	2.318	1.876	3.760	12.814	1.232
pidtune ($\omega_c = 0.8$)	53.0	60.0	1.763	2.366	4.917	15.626	2.337
pidtune ($\omega_c = 1$)	46.0	60.0	1.576	2.842	6.041	18.045	3.729
Opt 1	83.0	50.3	1.867	2.164	4.386	6.269	3.036
SSM Opt (Set 1)	81.0	52.7	2.028	2.082	3.583	4.806	1.450
SSM Opt (Set 2)	82.0	54.7	2.205	1.924	4.408	6.597	1.426
SSM Opt (Set 3)	81.0	51.5	1.929	2.223	4.945	7.108	2.250
SSM Opt (Set 4)	81.0	51.4	1.948	2.144	4.961	7.133	2.281
ZN UG	15.0	47.6	1.431	3.321	4.792	6.003	8.052
Delta PRC	13.0	13.1	1.412	5.451	13.563	23.612	6.575
Delta tuning, DIPTD approx	46.0	91.1	2.457	1.798	81.527	125.509	0.254

Table 19: Median robustness and performance values, graphical estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	75.0	61.3	2.559	1.777	5.243	17.434	0.707
Cohen Coon	55.0	55.6	1.732	2.689	6.849	13.198	1.510
ZN prc R L	41.0	88.9	942640.266	2.193	100.01	687 042	2.03e-7
ZN prc k T theta	59.0	73.6	1.609	2.741	9.121	16.784	1.282
ZN UG est	10.0	53.3	1.606	2.747	2.452	3.991	2.179

Table 20: Median robustness and performance values, optimization estimation, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	53.0	59.7	2.188	2.041	4.135	9.744	1.763
iSIMC	39.0	61.0	1.756	2.629	3.511	11.893	1.172
Cohen Coon	19.0	40.6	1.663	2.928	3.124	6.104	2.138
ZN prc k T theta	21.0	70.0	1.743	2.371	5.228	10.187	1.667
ZN UG est	6.0	50.7	1.409	3.470	2.623	4.380	2.641

Table 21: Median robustness and performance values procest, PID controller

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	54.0	61.5	2.390	1.808	3.941	11.675	1.029
iSIMC	47.0	61.4	2.343	1.909	4.117	14.718	1.078
Cohen Coon	33.0	85.1	3.015	2.208	13.247	99.704	0.946
ZN prc k T theta	37.0	91.1	5.730	1.534	98.854	49.977	0.523
ZN UG est	6.0	49.5	1.411	3.453	2.638	4.300	2.757

6 Average values by estimation method

Table 22: Mean robustness and performance values, using graphical parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV
iSIMC	75.6	62.27896	2.60588	1.75552	3.678	9.6724	1.23686
CC	60.4	51.77164	1.55648	3.0218	4.40848	6.9839	4.35132
ZN prc	66.4	70.76612	1.53912	2.97612	5.2952	7.97186	4.40722
ZN UG est	19.6	52.5668	1.57916	2.83926	2.46432	2.57442	6.44898

Table 23: Mean robustness and performance values, using optimization parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	60.6	60.09046	2.50058	1.83122	3.1561	5.76756	1.56802
iSIMC	52	62.53788	2.12532	2.08432	2.65606	6.07586	1.86776
CC	33	47.43622	1.48754	3.3621	3.13464	4.01748	5.6847
ZN prc	35	77.72956	1.57314	2.91838	4.69948	6.65974	4.2809
ZN UG est	13.8	52.72878	1.53478	2.99762	2.45764	2.41014	5.28208

Table 24: Mean robustness and performance values, using procest for parameter estimation

	Stable	PM	GM	Ms	IAEr	IAEv	TV
SIMC	60	60.84872	2.60176	1.78344	3.0239	6.97726	1.26468
iSIMC	53.4	62.2303	2.40028	1.87164	2.8081	7.14452	1.768
CC	41.4	60.79156	1.87692	2.8053	5.41594	23.61602	5.1614
ZN prc	46.4	89.72512	2.68542	2.20252	29.36764	16.72802	2.7258
ZN UG est	14	53.64636	1.54762	2.93382	2.32378	2.3299	6.22516

Appendix D

Matlab code

The Matlab code used for tuning, comparison and calculating performance are attached in this appendix. The code used to obtain the results presented in this thesis might be slight variations of the code presented here. The code used for delta tuning, comes from the papers written by Di Ruscio and Dalen, which are referred to in the thesis. The code given in these papers might also have been altered in ways that would not effect the calculation of the controller parameters.

