SPECIALIZATION PROJECT 2012

TKP 4550 - Process Systems Engineering, Specialization Project

PROJECT TITLE: Validation of the SIMC PID Tuning Rules

By Martin S. Foss

Supervisor for the project: Sigurd Skogestad and Chriss Grimholt
Date: 07.12.2012
Validation of the SIMC PID Tuning Rules

Written by: Martin S. Foss
martf@stud.ntnu.no

Supervisor:
Sigurd Skogestad
skoge@chemeng.ntnu.no

Co-supervisor:
Chriss Grimholt
chriss.grimholt@ntnu.no

December 7, 2012
Abstract

The aim of this report has been to validate the SIMC PID tuning rules for second order plus time delay processes. The PID controller is the most used controller in the process industry, and the presence of simple tuning rules that can be used to tune robust and high performing controllers would be a great advantage. All calculations and simulations has been accomplished with the use of MATLAB and SIMULINK.

The trade-off between robustness and performance for the SIMC tuning rules has been investigated with the Pareto-optimal curves as a foundation. The SIMC tuning rules have been found to perform close to optimal for $M_s$ values below two. Resulting in controllers that are less aggressive compared to the Pareto-optimal, i.e. having better setpoint performance and slightly reduced disturbance rejection.

The recommended choice of tuning parameter $\tau_c = \theta$ has been found to be too high for processes with $\frac{\tau_2}{\tau_1} < 0.5$. For such processes $\tau_c$ should be chosen smaller, e.g. $\tau_c = 0.5\theta$.

This report has tested nine different cases. The major challenge has been to get the numerical solver to converge and find the solution to the minimization problem. For future work, the numerical solver should be made more robust, or replaced, so that a wider selection of processes can be tested.
Preface

This report is the result of the Specialization Project, TKP4550, at the department of Chemical Engineering, within the group Process Systems Engineering, at NTNU, fall 2012.

The aim of the project has been to validate the SIMC PID tuning rules for a set of second order plus time delay process models. To reach the goal and achieve the results presented in this report MATLAB has been used for calculations together with simulations in SIMULINK.

The completion of this project would not have been possible without the help, guidance and support from some important people. First of all, I would like to thank my co-supervisor Chriss Grimholt for his support and guidance throughout the work with this project. Whenever I had questions he would answer to the best of his ability and get me back on the right track.

Secondly, I would like to give my thanks to my supervisor, professor Sigurd Skogestad. Though he has a busy schedule, he has directed me in the right direction when questions arose.

I would also like to give great thanks my friends, Peter Johan Bergh Lindersen and Ivar Magnus Jevne, and my partner, Pia Odden, for their moral support, inspiring discussions and proof-reading of this report. I know how hectic their schedule have been, and I am truly grateful for the help I have received.

Trondheim, December 7, 2012

Martin S. Foss
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Preface</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Theory - Background</td>
<td>2</td>
</tr>
<tr>
<td>2.1 The PID controller and SIMC tuning rules</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Pareto optimization</td>
<td>5</td>
</tr>
<tr>
<td>2.2.1 Performance</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Robustness</td>
<td>7</td>
</tr>
<tr>
<td>2.3 The objective function</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Cases</td>
<td>8</td>
</tr>
<tr>
<td>2.5 Calculations and Simulations</td>
<td>9</td>
</tr>
<tr>
<td>3 Results and Discussion</td>
<td>10</td>
</tr>
<tr>
<td>3.1 Pareto-optimal PID and PI weights</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Pareto-optimal vs. SIMC tunings</td>
<td>12</td>
</tr>
<tr>
<td>3.2.1 Time delay dominated processes</td>
<td>12</td>
</tr>
<tr>
<td>3.2.2 Lag dominated processes</td>
<td>14</td>
</tr>
<tr>
<td>3.2.3 Summary of Pareto-optimal vs. SIMC tunings</td>
<td>18</td>
</tr>
<tr>
<td>3.3 Step responses</td>
<td>20</td>
</tr>
<tr>
<td>3.3.1 Time delay dominated processes</td>
<td>22</td>
</tr>
<tr>
<td>3.3.2 Lag dominated processes</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3 Summary step responses</td>
<td>30</td>
</tr>
<tr>
<td>3.4 Challenges and future work</td>
<td>30</td>
</tr>
<tr>
<td>4 Conclusions</td>
<td>31</td>
</tr>
<tr>
<td>References</td>
<td>32</td>
</tr>
<tr>
<td>A MATLAB Scripts</td>
<td>A-1</td>
</tr>
<tr>
<td>A.1 Obtain pareto optimal tunings</td>
<td>A-1</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Optimal PID tunings, main file</td>
</tr>
<tr>
<td>A.1.2</td>
<td>Optimal PI tunings, main file</td>
</tr>
<tr>
<td>A.1.3</td>
<td>Cost function</td>
</tr>
<tr>
<td>A.1.4</td>
<td>Constraints</td>
</tr>
<tr>
<td>A.2</td>
<td>Obtain PO vs. SIMC tuning plots</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Main file</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Cost function</td>
</tr>
<tr>
<td>A.2.3</td>
<td>SIMC PI-tunings</td>
</tr>
<tr>
<td>A.2.4</td>
<td>SIMC PID-tunings</td>
</tr>
<tr>
<td>A.3</td>
<td>Obtain step responses</td>
</tr>
<tr>
<td>A.3.1</td>
<td>Main file</td>
</tr>
<tr>
<td>A.4</td>
<td>Obtain parallel tuning parameters</td>
</tr>
<tr>
<td>A.4.1</td>
<td>Main file</td>
</tr>
<tr>
<td>A.5</td>
<td>Shared files</td>
</tr>
<tr>
<td>A.5.1</td>
<td>Cases - second order models</td>
</tr>
<tr>
<td>A.5.2</td>
<td>Cases - first order models</td>
</tr>
<tr>
<td>A.5.3</td>
<td>Controller</td>
</tr>
<tr>
<td>A.5.4</td>
<td>Integral absolute error</td>
</tr>
<tr>
<td>A.5.5</td>
<td>Ms</td>
</tr>
</tbody>
</table>

B  SIMULINK Model | B-1
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Block diagram of general feedback control system.</td>
</tr>
<tr>
<td>2.2</td>
<td>Block diagram of feedback control system used in the project.</td>
</tr>
<tr>
<td>2.3</td>
<td>Typical Pareto-optimal curve of two conflicting objective functions.</td>
</tr>
<tr>
<td>3.1</td>
<td>PO vs. SIMC, time delay dominated processes.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 1, $\frac{\tau_2}{\tau_1} = 0.5, \frac{\theta}{\theta} = 0.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 2, $\frac{\tau_2}{\tau_1} = 0.8, \frac{\theta}{\theta} = 0.8$.</td>
</tr>
<tr>
<td>(c)</td>
<td>Case 3, $\frac{\tau_2}{\tau_1} = 0.3, \frac{\theta}{\theta} = 0.3$.</td>
</tr>
<tr>
<td>3.2</td>
<td>PO vs. SIMC, with $\frac{\tau_2}{\tau_1} = 0.5$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 4, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 7, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>3.3</td>
<td>PO vs. SIMC, with $\frac{\tau_2}{\tau_1} = 0.8$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 5, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 8, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>3.4</td>
<td>PO vs. SIMC, with $\frac{\tau_2}{\tau_1} = 0.3$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 6, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 9, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>3.5</td>
<td>Step responses, time delay dominated processes.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 1, $\frac{\tau_2}{\tau_1} = 0.5, \frac{\theta}{\theta} = 0.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 2, $\frac{\tau_2}{\tau_1} = 0.8, \frac{\theta}{\theta} = 0.8$.</td>
</tr>
<tr>
<td>(c)</td>
<td>Case 3, $\frac{\tau_2}{\tau_1} = 0.3, \frac{\theta}{\theta} = 0.3$.</td>
</tr>
<tr>
<td>3.6</td>
<td>Step responses, with $\frac{\tau_2}{\tau_1} = 0.5$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 4, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 7, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>3.7</td>
<td>Step responses, with $\frac{\tau_2}{\tau_1} = 0.8$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 5, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 8, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>3.8</td>
<td>Step responses, with $\frac{\tau_2}{\tau_1} = 0.3$.</td>
</tr>
<tr>
<td>(a)</td>
<td>Case 6, $\frac{\tau_2}{\tau_1} = 1.5$.</td>
</tr>
<tr>
<td>(b)</td>
<td>Case 9, $\frac{\tau_2}{\tau_1} = 2.0$.</td>
</tr>
<tr>
<td>B.1</td>
<td>SIMULINK model</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Pareto-optimal IAE-weights. ........................................... 11
3.2 Tuning variables, parallel form. ...................................... 21
# List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>Disturbance, input</td>
</tr>
<tr>
<td>$d_{out}$</td>
<td>Disturbance, output</td>
</tr>
<tr>
<td>$e$</td>
<td>Error, controller input</td>
</tr>
<tr>
<td>$g_c$</td>
<td>Controller model (transfer function)</td>
</tr>
<tr>
<td>$g_p$</td>
<td>Process model (transfer function)</td>
</tr>
<tr>
<td>$IAE$</td>
<td>Integral absolute error</td>
</tr>
<tr>
<td>$k$</td>
<td>Process gain</td>
</tr>
<tr>
<td>$K_c$</td>
<td>Controller gain</td>
</tr>
<tr>
<td>$J$</td>
<td>Cost function</td>
</tr>
<tr>
<td>$M_s$</td>
<td>Peak in sensitivity function</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Frequency</td>
</tr>
<tr>
<td>$s$</td>
<td>Laplace-variable</td>
</tr>
<tr>
<td>$t$</td>
<td>Time</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>$1^{st}$ time constant (largest)</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>$2^{nd}$ time constant (smallest)</td>
</tr>
<tr>
<td>$\tau_c$</td>
<td>Closed loop time constant</td>
</tr>
<tr>
<td>$\tau_D$</td>
<td>Derivative time constant</td>
</tr>
<tr>
<td>$\tau_I$</td>
<td>Integral time constant</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Time delay</td>
</tr>
<tr>
<td>$u$</td>
<td>Process input</td>
</tr>
</tbody>
</table>
1 Introduction

One of the most used controllers in the industry is the PID controller [1]. Despite the frequent use, this type of controller is often poorly tuned. Though there are only three tuning parameters in the PID controller the optimal tuning, i.e. optimal trade-off between performance and robustness, is difficult to obtain. The optimal performance can in itself also be difficult to define. The requirements of robustness and performance may need good engineering insight to be determined. It is not possible to get the best of both, so a settlement in the middle ground should be chosen.