- Functions for estimating model parameters from SSM to use for controller tuning
  - GRAPH_ hp_ Est(Y,t,hpType)
  - OPT_ hp_ Est(Y,t,hpType)
  - Model_ red_ dalenruscio2017(hp,hpType)
- Functions used for tuning controllers based on obtained model parameters
  - ZNpidtuneUG(hp,hcType)
  - ZNpidtunePRC(k,T,theta,R,hcType,tunType)
  - SIMCpidtune2(k,T1,T2,theta,Type)
  - CCpidtune2(K,T,theta,hcType)
- Functions used for tuning controllers directly from SSM
  - OPTpidtune3(hp,hcType,t,dt,n,Wsp,Wdr,Wtv) (SSM Opt)
  - OPTpidtune2(hp,hcType,Wsp,Wdr,Wms) (TF Opt)
- Functions used for closed loop simulation and to calculate performance and robustness values from tuned controllers
  - CL_ step_ pid(A,B,C,hc,theta,t,dt,n)
- Other

- OPTweights(hp,t,dt)
- PO_ pidtune(hp,hcType,IAEryW,IAEvyW,t,dt,MsP)

Matlab methods for controller tuning

Preben Sandve Solvang

June 9, 2019

1 Methods for model estimation

```
function [hp,R,k,T1,T2,theta] = GRAPH_hp_Est(Y,t,hpType)
% Graphical model parameter estimation tool - Preben Solvang Master thesis 2019
%  Inputs:
%       Y: Output vector from SSM or measured from process
%       t: Time vector
% hpTtype: Plant type 1 = FOPTD, 2 = SOPTD, 3 = IPTD

    s = tf('s'); % Laplace operator s
    dt = t(2)-t(1); %stepsize
    dy = diff(Y); %Change in y
    [R,i] = max(abs(dy)/dt); % Steepest slope
    L = t(i)-abs(Y(i)-Y(1))/R; % Lag
    T2=0;
    theta = L;
    if theta <= 0 %Lag cant be less than zero -> =stepsize
        theta = dt;
    end
    switch hpType
        case 1 % FOPTD
            k = Y(end)-Y(1);
            T1 = abs(Y(end)-Y(1))/R;
            hp = k*(1/(T1*s+1))*exp(-theta*s);
        case 2    % SOPTD, not used

        case 3 % IPTD
            A = diff(Y)/dt; %difference from step to step
            k=A(end);
            hp = k*(1/s)*exp(-theta*s);
    end
end

function [hp,k,T1,T2,theta] = OPT_hp_Est(Y,t,hpType)
% Optimization based model parameter estimation tool - Preben Solvang Master thesis 2019
%  Inputs:
%       Y: Output vector from SSM or measured from process
%       t: Time vector
% hpTtype: Plant type 1 = FOPTD, 2 = SOPTD, 3 = IPTD, 4 = DIPTD
    s=tf('s'); % Laplace operator
    opt = optimset('display','off','TolX',1e-5,'TolFun',1e-
5,'LargeScale','off','MaxIter',500);
    switch hpType
        case 1 % FIT FOPTD
            x = [1 1 1];  % x = [K T theta]
            cost = @(x) FOPTDcost(x, Y, t); %define cost function
            [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-1000 0.1 0.1],[1000 1000 20],[],opt);
            k = x(1); T1=x(2); T2=0; theta=x(3); % assign model parameters to function output
            hp = k*(1/(T1*s+1))*exp(-theta*s); % estimated TF plant model
```

```
        case 2    % FIT SOPTD
            x = [1 1 1 1]; % x = [K T1 T2 theta]
            cost = @(x) SOPTDcost(x, Y, t); %define cost function
            A=[0 0 1 0];b=x(2); % T1 > T2
            [x,fval,flag]=fmincon(cost,x,A,b,[],[],[-inf 0.1 0.1 0.1],[inf inf inf 20],[],opt);
            k = x(1); T1=x(2); T2=x(3); theta=x(4); % assign model parameters to function output
            hp = k* (1/((T1*s + 1)*(T2*s + 1))) *exp(-theta*s); % estimated TF plant model
        case 3 % 3 = IPTD,
            x = [1 1]; % x = [K theta]
            cost = @(x) IPTDcost(x, Y, t);
            [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.1],[inf 20],[],opt);
            k = x(1); T1=0; T2=0; theta=x(2); % assign model parameters to function output
            hp = k*(1/s)*exp(-theta*s); % estimated TF plant model
        case 4 % 4 = DIPTD
            x = [1 1]; % x = [K theta]
            cost = @(x) DIPTDcost(x, Y, t);
            [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.1],[inf 10],[],opt);
            k = x(1); T1=0; T2=0; theta=x(2); % assign model parameters to function output
            hp = k*(1/s^2)*exp(-theta*s); % estimated TF plant model
    end
end

% Cost functions
function J = FOPTDcost(x,Y,t)
    % x = [K T theta]
    s=tf('s');
    FOPTD = x(1)*(1/(x(2)*s + 1))*exp(-x(3)*s);
    Yest = step(FOPTD,t); % Y for estimated model
    J = (Y - Yest).^2; % Squared error
    J = mean(J); % MSE performance index
end

function J = SOPTDcost(x,Y,t)
    % x = [K T1 T2 theta]
    s=tf('s');
    SOPTD = x(1)* (1/((x(2)*s + 1)*(x(3)*s + 1))) *exp(-x(4)*s);
    Yest = step(SOPTD,t);
    J = (Y - Yest).^2;
    J = mean(J);
end

function J = IPTDcost(x,Y,t)
    % x = [K theta]
    s=tf('s');
    IPTD = x(1)* (1/s) *exp(-x(2)*s);
    Yest = step(IPTD,t);
    J = (Y - Yest).^2;
    J = mean(J);
end

function J = DIPTDcost(x,Y,t)
    % x = [K theta]
    s=tf('s');
    DIPTD = x(1)* (1/s^2) *exp(-x(2)*s);
    Yest = step(DIPTD,t);
    J = (Y - Yest).^2;
    J = mean(J);
end

function [hp,k,theta,tfin] = Model_red_dalenruscio2017(hp,hpType)
```

```
% Optimization based model approximation tool - Preben Solvang Master thesis 2019
% Implementation from Dalen, Di Ruscio 2017: PD/PID controller tuning based on model
% approximations: Model reduction of some unstable and higher order nonlinear models
%
%  Inputs:
%      hp: Plant model
% hpTtype: Plant type 1 = IPTD, 2 = DIPTD
    s=tf('s');
    dt=0.01;
    [Y,T]=step(hp);h=T(2)-T(1);
    A=diff(Y)/h;
    if Y(end)<0 % Choose lb and ub based on gain fortegn
        [R1, i] = min(A);
        c2=-0.01;c1=-10;
    elseif Y(end)>0
        [R1, i] = max(A);
        c1=0.01;c2=10;
    end
    y1=Y(i);t1=T(i);
    L=t1-y1/R1;
    if L<=0
        L=dt;
    end
    A=[1 0 0; 0 -1 0; 0 1 -1]; b=[c2; -dt/2; -2*dt;]; % COnstraints
    lb = [c1; dt/2; 2*dt]; ub=[c2; L/2; L/2];
    opt = optimset('display','off','TolX',1e-7,'TolFun',1e-
7,'LargeScale','off','MaxIter',500);
    x = [1 1 1];  % x = [K T tf] %initial guess for x
    switch hpType
        case 1 % FIT IPTD
            cost = @(x) IPTDcost(x,hp,dt);
            [x,fval,flag]=fmincon(cost,x,A,b,[],[],lb,ub,[],opt); %,opt
            k = x(1); theta=x(2); tfin=x(3);
            hp = (k/s)*exp(-theta*s);
        case 2    % FIT DIPTD
            cost = @(x) DIPTDcost(x,hp,dt);
            [x,fval,flag]=fmincon(cost,x,A,b,[],[],lb,ub,[],opt); % ,opt
            k = x(1); theta=x(2); tfin=x(3);
            hp = (k/s^2)*exp(-theta*s);
    end
end

% Cost functions
function J = IPTDcost(x,hp,dt)
    % x = [K theta tf]
    s=tf('s');
    t=0:dt:x(3);
    hpest = x(1)*(1/s)*exp(-x(2)*s);
    Y=step(hp,t);Yhat=step(hpest,t);
    J = (Y-Yhat)'*eye(length(Y))*(Y-Yhat);
    J = J/length(t);
end

function J = DIPTDcost(x,hp,dt)
    % x = [K theta tf]
    s=tf('s');
    t=0:dt:x(3);
    hpest = x(1)*(1/s^2)*exp(-x(2)*s);
    Y=step(hp,t);Yhat=step(hpest,t);
    J = (Y-Yhat)'*eye(length(Y))*(Y-Yhat);
```

```
    J = J/length(t);
end
```