The aim of this report is to validate the SIMC tuning rules presented by Skogestad [2]. These rules are developed to be easy to remember and to result in good closed-loop behavior. Earlier investigations that has been preformed to investigate the SIMC tuning rules with respect to PI control [3], has shown that these rules result in good trade-off between robustness and performance.

The tuning rules is tested to a set of second-order-plus-time-delay (SOPTD) processes. The performance of the PID controller tunings will be compared with the "Pareto-optimal" (PO) tunings, which can be referred to as "the best one can get". The PID tunings will also be compared with PI tunings. The latter to see if there is any need of implementing a PID controller.
2 Theory - Background

The SIMC rules presented by Skogestad [2], uses an open-loop process model to derive the controller settings for PID and PI controllers. Dependent on the desired controller (PID or PI) the model has to be reduced to a second-order-plus-time-delay (SOPTD) model or a first-order-plus-time-delay (FOPTD), respectively. The two types are presented in Equation (2.1) and (2.2), respectively. In this project the aim has been to validate the PID tuning rules and thus, second order models has been used. However, a PI controller has been used for comparison. Thus, the SOPTD models have been reduced to FOPTD models using the "half rule" [1,2].

\[
g_p = \frac{k_p}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot e^{-\theta s} \tag{2.1}
\]

\[
g_p = \frac{k_p}{(\tau_1 s + 1)} \cdot e^{-\theta s} \tag{2.2}
\]

Where \(g_p\) is the process transfer function, \(k_p\) is the process gain, \(\tau_1\) and \(\tau_2\) are the time constants, and \(\theta\) is the time delay.

The block diagram depicted in Figure 2.1 is the conventional feedback loop, where \(y_s\) is the setpoint, \(e\) is the controller error, \(u\) is the manipulated variable (process input), \(d\) is the input disturbance, \(d_{out}\) is the output disturbance and \(y\) is the process output. \(g_c\) and \(g_p\) are the controller and process transfer functions, respectively.

![Block diagram of general feedback control system (with input and output disturbances).](image-url)
The simulations performed in this project have tested the controller settings for setpoint changes and input disturbances. Output disturbances have not been investigated as they have the same effect as a change (or disturbance) in the setpoint, and thus can be treated as a special case of setpoint change. Hence, the block diagram in Figure 2.1 is slightly modified, and the block diagram used in this project can be presented as:

\[ y_s + g_p + 1 \downarrow d \quad y \]  

**Figure 2.2:** Block diagram of feedback control system used in the project.

### 2.1 The PID controller and SIMC tuning rules

The PID controller are often presented in its parallel form as given in Equation (2.3).

\[ g_{PID} = P + \frac{I}{s} + Ds \]  

Where P denotes the proportional part, I the integral part and D the derivative part. The three parts of the controller have different effects on the manipulated variable. The proportional part change the manipulated variable directly proportional to the error. The integral part change the manipulated variable proportional to the integrated error and the derivative part change the manipulated variable proportional to the derivative of the controlled variable. All in all the controller will try to minimize the error, \( e \), in Figures 2.1 and 2.2 by adjusting the process input (\( u \)).

The three parts of the PID controller, discussed above, has their individual tuning parameters, i.e. \( K_c \), \( \tau_i \) and \( \tau_D \). To find these parameters the SIMC tuning rules uses two main steps, i.e.:
2.1 The PID controller and SIMC tuning rules

1. Obtain a FOPTD or a SOPTD model.
   - Perform open or closed loop experiments.
   - If model of higher order is known, reduce with use of the "half rule".

2. Get controller settings from the tuning rules, presented below.

The SIMC tuning rules are given for the ideal, series form, PID controller, as defined in Equation (2.4).

\[
g_c(s) = K_c \cdot \left( \frac{\tau I s + 1}{\tau l s} \right) \cdot (\tau D s + 1)
\]

Where \(g_c\) is the controller transfer function, \(K_c\) is the controller gain, \(\tau I\) is the integral time and \(\tau D\) is the derivative time. The tuning rules for a PID controller can be found from a SOPTD process, see Equation (2.1), as follows:

\[
K_c = \frac{1}{k} \frac{\tau l}{\tau c + \theta}
\]

\[
\tau l = \min\{\tau l, 4(\tau c + \theta)\}
\]

\[
\tau D = \tau 2
\]

As seen from Equation (2.5) - (2.7) the SIMC tuning rules has only one independent variable, i.e. \(\tau c\). The recommended value for this parameter is \(\tau c = \theta\), which should yield tight control with good trade-off between robustness and performance.

If a PI controller is to be tuned, the \(K_c\) and \(\tau l\) parameter are defined in the same manner. \(\tau D = 0\), as the PI controller do not have this tuning parameter.

As the PID controller often is presented in its parallell form, Equation (2.8), recalculation of the tuning parameters are required. The corresponding parallell tuning parameters can be calculated by the translation formulas presented in Equations (2.9), (2.10) and (2.11).
2.2 Pareto optimization

The search after good controller tunings can be difficult without a systematic approach. A set of tuning parameters can never yield perfect performance and good robustness at the same time. A controller with good performance is normally not very robust, and vice versa. There will also be a trade-off between the response to a change in setpoint and disturbances.

To be able to find the optimal compromise between performance and robustness Pareto-optimal (PO) curves can be helpful. A PO-curve is represented in Figure 2.3. The figure depicts two conflicting objective functions plotted against each other. For each point, the optimal value of the two objective functions are plotted. The trade-off is clearly depicted. As objective function 1 is low, objective function 2 is high, and vice versa. The optimal point is somewhere in the middle (bold, red line), but exactly where is up to the individual engineer and the respective case.

\[ g'_c(s) = K'_c \left( 1 + \frac{1}{\tau'_I s} + \frac{\tau'_D}{\tau'_I} s \right) \]  \hspace{1cm} (2.8)

\[ K'_c = K_c \left( 1 + \frac{\tau_D}{\tau_I} \right) \]  \hspace{1cm} (2.9)

\[ \tau'_I = \tau_I \left( 1 + \frac{\tau_D}{\tau_I} \right) \]  \hspace{1cm} (2.10)

\[ \tau'_D = \frac{\tau_D}{1 + \frac{\tau_D}{\tau_I}} \]  \hspace{1cm} (2.11)

Where \( g'_c \), \( K'_c \), \( \tau'_I \) and \( \tau'_D \) are the parallel controller transfer function, controller gain, integral time and derivative time, respectively.
2.2 Pareto optimization

Figure 2.3: Typical Pareto-optimal curve of two conflicting objective functions.

In this project the two objective functions are, as mentioned, performance and robustness. For simplicity, the robustness has been made the independent variable, whilst the robustness is the dependent variable. That is:

\[
\text{performance} = f(\text{robustness})
\]  

(2.12)

In the following subsections the basis for the performance and robustness functions are highlighted.

2.2.1 Performance

There are several possible methods to evaluate the performance. In this project the integral absolute error (IAE) has been used. The IAE is defined in Equation (2.13) and is a good indication of the speed and precision of the controller.
2.2 Pareto optimization

2 THEORY - BACKGROUND

IAE = \int_{0}^{\infty} |e| dt = \int_{0}^{\infty} |y(t) - y_s(t)| dt \quad (2.13)

2.2.2 Robustness

The robustness is measured by the peak in the sensitivity function, \( M_s \). The sensitivity function, \( S(j\omega) \), is defined as the closed-loop transfer function between the output disturbance, \( d_{out} \), and the output, \( y \), see Figure 2.1 [4]. In addition the \( M_s^{-1} \) is the closest distance to the critical point -1 in the Nyquist plot. For stability, the best thing is to be as far away from this point as possible, i.e a \( M_s \) value of 1 is desired.

\[
M_s = \max_{\omega} |S(j\omega)| = \max_{\omega} \left| \frac{1}{1 + g_c(j\omega)} \right|
\quad (2.14)

For any given \( M_s \) value the following applies [4]:

\[
GM \geq \frac{M_s}{M_s - 1} \quad \text{and} \quad PM \geq \frac{1}{M_s}
\quad (2.15)
\]

As the \( M_s \)-value decreases the robustness of the controller will increase. The best thing, both for stability and performance, would be to have a \( M_s \)-value close to one [4]. The \( M_s \)-value should not exceed \( 2 \)\(^1\) and the closer to 1 the more robust the controller will get. However, the cost of decreasing the \( M_s \)-value will often be too high when approaching low values, so a value between 1.6 - 1.7 is typically "good" [3].