2 Functions used for tuning controllers based on obtained model parameters

```
function [hc,Kp,Ti,Td] = ZNpidtuneUG(hp,hcType)
% Ziegler Nichols Ultimate gain PID tuning tool - Preben Solvang Master thesis 2019
%  Inputs:
%      hp: Plant model
% hcTtype: controller type 1 = P, 2 = PI, 3 = PID

if nargin == 1;hcType=2;end
    [Gm,Pm,Wcg]=margin(hp); %stability margins
    if Wcg <=0 %To prevent crash
        Wcg=0.01;
    end
    ku=Gm; %ultimate gain
    pu=2*pi/Wcg; %ultimate periode
    switch hcType % Controller type
        case 1
            Kp = ku/2;
            Ti = 0;
            Td = 0;
            hc = pidstd(Kp);
        case 2
            Kp = ku/2.2;
            Ti = pu/1.2;
            Td = 0;
            hc = pidstd(Kp,Ti);
        case 3
            Kp = ku/1.7;
            Ti = pu/2;
            Td = pu/8;
            hc = pidstd(Kp,Ti,Td);
    end
end

function [hc,Kp,Ti,Td] = ZNpidtunePRC(k,T,theta,R,hcType,tunType)
% Ziegler Nichols PRC PID tuning tool - Preben Solvang Master thesis 2019
%  Inputs:
%       k: Plant gain
%       T: Plant time constant
%   theta: Plant dead time
%       R: Reaction rate (only for tunType 1)
% hcTtype: controller type 1 = P, 2 = PI, 3 = PID
% tunType: Tuning method 1 = L and R, 2 = k, theta, T
    if nargin == 3;tunType=2;hcType=2; R=0;end
if tunType == 1
    L = theta;
    switch hcType
        case 1
            Kp = 1/(L*R);
            Ti = 0;
            Td = 0;
            hc = pidstd(Kp);
        case 2
            Kp = 0.9/(L*R);
            Ti = 3.3*L;
```

```
            Td = 0;
            hc = pidstd(Kp,Ti);
        case 3
            Kp = 1.2/(L*R);
            Ti = 2*L;
            Td = 0.5*L;
            hc = pidstd(Kp,Ti,Td);
    end
else
    switch hcType
        case 1
            Kp = T/(k*theta);
            Ti = 0;
            Td = 0;
            hc = pidstd(Kp);
        case 2
            Kp = 0.9*(T/(k*theta));
            Ti = 3.33*theta;
            Td = 0;
            hc = pidstd(Kp,Ti);
        case 3
            Kp = 1.2*(T/(k*theta));
            Ti = 2*theta;
            Td = 0.5*theta;
            hc = pidstd(Kp,Ti,Td);
    end
end
end

function [hc,Kp,Ti,Td] = SIMCpidtune2(k,T1,T2,theta,Type)
% SIMC PID tuning tool - Preben Solvang Master thesis 2019
%  Inputs:
%       k: Plant gain
%      T1: Plant time constant
%      T2: Plant time constant (for PID)
%   theta: Plant dead time
%   Ttype: 1 = FOPTD PI, 2 = SOPTD PID, 3 = IPTD, 4 = DIPTD PID, 5 = iSIMC
%          PI, 6 = iSIMC PID
s=tf('s');
switch Type
    case 1 %FOPTD, PI controller
        tc = theta;
        Kp = (1/k)*(T1/(tc+theta));
        Ti = min([T1 4*(tc+theta)]);
        hc = pidstd(Kp, Ti);
        Td = 0;
    case 2 %SOPTD, PID controller
        tc = theta;
        Kps = (1/k)*(T1/(tc+theta)); %Kp for serial controller
        Tis = min([T1 4*(tc+theta)]);  %Ti for serial controller
        Tds = T2; %Td for serial controller
        hc = (Kps*(Tis*s+1)*(Tds*s+1))/(Tis*s); %serial PID controller
        f = 1 + (Tds/Tis);
        % PID parameters for standard PID formulation
        Kp = Kps*f;
        Ti = Tis*f;
        Td = Tds/f;
    case 3 % IPTD, PI controller
        Kp = 1/(2*k*theta);
        Ti = 8*theta;
```

```
        hc = pidstd(Kp, Ti);
        Td = 0;
    case 4 % DIPTD, PID controller, no need for conversion in this case since f=1
        tc = 1.5*theta;
        Kp = (1/k)*(1/(4*(tc+theta)^2));
        Ti = 4*(tc+theta);
        Td = 4*(tc+theta);
        hc = (Kp*(Ti*s+1)*(Td*s+1))/(Ti*s);
    case 5 % iSIMC PI
        Kp = (1/k) * ((T1+(theta/3)) / (2*theta)); %Kp for serial controller
        Ti = min([T1+(theta/3) 8*theta]);
        hc = pidstd(Kp, Ti);
        Td = 0;
    case 6 % iSIMC PID
        tc = theta/2;
        Kps = (1/k)*(T1/(tc+theta));
        Tis = min([T1 4*(tc+theta)]);
        Tds = theta/3;
        hc = (Kps*(Tis*s+1)*(Tds*s+1))/(Tis*s); %serial PID controller
        % PID parameters for standard PID formulation
        f = 1 + (Tds/Tis);
        Kp = Kps*f;
        Ti = Tis*f;
        Td = Tds/f;
end
end

function [hc,Kp,Ti,Td] = CCpidtune2(K,T,theta,hcType)
% Cohen-Coon PID tuning tool - Preben Solvang Master thesis 2019
%  Inputs:
%       k: Plant gain
%      T1: Plant time constant
%   theta: Plant dead time
% hcTtype: controller type 1 = P, 2 = PI, 3 = PID
if nargin == 3;hcType=2;end

    switch hcType
        case 1              % P-controller
            Kp = (1/K)* (1 + (0.35*theta)/(1-theta));
            Ti = 0;
            Td = 0;
            hc = pidstd(Kp);
        case 2              % PI-controller
            Kp = (1/K)* (0.9*(T/theta) + 0.083);
            Ti = T*((3.33*(theta/T) + 0.31*(theta/T)^2) / (1 + 2.22*(theta/T)));
            Td = 0;
            hc = pidstd(Kp,Ti);
        case 3              % PID-controller
            Kp = (1/K)*(1.35*(T/theta) + 0.25);
            Ti = T*((2.5*(theta/T) + 0.46*(theta/T)^2) / (1 + 0.61*(theta/T)));
            Td = (0.37*theta)/(1 + 0.19*(theta/T));
            hc = pidstd(Kp,Ti,Td);
    end
end
```