---

\(^1\) A \( M_s \) of two yields \( GM \geq 2 \) and \( PM \geq 29.0^\circ \), which represents the recommended upper bounds [4].
2.3 The objective function

The objective function, which should be minimized, used in this project is defined by Equation (2.16). The function has been calculated in the domain given in Equation (2.17).

\[ J(c) = 0.5 \left[ \frac{IAE_{ys}(c)}{IAE_{ys}^o} + \frac{IAE_d(c)}{IAE_d^o} \right] \]  
\[ M_s = \{1.25, 1.30, \ldots, 3.00\} \]

Where \( IAE_{ys}^o \) and \( IAE_d^o \) denotes the error when there is performed an input and a disturbance step with a Pareto-optimal tuning, respectively. In this way the performance is weighted against a constant reference. As the performance is a function of the robustness, a \( M_s \)-value of 1.59 is used when the PO-curves are constructed. The resulting weights are presented in Table 3.1.

2.4 Cases

In this project nine cases have been tested. These are given in Equation (2.18) – (2.23). The first three cases, case 1 – case 3, are time delay dominated processes, whilst case 4 – case 9 are lag dominated.

Case 1:

\[ g_p = \frac{1}{(s+1)(0.5s+1)} \cdot e^{-s} \]  
\[ (2.18) \]

Case 2:

\[ g_p = \frac{1}{(s+1)(0.8s+1)} \cdot e^{-s} \]  
\[ (2.19) \]

Case 3:

\[ g_p = \frac{1}{(s+1)(0.3s+1)} \cdot e^{-s} \]  
\[ (2.20) \]
2.5 Calculations and Simulations

All calculations and simulations in this project are performed by use of MATLAB and SIMULINK. The MATLAB scripts and SIMULINK block diagram are included in Appendix A and B respectively.

Case 4:
\[ g_p = \frac{1}{(s+1)(0.5s+1)} \cdot e^{-\frac{1}{7}s} \]  \hspace{1cm} (2.21)

Case 5:
\[ g_p = \frac{1}{(s+1)(0.8s+1)} \cdot e^{-\frac{8}{15}s} \]  \hspace{1cm} (2.22)

Case 6:
\[ g_p = \frac{1}{(s+1)(0.3s+1)} \cdot e^{-\frac{2}{15}s} \]  \hspace{1cm} (2.23)

Case 7:
\[ g_p = \frac{1}{(s+1)(0.5s+1)} \cdot e^{-0.25s} \]  \hspace{1cm} (2.24)

Case 8:
\[ g_p = \frac{1}{(s+1)(0.8s+1)} \cdot e^{-0.4s} \]  \hspace{1cm} (2.25)

Case 9:
\[ g_p = \frac{1}{(s+1)(0.3s+1)} \cdot e^{-0.1s} \]  \hspace{1cm} (2.26)
3 Results and Discussion

In the following sections all the obtained results are presented along with a discussion.

3.1 Pareto-optimal PID and PI weights

To be able to assess the performance of the controllers for the different cases the cost function in Equation (2.16) had to be solved. In order to achieve this, the $IAE_{ys}^c$ and $IAE_d^c$ had to be calculated. This was performed by calculating the error with only a step change in the setpoint and disturbance, respectively. These Pareto-optimal parameters was found for both the PID and a PI controllers for a $M_s$-value of 1.59. All the the weights are presented in Table 3.1.

As can be seen from Table 3.1 there is a consistently better performance by the PID controller, both for setpoint changes and disturbances, for all cases. This observation fits well with theory, as the PID controller has one extra tuning parameter compared with the PI controller, and should therefore perform better. The PO-controllers are also observed to perform better to disturbances than to a step in the setpoint.
### Table 3.1: Comparison between the IAE-weights with a Pareto-optimal PID and PI controller for all the nine cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>$IAE_{ys}^0$</th>
<th>$IAE_{y}^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PID</td>
<td>PI</td>
</tr>
<tr>
<td>Case 1</td>
<td>1.92</td>
<td>2.81</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{-s}$</td>
<td></td>
</tr>
<tr>
<td>Case 2</td>
<td>1.98</td>
<td>3.07</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{-s}$</td>
<td></td>
</tr>
<tr>
<td>Case 3</td>
<td>1.83</td>
<td>2.57</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{-s}$</td>
<td></td>
</tr>
<tr>
<td>Case 4</td>
<td>0.70</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{-\frac{1}{2}s}$</td>
<td></td>
</tr>
<tr>
<td>Case 5</td>
<td>1.11</td>
<td>2.12</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{-\frac{8}{15}s}$</td>
<td></td>
</tr>
<tr>
<td>Case 6</td>
<td>0.43</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{-0.2s}$</td>
<td></td>
</tr>
<tr>
<td>Case 7</td>
<td>0.53</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{-0.25s}$</td>
<td></td>
</tr>
<tr>
<td>Case 8</td>
<td>0.84</td>
<td>1.83</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{-0.4s}$</td>
<td></td>
</tr>
<tr>
<td>Case 9</td>
<td>0.32</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{-\frac{3}{20}s}$</td>
<td></td>
</tr>
</tbody>
</table>
3.2 Pareto-optimal vs. SIMC tunings

To assess the performance of the SIMC PID tuning rules the SIMC PID curve has been plotted in the same figure as the Pareto-optimal PID curve. The Pareto-optimal PI and SIMC PI curves has also been included in the same figure. The latter to investigate if there is any need and readily "profit" by implementing a PID controller, or if a PI controller will suffice.

3.2.1 Time delay dominated processes

The SIMC tuning rules state that as long as the time delay is greater than the second time constant, i.e. \( \theta > \tau_2 \), there is no need for implementing a PID controller. This statement has been tested for three different cases, case 1 - case 3, which are depicted in Figure 3.1.

As shown in Figures 3.1a, 3.1b and 3.1c the Pareto-optimal PID (blue) curve yields, as expected, the best performance. The Pareto-optimal PI and the SIMC PI curves (green and cyan, respectively) coincide for \( M_s \) values smaller than approximately 1.6. The SIMC PID shows better performance, compared with the PI controller, for all the three cases. In the \( M_s \) domain \([1.3 - 2.0]\), the SIMC PI controllers underperform the SIMC PID controller with approximately 35, 50 and 25 % for the three cases, respectively. Hence, the advantage of implementing a PID controller is quite small. However, the decision to install a PID or not, is not unambiguous and it has to be investigated in more detail for each individual case. As the ratio \( \frac{\tau_2}{\tau_1} \) increase, the advantage of a PID controller will also increase.

The figures show that the SIMC tuning rules give close to optimal controllers for the three cases. The recommended choice of the tuning parameter \( \tau_c \), i.e. \( \tau_c = \theta \), yields a \( M_s \) value of approximately 1.6 (PID) and 1.8 (PI) for all the three cases, and thus results in a good trade-off between performance and robustness.
3.2 Pareto-optimal vs. SIMC tunings

RESULTS AND DISCUSSION

Figure 3.1: Pareto optimal (PO) vs. SIMC tuning curves for time delay dominated second order plus time delay processes on the form \( g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot \exp(-\theta s) \).
3.2 Pareto-optimal vs. SIMC tunings

3.2.2 Lag dominated processes

The remaining cases has been plotted in the same manner as the time delay dominated processes, i.e. by comparing the Pareto-optimal PID and PI tuning curves with the SIMC PID and PI tuning curves. The results are presented in Figures 3.2 – 3.4.

Figure 3.2a and 3.2b depicts case 4 and case 7, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.5, whilst the ratio $\frac{\tau_2}{\theta}$ is 1.5 and 2.0, respectively. For both processes the SIMC tuning rules produces controllers with almost no non-optimality loss in the preferred $M_s$ domain, i.e. $M_s < 2.0$. As $M_s$ decreases and approach one, the cost function increases dramatically.

In the $M_s$ domain $[1.3 – 2.0]$ the SIMC PI underperform the SIMC PID on average with approximately 105 % for case 4 and 125 % for case 7.

The figures show that increasing the ratio $\frac{\tau_2}{\theta}$ will result in a shift towards lower $M_s$ values for given $\tau_c$’s. The recommended choice of $\tau_c = \theta$ is shown to result in a $M_s$ value of 1.25 for case 4 and 1.20 for case 7. This will give a robust controller, but because of the trade-off between performance and robustness, the controller will loose performance. The steep gradients in these points indicate that by increasing the $M_s$ value a small amount, will give a large advantage/increase in performance. A $\tau_c$ equal to $0.5 \cdot \theta$, diamond shaped point in the figures, would be a better alternative for both processes. This tuning parameter will give a $M_s$-value of 1.44 and 1.35 for the two cases, respectively.
3.2 Pareto-optimal vs. SIMC tunings

RESULTS AND DISCUSSION

Figure 3.2: Pareto optimal (PO) vs. SIMC tuning curves for second order plus time delay processes on the form \( g_p = \frac{1}{(\tau_1 s+1)(\tau_2 s+1)} \cdot \exp(-\theta s) \), with \( \frac{\dot{\omega}}{\bar{\phi}} = 0.5 \).
3.2 Pareto-optimal vs. SIMC tunings

Figure 3.3a and 3.3b depicts case 5 and case 8, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.8, whilst the ratio $\frac{\tau_2}{\theta}$ is 1.5 and 2.0, respectively. For both processes the SIMC tuning rules produces controllers with almost no non-optimality loss in the preferred $M_s$ domain, i.e. $M_s < 2.0$. As $M_s$ decreases and approach one, the cost function increases dramatically.