3 Functions used for tuning controllers directly from SSM

```
function [hc,Kp,Ti,Td]=OPTpidtune3(hp,hcType,t,dt,n,Wsp,Wdr,Wtv)
% Optimization based PID tuning tool, using SSM - Preben Solvang Master thesis 2019
%  Inputs:
```

```
%      hp: Plant model
% hcTtype: controller type 2 = PI, 3 = PID
%     Wsp: Weight for SP tracking
%     Wdr: Weight for Disturbance rejection
%     Wtv: Weight for input usage TV
if nargin == 5;Wsp=1;Wdr=1;Wtv=1;end

% Initial controller found with pidtune
opt = pidtuneOptions('PhaseMargin', 40); % lower PM than default to get closer to the optimal setti
if hcType==2 %Initial PI controller
    c0=pidstd(1,1);
    hc = pidtune(hp,c0,opt);
    x = [hc.Kp, hc.Ti];
elseif hcType==3 %Initial PID conntroller
    c0=pidstd(1,1,1);
    hc = pidtune(hp,c0,opt);
    x = [hc.Kp, hc.Ti hc.Td];
end
theta = hp.inputdelay;
opt=optimset('display','off','TolX',1e-5,'TolFun',1e-5,'LargeScale','off','MaxIter',700);
% Optimization to find controller parameters
if hcType==2 % PI controller
    cost = @(x) costfuncPI(x, hp, theta, t, dt,n, Wsp, Wdr, Wtv); % x = [Kp Ti]
    [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001],[inf inf],[],opt);
else % PID controller
    cost = @(x) costfuncPID(x, hp, theta, t, dt,n, Wsp, Wdr, Wtv); % x = [Kp Ti Td]
    [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001 0],[inf inf inf],[],opt);
end
if hcType == 2
    hc = pidstd(x(1),x(2));
    Kp = hc.Kp;
    Ti = hc.Ti;
    Td = 0;
elseif hcType == 3
    hc = pidstd(x(1),x(2),x(3));
    Kp = hc.Kp;
    Ti = hc.Ti;
    Td = hc.Td;
end
end
% Cost functions
function J=costfuncPI(x, hp, theta, t, dt,n, Wsp, Wdr, Wtv)
    A=hp.a; B=hp.b; C=hp.c; % State space matrices
    [IAEr, IAEv, TV]= numric_steppi(A,B,C,x,theta,t,dt,n); % calc performance measures
    J = Wsp*IAEr + Wdr*IAEv + Wtv*TV; % Weighted performance index
end

function J=costfuncPID(x, hp, theta, t, dt,n, Wsp, Wdr, Wtv)
    A=hp.a; B=hp.b; C=hp.c;
    [IAEr, IAEv, TV]= numric_steppid(A,B,C,x,theta,t,dt,n);
    J = Wsp*IAEr + Wdr*IAEv + Wtv*TV;
end

function [IAEr, IAEv, TV]= numric_steppi(A,B,C,x,theta,t,dt,n)
    Kp=x(1); Ti=x(2);% PI controller
    r=1; % The reference signal
    N=length(t);
    nt=theta/dt; nt=round(nt); Delay=zeros(nt,1);
    hc=dt*Kp/Ti; % Controller param. to save computing time
    x=zeros(n,1);% Initial values for the states
```

```
    z=0; y_old=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        u=z+Kp*e; % The controller output PI controller
        %u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
        z=z+hc*e; % Updating the controller state
        Y1(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        %u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEr = sum(abs(E)*dt);
    TV = abs(diff(U));
    TV = sum(TV);

    Delay=zeros(nt,1);
    x=zeros(n,1);% Initial values for the states
    z=0; y_old=0;
    r=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        u=z+Kp*e; % The controller output PI controller
        %u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
        z=z+hc*e; % Updating the controller state
        Y2(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEv = sum(abs(E)*dt);
end

function [IAEr, IAEv, TV]= numric_steppid(A,B,C,x,theta,t,dt,n)
    Kp=x(1); Ti=x(2); Td=x(3); % PID controller
    r=1; % The reference signal
    N=length(t);
    nt=theta/dt; nt=round(nt); Delay=zeros(nt,1);
    hc=dt*Kp/Ti; % Controller param. to save computing time
    x=zeros(n,1);% Initial values for the states
    z=0; y_old=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        %u=z+Kp*e; % The controller output PI controller
        u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
```

```
        z=z+hc*e; % Updating the controller state
        Y1(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        %u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEr = sum(abs(E)*dt);
    TV = abs(diff(U));
    TV = sum(TV);

    Delay=zeros(nt,1);
    x=zeros(n,1);% Initial values for the states
    z=0; y_old=0;
    r=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        %u=z+Kp*e; % The controller output PI controller
        u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
        z=z+hc*e; % Updating the controller state
        Y2(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEv = sum(abs(E)*dt);
end

function [hc,Kp,Ti,Td]=OPTpidtune2(hp,hcType,Wsp,Wdr,Wms)
% Optimization based PID tuning tool, using TF - Preben Solvang Master thesis 2019
%  Inputs:
%      hp: Plant model
% hcTtype: controller type 2 = PI, 3 = PID
%     Wsp: Weight for SP tracking
%     Wdr: Weight for Disturbance rejection
%     Wms: Weight for Robusness measure Ms
if nargin == 2;Wsp=1;Wdr=1;Wms=0;end
if nargin == 1;Wsp=1;Wdr=1;Wms=0;hcType=2;end

% Initial controller found with pidtune
opt = pidtuneOptions('PhaseMargin', 40); % lower PM than default to get closer to the optimal sett
if hcType==2 %Initial PI controller
    c0=pidstd(1,1);
    hc = pidtune(hp,c0,opt);
    x = [hc.Kp, hc.Ti];
elseif hcType==3 %Initial PID conntroller
    c0=pidstd(1,1,1);
    hc = pidtune(hp,c0,opt);
    x = [hc.Kp, hc.Ti hc.Td];
end
% closed-loop response of initial tuning to find t and dt. T final is extended by
% 50 percent
[y1,t1]=step(hc*hp/(hc*hp+1));
dt=(t1(2)-t1(1));
t=0:dt:t1(end)*1.5;
opt=optimset('display','off','TolX',1e-5,'TolFun',1e-5,'LargeScale','off','MaxIter',700);
% Cost function
```

```
if hcType==2 % PI controller
    cost = @(x) costfuncPI(x, hp, t, dt, Wsp, Wdr, Wms); % x = [Kp Ti]
    [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001],[inf inf],[],opt);
else % PID controller
    cost = @(x) costfuncPID(x, hp, t, dt, Wsp, Wdr, Wms); % x = [Kp Ti Td]
    [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001 0],[inf inf inf],[],opt);
end
if hcType == 2
    hc = pidstd(x(1),x(2));
    Kp = hc.Kp;
    Ti = hc.Ti;
    Td = 0;
elseif hcType == 3
    hc = pidstd(x(1),x(2),x(3));
    Kp = hc.Kp;
    Ti = hc.Ti;
    Td = hc.Td;
end
end

function J=costfuncPI(x,hp,t,dt,Wsp,Wdr,Wms)
    hc = pidstd(x(1),x(2));
    spLoop = hc*hp/(hc*hp+1);
    DistLoop =hp/(hc*hp+1);
    S = 1/(hc*hp+1); % Sensitivity function
    F=logspace(-10,10,1e4); % Logaritmic scale
    Ms = max(abs(freqresp(S,F))); % Ms, Robustness
    eSP = 1-step(spLoop,t); %SP tracking error
    eDist = 0-step(DistLoop,t); %Disturbance rejection error
    % performance calculation
    J = Wsp*(sum(abs(eSP)*dt)) + Wdr*(sum(abs(eDist)*dt) + Wms*Ms);
end

function J=costfuncPID(x,hp,t,dt,Wsp,Wdr,Wms)
    hc = pidstd(x(1),x(2),x(3));
    spLoop = hc*hp/(hc*hp+1);
    DistLoop =hp/(hc*hp+1);
    S = 1/(hc*hp+1); % Sensitivity function
    F=logspace(-10,10,1e4); % Logaritmic scale
    Ms = max(abs(freqresp(S,F))); % Ms, Robustness
    eSP = 1-step(spLoop,t); %SP tracking error
    eDist = 0-step(DistLoop,t); %Disturbance rejection error
    % performance calculation
    J = Wsp*(sum(abs(eSP)*dt)) + Wdr*(sum(abs(eDist)*dt) + Wms*Ms);
end
```

4 Functions used for closed loop simulation and to calculate performance and robustness values from tuned controllers

```
%function [Y1,IAEr,IAEv,TV] = CL_step_pid(A,B,C,hc,theta,t,dt,n)
function [PM, GM, Ms, IAEr, IAEv, TV]= CL_step_pid(A,B,C,hc,theta,t,dt,n)
     Ti=hc.Ti; Kp=hc.Kp; Td=hc.Td; % PID controller
    s=tf('s');

    [b,a] = ss2tf(A,B,C,0);
    hp = tf(b,a)*exp(-theta*s);
    L = series(hc,hp);
    S = 1/(1+L); % Sensitivity function
    F=logspace(-10,10,1e4); % Logaritmic scale
```

```
    [GM,PM,Wcg,Wcp] = margin(L);
    Ms = max(abs(freqresp(S,F))); % Ms, Robustness

    r=1; % The reference signal
    N=length(t);
    nt=theta/dt; nt=round(nt); Delay=zeros(nt,1);
    hc=dt*Kp/Ti; % Controller param. to save computing time
    x=zeros(n,1);% Initial values for the states
    z=0; y_old=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
        z=z+hc*e; % Updating the controller state
        Y1(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        %u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEr = sum(abs(E)*dt);
    TV = abs(diff(U));
    TV = sum(TV);

    Delay=zeros(nt,1);
    x=zeros(n,1);% Initial values for the states
    z=0; y_old=0;
    r=0;
    for k=1:N
        yp=C*x; % The measurement output
        y=Delay(nt); % the delayed process output Y=y(i-nt)
        for id=nt:-1:2
            Delay(id)=Delay(id-1);
        end
        Delay(1)=yp;
        e=r-y; % The controller input
        u=z+Kp*e - Kp*Td*((y-y_old)/dt); y_old=y; % The controller output PID controller
        z=z+hc*e; % Updating the controller state
        Y2(k,1)=y; U(k,1)=u; E(k,1)=e;% Storing variables
        u = u+1; %input disturbance
        x=x + dt*(A*x+B*u); % Puting the control to the process
    end
    IAEv = sum(abs(E)*dt);
end
```