In the $M_s$ domain $[1.3 - 2.0]$ the SIMC PI underperform the SIMC PID on average with approximately 95% for case 5 and 120% for case 8.

The figures show that increasing the ratio $\frac{\tau_2}{\theta}$ will result in a shift towards lower $M_s$ values for given $\tau_c$’s. The recommended choice of $\tau_c = \theta$ is shown to result in a $M_s$ value of 1.37 for case 5 and 1.29 for case 8. These tunings give a good trade-off between performance and robustness. If the tuning parameter, $\tau_c$, was selected to $0.5 \cdot \theta$, diamond shaped point in the figures, the resulting $M_s$ values would be 1.62 and 1.50 for the two cases, respectively. These values will also result in a decent trade-off.
3.2 Pareto-optimal vs. SIMC tunings

RESULTS AND DISCUSSION

Figure 3.3: Pareto optimal (PO) vs. SIMC tuning curves for second order plus time delay processes on the form \( g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot \exp(-\theta s) \), with \( \frac{\tau_2}{\tau_1} = 0.8 \).
3.2 Pareto-optimal vs. SIMC tunings

Figure 3.4a and 3.4b depicts case 6 and case 9, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.3, whilst the ratio $\frac{\tau_2}{\theta}$ is 1.5 and 2.0, respectively. For both processes the SIMC tuning rules produces controllers with almost no non-optimality loss in the preferred $M_s$ domain, i.e. $M_s < 2.0$. As $M_s$ decreases and approach one, the cost function increases dramatically.

In the $M_s$ domain $[1.3 - 2.0]$ the SIMC PI underperform the SIMC PID on average with approximately 100 % for case 6 and 130 % for case 9.

The figures show that increasing the ratio $\frac{\tau_2}{\theta}$ will result in a shift towards lower $M_s$ values for given $\tau_c$’s. The recommended choice of $\tau_c = \theta$ is shown to result in a $M_s$ value of 1.16 for case 6 and 1.12 for case 8. These tunings will give a robust controller, but the robustness/performance trade-off yields quite poor performance. The steep gradients in these points indicate that a slight increasing in the $M_s$-value lead to a large advantage/increase in performance. A $\tau_c$ equal to $0.5 \cdot \theta$, diamond shaped point in the figures would be a better alternative for both processes. This tuning parameter result in a $M_s$-value of 1.29 and 1.23 for the two cases, respectively. For these two cases the tuning parameter could even be chosen smaller than $0.5 \cdot \theta$.

3.2.3 Summary of Pareto-optimal vs. SIMC tunings

As depicted in Figures 3.2, 3.3 and 3.4 the SIMC tuning rules result in controllers with close to zero non-optimality loss for $M_s$ values in the domain $[1.3 - 2.0]$. The SIMC PI tunings is shown to follow the Pareto-optimal surprisingly well over the this $M_s$-domain. The Pareto-optimal curves are only calculated in the domain $M_s = [1.25 - 3]$, as the MATLAB-solved did not converge for smaller $M_s$-values.

All the simulations show that there will be a profit from implementing a PID controller instead of a PI controller, as the latter underperform on average approximately 100 %.

There is a clear shift towards lower $M_s$ values for the tuning parameter $\tau_c$ as the ratio $\frac{\tau_2}{\tau_1}$ decreases and the ratio $\frac{\tau_2}{\theta}$ increases.
3.2 Pareto-optimal vs. SIMC tunings

Figure 3.4: Pareto optimal (PO) vs. SIMC tuning curves for second order plus time delay processes on the form \( g_P = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot \exp(-\theta s), \) with \( \frac{\tau_2}{\tau_1} = 0.3. \)
3.3 Step responses

The Pareto-optimal and SIMC tunings has been tested by performing step changes in setpoint and input disturbance. The SIMULINK block diagram is included in Appendix A. The controller tunings used corresponds to a $M_s$ value of 1.7. All the tuning variables, in parallel form, used for the step response experiments are presented in Table 3.2.

In the subsequent sections the step response experiments for the different cases are presented by the use of plots of the output, $y$, and input, $u$, as functions of time, $t$. 
Table 3.2: Tuning variables in parallel form for PID and PI controllers, for both Pareto-optimal and SIMC tunings. $M_s = 1.7$

<table>
<thead>
<tr>
<th>Case</th>
<th>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{(-s)}$</th>
<th>Pareto-optimal</th>
<th>SIMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PID</td>
<td>PI</td>
</tr>
<tr>
<td>Case 1</td>
<td>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{(-s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.5$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.5$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.5$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 2</td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{(-s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.8$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.8$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.8$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 3</td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{(-s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.3$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.3$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 0.3$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 4</td>
<td>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{(-\frac{1}{2} s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.5$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 7</td>
<td>$\frac{1}{(s+1)(0.5s+1)} \cdot e^{(-0.25 s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.5$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 5</td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{(-\frac{8}{15} s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.8$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 8</td>
<td>$\frac{1}{(s+1)(0.8s+1)} \cdot e^{(-0.4 s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.8$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 6</td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{(-0.2 s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.3$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 1.5$</td>
<td>$\tau_D$</td>
</tr>
<tr>
<td>Case 9</td>
<td>$\frac{1}{(s+1)(0.3s+1)} \cdot e^{(-\frac{3}{20} s)}$</td>
<td>$\frac{\tau_2}{\theta} = 0.3$</td>
<td>$K_c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_f$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\frac{\tau_2}{\theta} = 2.0$</td>
<td>$\tau_D$</td>
</tr>
</tbody>
</table>
3.3 Step responses

3.3.1 Time delay dominated processes

Figure 3.5a, 3.5b and 3.5c depicts the step responses for the time delay dominated processes, i.e., case 1, 2 and 3, respectively. For the time delay dominated processes the SIMC PID tunings have a bit slower response and a higher overshoot compared with the Pareto-optimal PID in the output, $y$. Compared to the PI controllers, both the Pareto-optimal and the SIMC, the PID controllers have a faster response with approximately the same overshoot for a setpoint change. The response to a disturbance is both faster and with less overshoot.

As can be seen from the figures, the Pareto-optimal and SIMC PID controllers have a spike in the input, $u$, when they are subjected to a setpoint change. This is due to the derivative action in the controller. The derivative part change the manipulated variable, i.e., the input, proportional to the derivative of the controlled variable. As a setpoint change is performed, this becomes infinity, and the spike in the input function is observed. This could be avoided by reconstructing the controller so that the derivative part is only dependent on the feedback and not on the input. This has not been implemented in this project as the problem would only be transferred to a step in the output disturbance. As stated earlier, the input step is treated as a special case of output disturbance.

As the ratio $\frac{\tau_2}{\tau_1}$ decrease, the difference between the PID and PI controllers gets smaller. This correspond the the fact that in the limit $\tau_2 \to 0$, the PID controller should be equal to the PI controller.
3.3 Step responses

Figure 3.5: Step responses for time delay dominated second order plus time delay processes on the form
\[ g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot exp(-\theta s). \]
3.3 Step responses

3.3.2 Lag dominated processes

Figure 3.6a and 3.6b depicts case 4 and case 7, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.5, whilst the ratio $\frac{\tau_2}{\theta}$ is 1.5 and 2.0, respectively. The output performance, $y$, can be seen to be nearly optimal for both the PID and PI controllers. The SIMC PID controller has a little slower response than the corresponding Pareto-optimal PID controller, but result in less overshoot for both cases. It is also observed that the SIMC PID controller reaches the new setpoint in less time than the Pareto-optimal PID controller. The faster response and higher overshoot points to a more aggressive controller, which is confirmed by the values given in Table 3.2.

Both the Pareto-optimal and SIMC PI controllers have a slower response, higher overshoot and uses more time to stabilize at the new setpoint. This slower, smoother control is confirmed by evaluating the input, $u$, as it is much more aggressive for the PID controllers than for the PI controllers. Again a derivative spike is observed for the PID controllers at the setpoint step. The controllers gets more aggressive as the ratio $\frac{\tau_2}{\theta}$ increases, which follows from the SIMC rules.

For the disturbance rejection some of the same observations can be made. The respective SIMC controllers behaves almost close to optimal with only a little higher overshoot and the need of some extra time before stabilizing, compared to the Pareto-optimal controllers. This observation decrease as the $\frac{\tau_2}{\theta}$ ratio increases as expected for the more aggressive controllers.
Figure 3.6: Step responses for Pareto-optimal PID and PI, and SIMC PID and PI for second order plus time delay processes on the form $g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot \exp(-\theta s)$. $M_s = 1.7$ and $\tau_2/\tau_1 = 0.5$. 

(a) Case 4, $\tau_2/\tau_1 = 1.5$. 

(b) Case 7, $\tau_2/\tau_1 = 2.0$. 