5 Other

```
function [Wry,Wvy] = OPTweights(hp,t,dt)
%  Find IAE-optimal PI-controllers to obtain reference IAE values - Preben Solvang Master thesis 2
%  Inputs:
%      hp: Plant model
%       t: time vector
%      dt: time step size

% Initial controller found with pidtune
opt = pidtuneOptions('PhaseMargin', 40); % lower PM than default to get closer to the optimal setti
```

```
c0=pidstd(1,1);
hc = pidtune(hp,c0,opt);
x = [hc.Kp, hc.Ti];

opt=optimset('display','off','TolX',1e-7,'TolFun',1e-7,'LargeScale','off','MaxIter',700);
cost = @(x) costfuncPIry(x, hp, t, dt); % x = [Kp Ti]
nonlcon = @(x) CalcCPI(x,hp);
%[x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001],[inf inf],[],opt);
[x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001],[inf inf],nonlcon,opt);
Wry=fval;
cost = @(x) costfuncPIvy(x, hp, t, dt); % x = [Kp Ti]
[x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0.0001],[inf inf],nonlcon,opt);
Wvy=fval;
end

function J = costfuncPIry(x,hp,t,dt)
    hc = pidstd(x(1),x(2));
    RY = hc*hp/(hc*hp+1); % SP tracking
    eRY = 1-step(RY,t); %SP tracking error
    IAEry = (sum(abs(eRY)*dt));
    % performance calculation
    J = IAEry;
end

function J = costfuncPIvy(x,hp,t,dt)
    hc = pidstd(x(1),x(2));
    VY = hp/(hc*hp+1); % output disturbance
    eVY = 0-step(VY,t); %SP tracking error
    IAEvy = (sum(abs(eVY)*dt));
    % performance calculation
    J = IAEvy;
end

function [c,ceq] = CalcCPI(x,hp)
    c=[];
    MsPre = 1.59;
    hc=pidstd(x(1),x(2));
    F=logspace(-10,10,1e4);
    Ms = max(abs(freqresp(1/(hp*hc+1),F)));
    cs=Ms-MsPre;
    ceq=cs;
end

function [hc,Kp,Ti,Td]=PO_pidtune(hp,hcType,IAEryW,IAEvyW,t,dt,MsP)
% PO optimal PI tuning tool - Preben Solvang Master thesis 2019
%  Inputs:
%      hp: Plant model
% hcTtype: controller type 2 = PI, 3 = PID

% Initial controller found with pidtune
opt = pidtuneOptions('PhaseMargin', 40); % lower PM than default to get closer to the optimal sett
%Initial PI controller
    c0=pidstd(1,1);
    hc = pidtune(hp,c0,opt);
    x = [hc.Kp, hc.Ti];

opt=optimset('display','off','TolX',1e-6,'TolFun',1e-6,'LargeScale','off','MaxIter',700);
% Cost function
% PI controller
    cost = @(x) costfuncPI(x, hp, t, dt, IAEryW, IAEvyW); % x = [Kp Ti Td]
```

```
    nonlcon = @(x) CalcCPI(x,hp,MsP);
    [x,fval,flag]=fmincon(cost,x,[],[],[],[],[-inf 0],[inf inf],nonlcon,opt);
    hc = pidstd(x(1),x(2));
    Kp = hc.Kp;
    Ti = hc.Ti;
    Td = 0;
end

function J=costfuncPI(x,hp,t,dt,IAEryW,IAEvyW)
    hc = pidstd(x(1),x(2));
    RY = hc*hp/(hc*hp+1); % SP tracking
    VY = hp/(hc*hp+1); % input disturbance
    eRY = 1-step(RY,t); %SP tracking error
    eVY = 0-step(VY,t); %input disturbance error
    IAEry = (sum(abs(eRY)*dt));
    IAEvy = (sum(abs(eVY)*dt));
    %IAEvyW=1;IAEvuW=1;
    % performance calculation
    J = 0.5*((IAEvy/IAEvyW) + (IAEry/IAEryW));
end

function [c,ceq] = CalcCPI(x,hp,MsP)
    c=[];
    MstPre = MsP;%1.59;
    MksPre = 50;
    hc=pidstd(x(1),x(2));
    F=logspace(-10,10,1e4);
    Ms = max(abs(freqresp(1/(hp*hc+1),F)));
    Mt = max(abs(freqresp(hp*hc/(hp*hc+1),F)));
    Mks=max(abs(freqresp(hc/(hp*hc+1),F)));
    ct=Mks-MksPre;
    Mst = Ms;%max(Ms, Mt);
    cs=Mst-MstPre;
    ceq=cs;
end
```

Appendix E

Survey results

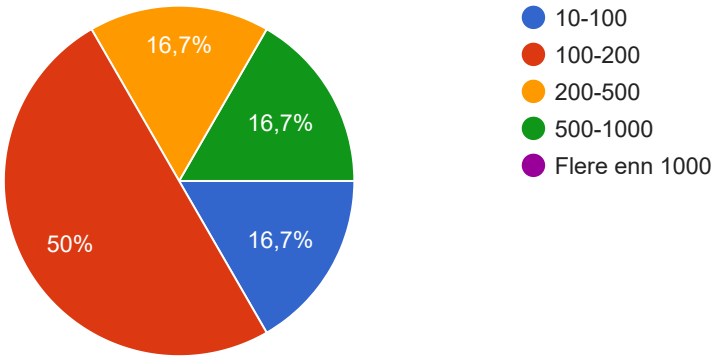
All the answers from the survey is presented here.

Spørreskjema Reguleringssløyfer og PID regulering

6 svar

Kan du anslå hvor mange reguleringssløyfer med en PI eller PID regulator dere har?

6 svar

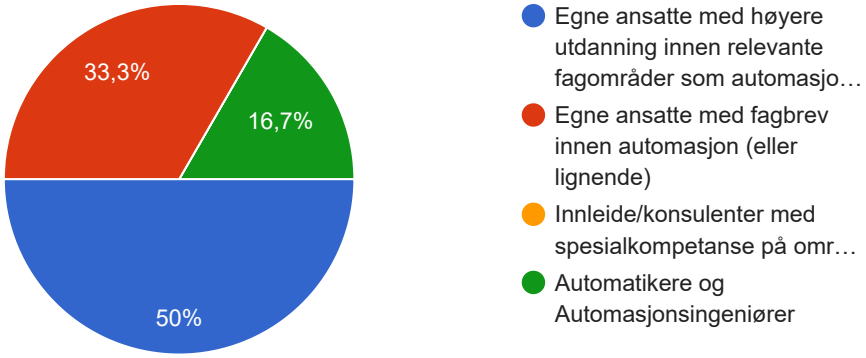


Har dere en overordnet strategi for hvordan disse reguleringssløyfene tunes? (Identifisering av spesielt viktige sløyfer, kursing/opplæring av personell, skriftlige prosedyrer osv)

6 svar

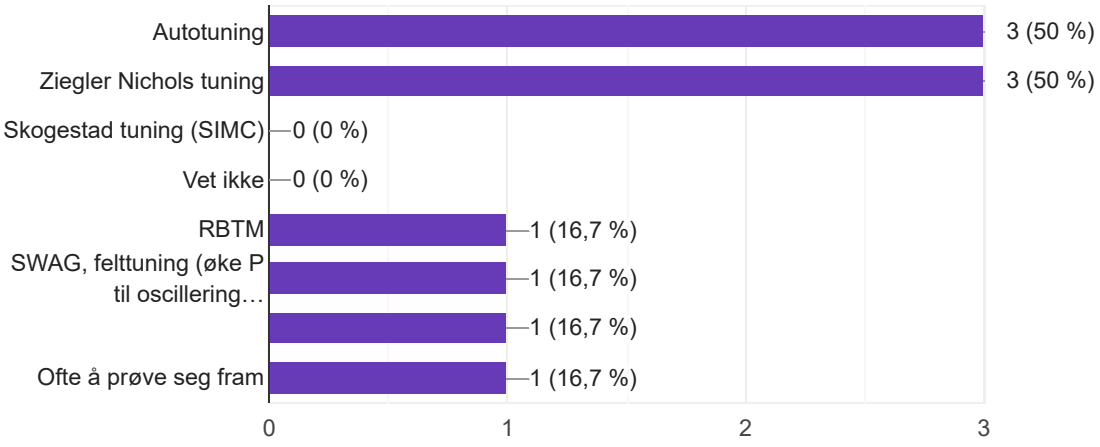
Hvem tar seg av tuning av reguleringssløyfer?

6 svar



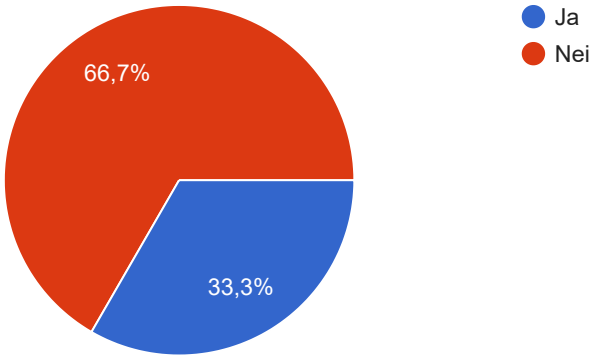
Hvilke metoder blir brukt for tuning av reguleringssløyfer?

6 svar



Bruker dere matematiske modeller i arbeidet med reguleringssløyfer?

6 svar



Andre kommentarer?

2 svar

VI har prosesser som normalt ikke

Generelt har vi veiledende parametersett for nye regulatorer avhengig av type, TIC, PIC ol. Normalt tas aksjon kun hvis regulatorer ikke oppfører seg tilfredsstillende og da benyttes prøve og feile metoden hvor en skotter til Ziegler Nichols, Skogestad ol. Har kun forsøksmessig benyttet autotuning. For noen prosessavsnitt med spesielle behov er det gjort offline modellforsøk. For 2 PVC-autoklaver har et firma Cybernetica levert en installasjon hvor MPC med Kalman Filter og Moving Horizon Estimator er i bruk.
Ca antall PID regulatorer pr fabrikk:
Klor 360
VCM 340
PVC 174