3.3 Step responses 3 RESULTS AND DISCUSSION
3.3 Step responses

Figure 3.7a and 3.7b depicts case 5 and case 8, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.8, whilst the ratio $\frac{\tau_2}{\theta}$ is 1.5 and 2.0, respectively. The output response, $y$, on a step change in setpoint can be seen to be nearly optimal for both the PID and PI controllers. The SIMC PID controller has a little slower response than the corresponding Pareto-optimal PID controller, but result in less overshoot for case 8. It is also observed that the SIMC PID controller reaches the new setpoint in less time than the Pareto-optimal PID controller. Both the Pareto-optimal and SIMC PI controllers have a slower response, higher overshoot and uses more time to stabilize at the new setpoint. This slower and smoother control is confirmed by evaluating the input, $u$, as it is much more aggressive for the PID controllers than for the PI controllers, or by examination of Table 3.2. Again a derivative spike is observed for the PID controllers at the setpoint step. The controllers gets more aggressive as the ratio $\frac{\tau_2}{\theta}$ increases.

For the disturbance rejection some of the same observations can be made. The respective SIMC controllers behaves almost close to optimal with only a little higher overshoot and the need of some extra time before stabilizing, compared to the Pareto-optimal controllers. This observation decrease as the $\frac{\tau_2}{\theta}$ ratio increases as expected for the more aggressive controllers.
3.3 Step responses

RESULTS AND DISCUSSION

Figure 3.7: Step responses for Pareto-optimal PID and PI, and SIMC PID and PI for second order plus time delay processes on the form \( g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \exp(-\theta s) \). \( M_s = 1.7 \) and \( \frac{\tau_2}{\tau_1} = 0.8 \).
3.3 Step responses

Figure 3.8a and 3.8b depicts case 6 and case 9, respectively. In both these cases the ratio $\frac{\tau_2}{\tau_1}$ is kept constant at 0.3, whilst the ratio $\frac{\tau_2}{\dot{\theta}}$ is 1.5 and 2.0, respectively. The output performance, $y$, on a step change in setpoint can be seen to be nearly optimal for both the PID and PI controllers. The SIMC PID controller has a less aggressive response than the corresponding Pareto-optimal PID controller, and thus result in less overshoot for both cases. It is also observed that the SIMC PID controller reaches the new setpoint in less time than the Pareto-optimal PID controller. Both the Pareto-optimal and SIMC PI controllers have a slower response, higher overshoot and uses more time to stabilize at the new setpoint. This slower, smoother control is confirmed by evaluating the input, $u$, as it is much more aggressive for the PID controllers than for the PI controllers. Again a derivative spike is observed for the PID controllers at the setpoint step. The controllers gets more aggressive as the ratio $\frac{\tau_2}{\dot{\theta}}$ increases.

For the disturbance rejection some of the same observations can be made. The respective SIMC controllers behaves almost close to optimal with only a little higher overshoot and the need of some extra time before stabilizing, compared to the Pareto-optimal controllers. This observation decrease as the $\frac{\tau_2}{\dot{\theta}}$ ratio increases as expected for the more aggressive controllers.
3.3 Step responses

Figure 3.8: Step responses for Pareto-optimal PID and PI, and SIMC PID and PI for second order plus time delay processes on the form \( g_p = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot \exp(-\theta s) \). \( M_s = 1.7 \) and \( \frac{\tau_2}{\tau_1} = 0.3 \).
3.3.3 Summary step responses

As depicted in Figures 3.6, 3.7, and 3.8, the SIMC tuning rules result in controllers with a good trade-off between setpoint performance and disturbance rejection, if tunings corresponding to $M_s = 1.7$ is used. As can be seen from both the figures and the values in Table 3.2, the Pareto-optimal controllers behave more aggressive than the corresponding SIMC controllers. This result in enhanced disturbance rejection, for five out of six cases, while the performance of setpoint response is reduced.

The difference between the SIMC PID and PI controllers are seen to decrease as the ratio $\frac{\tau_2}{\tau_1}$ decreases. As $\tau_2$ tends to zero, the PID and PI controllers should be the same. Hence, this observation fits well with theory.

3.4 Challenges and future work

Throughout the work with this project the major challenge has been to obtain solutions from the numeric solver in MATLAB. The problem at hand, is a minimization problem to a convex function and to find the right solution has not been easy. Many more cases have been attempted, without been able to make them converge.

The cases tested in this project cannot be used as a satisfactory basis for any conclusions regarding the SIMC tuning rules, but they can be used as a starting point. So far the SIMC tuning rules seems to perform close to optimal, and give good trade-off between performance and robustness. For future work the solution algorithm may have to be improved. In addition to be able to solve for $M_s < 1.25$ a broader specter of processes can be examined and more accurate conclusions can be drawn. Pareto-optimal tunings for a FOPTD process on the form: $g_p = \frac{1}{s+1} e^{-\theta s}$, should also be calculated to show how much there is to profit from implementing a PID instead of a PI controller in the limit as $\tau_2$ tend to zero.
4 Conclusions

The rather small selection of cases tested in the project does not give a solid foundation to build any conclusions on, however it can be used as a starting point. The performed calculations and simulations show that the SIMC PID tuning rules give close to optimal performance when $M_s$ is used as a measure of robustness and a weighted function of the absolute integral error is used as a measure for performance. The investigations show that the recommended choice of not to implement a PID controller for time delay dominated process will be dependent on the process. For the three cases tested in this project the SIMC PI controller underperformed, on average, approximately 35%. The performance reward by implementing a PID controller for these processes must be compared with the extra price and complexity a PID controller introduces. For the lag dominated cases tested in this project the PID controller is shown to significantly overperform the PI controller, and thus the reward of implementing a PID controller is much greater than for the time delay dominated processes.

The recommended choice of the SIMC tuning parameter, $\tau_c = \theta$ is shown to hold for processes with a $\frac{\tau_2}{\tau_1}$ ratio greater than 0.5. As the ratio decreases the corresponding $M_s$ values decreases, and the controller get more and more robust. As the trade-off between robustness and performance always prevails, the controller performance is decreased. If the $\tau_c$ value is decreased, the controller will lose some robustness, but gain performance. A $\tau_c = 0.5\theta$, or lower, is thus recommended to increase the performance.

Compared to the Pareto-optimal tunings, the SIMC tuning rules result in less aggressive controllers. These controllers have better setpoint performance, and slightly poorer disturbance rejection.
References


A MATLAB Scripts

In the following sections the different MATLAB scripts, used in this project, are presented. They are presented in the order which they need to be executed, that is:

1. Pareto optimal PID tunings (mainOptimalTuningPID.m).
2. Pareto optimal PI tunings (mainOptimalTuningPI.m).
3. Pareto optimal vs. SIMC tunings (mainPoVsSimcPlot.m).
4. Step response (mainStepResponsePlotPo.m).
5. Parallel tunings (tuningParmParallel.m).

A.1 Obtain pareto optimal tunings

A.1.1 Optimal PID tunings, main file

1 % Script for generating the PO−PID curve for a given process gp
2 % The controller used is an ideal PID controller
3 % Written by: Martin S. Foss, fall 2012
4
5 %clc
6 clear all
7
8 global gp msEq iaeWeights manWeights iaeTuning
9
10 %Adding "sharedFiles" to MatLab search directory
11 curDir = pwd;
12 mainDir = fileparts(curDir);
13 sharedDir = fullfile(mainDir,'sharedFiles');
14 addpath(sharedDir);
15
16 tic
17 modelId = 9;
A.1 Obtain pareto optimal tunings

MATLAB SCRIPTS

% getting the model

gp = model(modelId); %getting the model

%% Finding the iaeWeights
fprintf('Finding Optimal IAE Weights \n')
fprintf('Case: %g\n',modelId)
fprintf('***************************\n')
fprintf('costFun(iae) \t minTuning \t Ms \t exitFlag \n')
fprintf('−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
')

msEq = 1.59; %Ms for the iaeWeights
opt = optimset('algorithm','active-set','Display','off','TolCon',1e-4);

%Initial solution guesses
X0 = [0.9 0.5 0.4 %case 1
1.0 0.6 0.6 %case 2
0.8 0.7 0.3 %case 3
2.2 1.5 0.8 %case 4
1.7 0.9 0.8 %case 5
3.2 2.4 0.8 %case 6
2.9 1.9 1.0 %case 7
2.2 1.2 1.0 %case 8
4.2 3.2 1.0]; %case 9
x0 = X0(modelId,:)';

iaeWeights = [1; 1]; %cost function weigths
manWeights = [1 0];

[minTuningSp,iaeSp,exitFlagSp] = fmincon(@costFun,x0,[],[],[],[],
[0;0;0],[],[],@conFun,opt);

fprintf('%0.2f \t \t %.2f %.2f %.2f \t \t %.2f %.2f %.2f \t %i \t \n',iaeSp,...
minTuningSp, msEq, exitFlagSp)

%%

%Initial solution guesses
A.1 Obtain pareto optimal tunings

MATLAB SCRIPTS

```matlab
X0 = [0.9 0.7 0.5 %case 1
1.0 0.6 0.6 %case 2
0.8 0.7 0.35 %case 3
2.2 2.3 0.8 %case 4
1.7 1.3 0.8 %case 5
3.3 5.0 0.8 %case 6
3.0 3.4 1.0 %case 7
2.2 1.8 1.0 %case 8
4.5 7.6 1.0]; %case 9
x0 = X0(modelId,:)

manWeights = [0 1];

[minTuningD,iaeD,exitFlagD] = fmincon(@costFun,x0,[],[],[],
[0;0;0],[],@conFun,opt);

fprintf('%0.2f 	 	 %.2f %.2f %.2f 	 %.2f	 %i 	 
',iaeD,...
minTuningD, msEq, exitFlagD)

iaeWeights = [iaeSp; iaeD];

%% Generating curve
i = 1; %iteration counter for command window printout
minTuning = []; %matrices for storing results
costTuning = [];
iaeOptTun = [];

msSpace = 1.25:0.05:3; %Ms search range

%Initial solution guesses
X0 = [0.7 0.5 0.4 %case 1
0.8 0.6 0.5 %case 2
0.4 0.4 0.3 %case 3
1.1 1.1 0.4 %case 4
0.9 0.7 0.5 %case 5

A-3
A.1 Obtain pareto optimal tunings

```matlab
x0 = X0(modelId, :);
manWeights = [.5, .5];

fprintf('Generating the PO Curve
')
fprintf('***********************
')
fprintf('
')
fprintf('Number of iterations: %i 
', length(msSpace))
fprintf('
')
fprintf('costFun(%.1f, %.1f) 	 minTuning 	 	 Ms ', manWeights)
fprintf('exitFlag 	 iterations left 
')
fprintf('−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
')
fprintf('−−−−−−−−−−−−−−−−−−−
')

opt = optimset('algorithm', 'active-set', 'Display', 'off', 'TolCon', 1e-4);
  % 'active-set', 'trust-region-reflective', 'interior-point',
  % 'interior-point-convex', 'levenberg-marquardt',
  % 'trust-region-dogleg', 'lm-line-search', or 'sqp'.

% Optimizing
for msEq = msSpace;

[minTuningTemp, iaeTuningTemp, exitFlagTuningTemp] = ... 
  fmincon(@costFun, x0, [], [], [], [], [0; 0; 0], [], @conFun, opt);

minTuning(:, i) = minTuningTemp; % storing results
costTuning(i) = iaeTuningTemp;
exitFlagTuning(i) = exitFlagTuningTemp;
iaeOptTun(i, :) = iaeTuning;

if modelId == 2 && i == 1 || modelId == 3 && i == 1 ||... 
  modelId == 5 && i == 1
  x0 = 1.1*minTuningTemp;
elseif modelId == 6 && i == 7 || modelId == 9 && i == 32
  x0 = 0.9*minTuningTemp;
```

A.1 Obtain pareto optimal tunings

MATLAB SCRIPTS

```matlab
elseif modelId == 9 && i == 17
    x0 = 0.9*minTuningTemp;
elseif modelId == 9 && i == 16 || modelId == 9 && i >= 34
    x0 = 0.91*minTuningTemp;
else
    x0 = minTuningTemp;
end

fprintf('%0.2f 	 	 	 %.2f %.2f %.2f 	 %.2f	 %i 	 		 %i 
',
    iaeTuningTemp, minTuningTemp, msEq, exitFlagTuningTemp,
    length(msSpace)-i)

i = i + 1;  % updating iteration counter
end

fprintf('
')
fprintf('Calculation Finished!
')
fprintf('=====================
')
toc

%% Ploting the results
% Cost function, J vs. Ms
figure(modelId)
clf
h(1) = plot(msSpace,costTuning);
axis([1.2 3 0.8 1.3])
xlabel('Robustness, $M_s$', 'interpreter','latex','FontSize',14)
ylabel('Performance, $J$', 'interpreter','latex','FontSize',14)
titleName = {'Case 1','Case 2','Case 3','Case 4','Case 5','Case 6','Case 7','Case 8','Case 9'};
title(titleName{modelId},'interpreter','latex','FontSize',14)

%% Storing results
modelName = {'case1','case2','case3','case4','case5','case6','case7','case8','case9'};
res.case = num2str(modelName{modelId});
```

A-5
A.1 Obtain pareto optimal tunings

The following MATLAB scripts are used to obtain pareto optimal tunings for a given process.

```matlab
res.POpid.minSp.tuning = minTuningSp;
res.POpid.minSp.iae = iaeSp;
res.POpid.minSp.exitFlag = exitFlagSp;
res.POpid.minD.tuning = minTuningD;
res.POpid.minD.iae = iaeD;
res.POpid.minD.exitFlag = exitFlagD;
res.POpid.minTuning.ms = msSpace;
res.POpid.minTuning.tuning = minTuning;
res.POpid.minTuning.costFun = costTuning;
res.POpid.minTuning.exitFlag = exitFlagTuning;
res.POpid.minTuning.iae = iaeOptTun;

% pause
% saving struct
save([mainDir, '\dataFiles\', 'resPOpid_', ...
    num2str(modelName{modelId}), '.mat'], 'res') %saving "globaly"
save(['poPIDresults\', 'resPOpid_', ...
    num2str(modelName{modelId}), '.mat'], 'res') %saving "locally"

% saving figures
saveas(h(1), ['figures\', 'POpidCurve_', num2str(modelName{modelId}), '.fig'])

restoredefaultpath
```

A.1.2 Optimal PI tunings, main file

```matlab
% Script for generating the PO-PI curve for a given process gp
% The controller used is an ideal PI controller
% Written by: Martin S. Foss, fall 2012

%clc
clear all

global gp msEq iaeWeights manWeights iaeTuning

%Adding "sharedFiles" to MatLab search directory
```
curDir = pwd;
mainDir = fileparts(curDir);
sharedDir = fullfile(mainDir,'sharedFiles');
addpath(sharedDir);

%%
tic
modelId = 9;
modelName = {'case1','case2','case3','case4','case5','case6',... 'case7','case8','case9','case10'};
gp = model(modelId); % getting the model

%% Finding the iaeWeights for the PI-controller
fprintf('Finding Optimal IAE Weights (PI-controller) \n')
fprintf('Case: %g
',modelId)
fprintf('***************************
')
fprintf('costFun(iae) \t minTuning \t	 Ms \t exitFlag \n')
fprintf('-----------------------------------------------------------------------------------
')
msEq = 1.59; %Ms for the iaeWeights
opt = optimset('algorithm','active-set','Display','off','TolCon',1e-4);
% 'active-set', 'trust-region-reflective', 'interior-point',
% 'interior-point-convex', 'levenberg-marquardt',
% 'trust-region-dogleg', 'lm-line-search', or 'sqp'.
%
%Initial solution guesses
X0 = [0.5 0.4 0 %case 1
0.5 0.3 0 %case 2
0.5 0.4 0 %case 3
1.1 0.7 0 %case 4
0.8 0.5 0 %case 5
1.5 1.2 0 %case 6
1.3 0.9 0 %case 7
1.0 0.6 0 %case 8
1.9 1.4 0]; %case 9
x0 = X0(modelId,:);

iaeWeights = [1; 1]; %cost function weights
manWeights = [1 0];

Aeq = [0 0 1]; % constraints
Beq = 0;

[minTuningSp,iaeSp,exitFlagSp] = fmincon(@costFun,x0,[],[],Aeq,Beq,...
    [0;0;0],[],@conFun,opt);

fprintf('%0.2f		%.2f%.2f %.2f	%.2f
',iaeSp,...
    minTuningSp, msEq, exitFlagSp)

%%

%Initial solution guesses
X0 = [0.5 0.4 0 %case 1
    0.5 0.3 0 %case 2
    0.5 0.4 0 %case 3
    0.9 0.8 0 %case 4
    0.7 0.5 0 %case 5
    1.3 1.3 0 %case 6
    1.1 0.9 0 %case 7
    0.9 0.6 0 %case 8
    1.5 1.6 0]; %case 9
x0 = X0(modelId,:);'

manWeights = [0 1];

[minTuningD,iaeD,exitFlagD] = fmincon(@costFun,x0,[],[],Aeq,Beq,...
    [0;0;0],[],@conFun,opt);

fprintf('%0.2f		%.2f%.2f %.2f	%.2f
',iaeD,...
    minTuningD, msEq, exitFlagD)

fprintf('
');
fprintf('
');
fprintf('
');
fprintf('
');

%% Generating curve
try
    load(fullfile(mainDir,'dataFiles',
    try
        load(fullfile(mainDir,'dataFiles',

A.1 Obtain pareto optimal tunings

```matlab
num2str(modelName{modelId},'mat'))
catch me
    ME = MException(me.identifier,...
        'could not open file, check correct modelId and data folder!');
    throw(ME)
end
minSp = res.POpid.minSp.iae;
minD = res.POpid.minD.iae;
iaeWeights = [minSp; minD];
i = 1; %iteration counter for command window printout
minTuning = []; %matrices for storing results
costTuning = [];
iaeOptTun = [];
msSpace = 1.25:0.05:3; %the Ms search range

%Initial solution guesses
X0 = [0.2 0.2 0 %case 1
    0.3 0.2 0 %case 2
    0.2 0.2 0 %case 3
    0.4 0.4 0 %case 4
    0.5 0.4 0 %case 5
    0.6 0.6 0 %case 6
    0.5 0.4 0 %case 7
    0.4 0.3 0 %case 8
    1.2 1.1 0]; %case 9
x0 = X0(modelId,:);
manWeights = [.5, .5];

fprintf('Generating the PO Curve
')
fprintf('**************************
')
fprintf('Number of iterations: %i 
', length(msSpace))
fprintf('costFun(%.1f, %.1f) 	 minTuning 	 	 Ms',manWeights)
fprintf(' 	 exitFlag 	 iterations left 
')
```
A.1 Obtain pareto optimal tunings

MATLAB SCRIPTS

128 fprintf('--------------------------------
')
129 opt = optimset('algorithm','active-set','Display','off','TolCon',1e-4);
130 %'active-set', 'trust-region-reflective', 'interior-point',
131 %'interior-point-convex', 'levenberg-marquardt',
132 %'trust-region-dogleg', 'lm-line-search', or 'sqp'.
133
134 %Optimizing
135 for msEq = msSpace;
136
137 [minTuningTemp, iaeTuningTemp, exitFlagTuningTemp] = ...
138 fmincon(@costFun,x0,[],[],Aeq,Beq,[0;0;0],[],@conFun,opt);
139
140 minTuning(:,i) = minTuningTemp; %storing results
141 costTuning(i) = iaeTuningTemp;
142 exitFlagTuning(i) = exitFlagTuningTemp;
143 iaeOptTun(i,:) = iaeTuning;
144
145 if modelId == 1 && i == 22 || modelId == 1 && i == 32 || ...
146 modelId == 3 && i == 22 || modelId == 9 && i == 29 || ...
147 modelId == 9 && i >= 32
148 x0 = 0.9*minTuningTemp;
149 elseif modelId == 5 && i == 1 || modelId == 5 && i == 2
150 x0 = 1.2*minTuningTemp;
151 else
152 x0 = minTuningTemp;
153 end
154
155 fprintf('%0.2f 	 %.2f %.2f %.2f 	 %.2f	 %i 	 		 %i 
',... 
156 iaeTuningTemp, minTuningTemp, msEq, exitFlagTuningTemp,... 
157 length(msSpace)-i)
158 i = i + 1; %updating iteration counter
159 end
160
161 fprintf('
')
162 fprintf('Calculation Finished!
')
163 fprintf('=====================
')
164 toc

A-10
% Plotting the results
% Cost function, J, vs. Ms
figure(modelId)
clf
h(1) = plot(msSpace,costTuning);
xlabel('Robustness, $M_s$', 'interpreter','latex','FontSize',14)
ylabel('Performance, $J$', 'interpreter','latex','FontSize',14)
titleName = {'Case 1','Case 2','Case 3','Case 4','Case 5','Case 6','Case 7','Case 8','Case 9'};
title(titleName{modelId},'interpreter','latex','FontSize',14)

% Storing results (in the same struct as the PO (PID) tunings)
res.POpi.minSp.tuning = minTuningSp;
res.POpi.minSp.iae = iaeSp;
res.POpi.minSp.exitFlag = exitFlagSp;
res.POpi.minD.tuning = minTuningD;
res.POpi.minD.iae = iaeD;
res.POpi.minD.exitFlag = exitFlagD;
res.POpi.minTuning.ms = msSpace;
res.POpi.minTuning.tuning = minTuning;
res.POpi.minTuning.costFun = costTuning;
res.POpi.minTuning.exitFlag = exitFlagTuning;
res.POpi.minTuning.iae = iaeOptTun;

pause
% saving struct
save(fullfile(mainDir,'\dataFiles\','resPO_','...num2str(modelName{modelId}),'\mat\','res'))
save(fullfile('poPIresults\','resPO_','...num2str(modelName{modelId}),'\mat\','res'))

% saving figures
saveas(h(1),[\'figures\','POpiCurve_','num2str(modelName{modelId}),'\fig\'])
restoredefaultpath
A.1.3 Cost function

```matlab
function J = costFun(x)
global gp iaeWeights manWeights iaeTuning

%Controller
gc = controller(x(1),x(2),x(3));

%Feedback loops
dgey = feedback(1,gc*gp);
ged = feedback(gp*1,gc,1);

%Output response to input and output disturbance
sys = [ggy;ged];
[e,t]=step(sys,100);

iaeTuning = iae(t,e);
J = manWeights*(iaeTuning./iaeWeights);
return
```

A.1.4 Constraints

```matlab
function [c, ceq] = conFun(x0)
global gp msEq

c = [];
ceq = msEq - ms(gp, controller(x0(1),x0(2),x0(3)));
return
```

A.2 Obtain PO vs. SIMC tuning plots

A.2.1 Main file
% Script for creating plots comparing SIMC tunings with the PO curve
% Written by: Martin S. Foss, fall 2012

clear all

%Adding "sharedFiles" to MatLab search directory
curDir = pwd;
mainDir = fileparts(curDir);
sharedDir = fullfile(mainDir,'sharedFiles');
addpath(sharedDir);

for m = 1:8
    modelId = 9;
    modelName = {'case1','case2','case3','case4','case5','case6','...'
                  'case7','case8','case9','case10'};

    fprintf('Finding SIMC tunings \n')
    fprintf('Case: %g
',modelId)
    fprintf('***************************

    try
        load(fullfile(mainDir,'dataFiles',
                        ['resPO_','...
                         num2str(modelName{modelId}),'.mat'])) %loading datafiles
    catch me
        ME = MException(me.identifier,...
                        'could not open file, check correct modelId!');
        throw(ME)
    end

    minSp = res.POpid.minSp.iae;    %loading iaeWeights;
    minD = res.POpid.minD.iae;
    minWeights = [minSp;minD];

    manWeights = [.5 .5];    %setting manWeights

    tcSpace = 0:0.1:5;    %closed loop time constant search rang for simc
%% SIMC PI–controller

gp = model(modelId); % get the model

simcJ = []; % setting up result matrices
simcMs = [];
simcTuning = [];
i = 1; % setting iteration counter

for tc = tcSpace
    [simcTuningTemp, simcGc] = simcPID(gp,tc); % finding the simc PID tuning
    jTemp = costFun(gp,simcGc,minWeights,manWeights); % finding the cost
    msTemp = ms(gp,simcGc); % finding simc ms value
    simcJ(i) = jTemp; % storing results
    simcMs(i) = msTemp;
simcTuning(:,i) = simcTuningTemp;
i = i+1; % updating iteration counter
end

% Finding reference dots for SIMC PID–tuning;
simcRefJ = []; % setting up result matrices
simcRefMs = [];

for tc = [.5 1 1.5]
    [simcTuningTemp simcGc] = simcPID(gp,tc); % finding the simc PID tuning
    jTemp = costFun(gp,simcGc,minWeights,manWeights); % finding the cost
    msTemp = ms(gp,simcGc); % finding simc ms value
    simcRefJ = [simcRefJ jTemp]; % storing results
    simcRefMs = [simcRefMs msTemp];
end

%% SIMC PI–controller

gpPI = modelPI(modelId); % get the model

simcJPI = []; % setting up result matrices
simcMsPI = [];
simcTuningPI = [];
i = 1; % setting iteration counter
A.2  Obtain PO vs. SIMC tuning plots

MATLAB SCRIPTS

```
for tc = tcSpace
    [simcTuningTemp, simcGc] = simcPI(gpPI,tc); %finding the simc PI-tuning
    jTemp = costFun(gp,simcGc,minWeights,manWeights); %finding the cost
    msTemp = ms(gp,simcGc); %finding simc ms value
    simcJPI(i) = jTemp;  %storing results
    simcMsPI(i) = msTemp;
    simcTuningPI(:,i) = simcTuningTemp;
    i = i+1; %update the iteration counter
end

%Finding reference dots for SIMC PI-tuning;
simcRefJPI = []; %setting up results matrices
simcRefMsPI = [];

for tc = [.5 1 1.5]
    [simcTuningTemp simcGc] = simcPI(gpPI,tc); %finding the simc PI-tuning
    jTemp = costFun(gp,simcGc,minWeights,manWeights); %finding the cost
    msTemp = ms(gp,simcGc); %finding simc ms value
    simcRefJPI = [simcRefJPI jTemp]; %storing results
    simcRefMsPI = [simcRefMsPI msTemp];
end

%% Plotting the results
colorSet = colormap('lines');
figure(modelId)
clf

h = plot(res.POpid.minTuning.ms,res.POpid.minTuning.costFun,... %PO(PID)
    res.POpi.minTuning.ms,res.POpi.minTuning.costFun,... %PO(PI)
    simcMs,simcJ,... %SIMC(PID)
    simcMsPI,simcJPI); %SIMC(PI)
set(h,'LineWidth',1.5)

markerStyles = cellstr(char('d','o','s'));

%Points for SIMC (PID)
```

A.2 Obtain PO vs. SIMC tuning plots

MATLAB

```matlab
hold on
for i = 1:length(simcRefMs)
    h(i) = plot(simcRefMs(i), simcRefJ(i));
    set(h(i), 'color', colorSet(3,:), 'LineWidth', 1.5, 'Marker', ...
        markerStyles(i), 'MarkerSize', 10);
end

axis([1 3 0.75 8]);

% Points for SIMC (PI)
for i = 1:length(simcRefMsPI)
    h(i) = plot(simcRefMsPI(i), simcRefJPI(i));
    set(h(i), 'color', colorSet(4,:), 'linewidth', 1.5, 'Marker', ...
        markerStyles(i), 'MarkerSize', 10);
end

%% Printing info
tau2tau1Info = {'$\frac{\tau_2}{\tau_1}=0.5$', ...
    '$\frac{\tau_2}{\tau_1}=0.8$', ...
    '$\frac{\tau_2}{\tau_1}=0.3$', ...
    '$\frac{\tau_2}{\tau_1}=0.5$', ...
    '$\frac{\tau_2}{\tau_1}=0.8$', ...
    '$\frac{\tau_2}{\tau_1}=0.3$'};

tau2thetaInfo = {'$\frac{\tau_2}{\theta}=0.5$', ...
    '$\frac{\tau_2}{\theta}=0.8$', ...
    '$\frac{\tau_2}{\theta}=0.3$', ...
    '$\frac{\tau_2}{\theta}=1.5$', ...
    '$\frac{\tau_2}{\theta}=2.0$', ...
    '$\frac{\tau_2}{\theta}=2.0$'};

curveInfo = {'PO (PID)', 'PO (PI)', 'SIMC (PID)', 'SIMC (PI)'};

pointInfo = {'$\tau_c=0.5\theta$', '$\tau_c=\theta$', '$\tau_c=1.5\theta$'};

infoFontSize = 18;
```
A.2 Obtain PO vs. SIMC tuning plots

MATLAB SCRIPTS

```matlab
xlab = xlabel('Robustness, $M_s$');
ylab = ylabel('Performance, $J(c)$');
set(xlab,'interpreter','latex','fontsize',infoFontSize)
set(ylab,'interpreter','latex','fontsize',infoFontSize)
set(gca,'fontsize',16,'FontName','Times New Roman')

% Model info
[figx figy] = dsxy2figxy(gca,2.4,5.5);
textBoxTau1Tau2Info = annotation('textbox',[figx figy .07 .03],...
    'string',tau2tauInfo(modelId),'interpreter','latex',...
    'fontsize',infoFontSize,'color',[0 0 0],'FitBoxToText','on',...
    'LineStyle','none');

% Markers for different tau_c
h = plot(2.4,4.5,'marker',markerStyles{1},'markerSize',10,...
    'linewidth',1.5,'color',[0 0 0]);
[figx figy] = dsxy2figxy(gca,2.4,4.5);
point1 = annotation('textarrow',[figx+0.025 figx+0.015],[figy figy],...
    'string',pointInfo{1},'interpreter','latex',...
    'fontsize',infoFontSize,'headstyle','none');

h = plot(2.4,4,'marker',markerStyles{2},'markerSize',10,...
    'linewidth',1.5,'color',[0 0 0]);
[figx figy] = dsxy2figxy(gca,2.4,4);
point2 = annotation('textarrow',[figx+0.025 figx+0.015],[figy figy],...
    'string',pointInfo{2},'interpreter','latex',...
    'fontsize',infoFontSize,'headstyle','none');

h = plot(2.4,3.5,'marker',markerStyles{3},'markerSize',10,...
    'linewidth',1.5,'color',[0 0 0]);
[figx figy] = dsxy2figxy(gca,2.4,3.5);
```

A-17
A.2 Obtain PO vs. SIMC tuning plots

MATLAB SCRIPTS

point3 = annotation('textarrow',[figx+0.025 figx+0.015],[figy figy],...
    'string',pointInfo{3},'interpreter','latex',...
    'fontsize',infoFontSize,'headstyle','none');

%PO (PID)
if modelId == 8
    x1 = find(res.POpid.minTuning.ms >= 1.4);
else
    x1 = find(res.POpid.minTuning.ms >= 1.5);
end
[figx figy] = dsxy2figxy(gca,res.POpid.minTuning.ms(x1(1)),...
    res.POpid.minTuning.costFun(x1(1)));  
curve1 = annotation('textarrow',[figx−0.03 figx],[figy−0.005 figy],...
    'string',curveInfo{1},'interpreter','latex',...
    'fontsize',infoFontSize,'color',colorSet(1,:));

%PO (PI)
x2 = find(res.POpi.minTuning.ms >= 2.4);
[figx figy] = dsxy2figxy(gca,res.POpi.minTuning.ms(x2(1)),...
    res.POpi.minTuning.costFun(x2(1)));  
if modelId < 4
    curve2 = annotation('textarrow',[figx+0.05 figx],[figy+0.05 figy],...
        'string',curveInfo{2},'interpreter','latex',...
        'fontsize',infoFontSize,'color',colorSet(2,:));
else
    curve2 = annotation('textarrow',[figx+0.05 figx],[figy−0.05 figy],...
        'string',curveInfo{2},'interpreter','latex',...
        'fontsize',infoFontSize,'color',colorSet(2,:));
end

%SIMC (PID)
if modelId == 1 || modelId == 2 || modelId == 3
    x3 = find(simcMs <= 1.20);
elseif modelId == 7
    x3 = find(simcMs <= 1.3);
else
    x3 = find(simcMs <= 1.4);
end
[figx figy] = dsxy2figxy(gca,simcMs(x3(1)),simcJ(x3(1)));
if modelId == 4
    curve3 = annotation('textarrow',[figx+0.05 figx],[figy+0.01 figy],...
                     'string',curveInfo{3},'interpreter','latex',...
                     'fontsize',infoFontSize,'color',colorSet(3,:));
else
    curve3 = annotation('textarrow',[figx+0.05 figx],[figy figy],...
                     'string',curveInfo{3},'interpreter','latex',...
                     'fontsize',infoFontSize,'color',colorSet(3,:));
end

%SIMC (PI)
if modelId == 5 || modelId == 7
    x4 = find(simcMsPI <= 2.1);
else
    x4 = find(simcMsPI <= 2);
end
[figx figy] = dsxy2figxy(gca,simcMsPI(x4(1)),simcJPI(x4(1)));
curve4 = annotation('textarrow',[figx+0.02 figx],[figy+0.03 figy],...
                     'string',curveInfo{4},'interpreter','latex',...
                     'fontsize',infoFontSize,'color',colorSet(4,:));

set(gca,'Layer','top','Box','on')
set(gcf,'paperpositionmode','auto')

%% Storing results and figures
res.simcPID.simcJ = simcJ;
res.simcPID.simcMs = simcMs;
res.simcPID.simcTuning = simcTuning;
res.simcPI.simcJ = simcJPI;
res.simcPI.simcMs = simcMsPI;
res.simcPI.simcTuning = simcTuningPI;

pause
save([mainDir,'\dataFiles\',resSimc_,...
     num2str(modelName{modelId}),'.mat'],'res')
save([simcResults\',resSimc_,num2str(modelName{modelId}),'.mat'],'res')
saveas(h,['SIMCfigures\','simcRes_',...
A.2 Obtain PO vs. SIMC tuning plots

MATLAB SCRIPTS

```matlab
274    num2str(modelName{modelId},'\'.eps','psc2')
275    saveas(h,['SIMCfigures\','simcRes_\',num2str(modelName{modelId}),'\'.fig'])
276    % end
277
278    restoredefaultpath

A.2.2 Cost function

1 function J = costFun(gp,gc,iaeWeights,manWeights)
2
3    % Feedback loops
4    gey = feedback(1,gc*gp);
5    ged = feedback(gp*-1,gc,1);
6
7    % Output response to input and output disturbance
8    sys = [gey;ged];
9    [e,t] = step(sys,100);
10
11    iaeTuning = iae(t,e);
12
13    J = manWeights*(iaeTuning./iaeWeights);
14    return

A.2.3 SIMC PI-tunings

1    % function for simc PI tuning
2    % input: 2nd order model + tuning parameter
3    % returns: simc tuning parameter for a PI controller on the form
4    % gc = K + I/s
5    % Written by: Martin S. Foss, fall 2012
6
7    function [tuning gc] = simcPI(gp,tc)
8
9    % Determining the model
10    t = gp.den{1}(1);          % time constant
11    d = totaldelay(gp);      % time delay
12    g = gp.num{1}(end);     % gain
```

A-20
A.2 Obtain PO vs. SIMC tuning plots

MATLAB SCRIPTS

13 z = zero(gp); %zeros
14 p = pole(gp); %poles
15
16 if length(gp.den{1}) > 2
17    disp('model order to high')
18    return
19 elseif isempty(z) == 0
20    disp('model cannot contain zeros')
21    return
22 end
23
24 K = t/(g*(tc+d));
25 I = K/min(t,4*(tc+d));
26
27 tuning = [K I]';
28 [gcPID gc] = controller(K,I);
29 return

A.2.4 SIMC PID-tunings

% function for simc PID tuning
% input: 2nd order model + tuning parameter
% returns: simc tuning parameter for a PID controller on the form
gen = [K + I/s + D*s]
% Written by: Martin S. Foss, fall 2012

function [tuning, gc] = simcPID(gp,tc)

%Determining the model
10 d = totaldelay(gp); %time delay
11 g = gp.num(1)(end); %gain
12 z = zero(gp); %zeros
13 p = pole(gp); %poles
14 t1 = abs(p(2)); %time constant 1 (largest)
15 t2 = abs(p(2)/p(1)); %time constant 2 (smallest)
16
17 if nargin == 1
18    tc = d;
end

if length(gp.den{1}) > 3
    disp('model order to high')
    return
elseif isempty(z) == 0
    disp('model cannot contain zeros')
    return
end

%SIMC-tuning (series)
Kc = 1/g*(t1/(tc+d));
tau_i = min(t1,4*(tc+d));
tau_d = t2;

%SIMC-tuning (parallel)
K_merk = Kc*(1+tau_d/tau_i);
I_merk = tau_i*(1+tau_d/tau_i);
D_merk = tau_d/(1+tau_d/tau_i);

%SIMC-tuning (given parameterization)
K = K_merk;
I = K_merk/I_merk;
D = K_merk*D_merk;
tuning = [K I D]';
gc = controller(K,I,D);
return
A.3 Obtain step responses

A.3.1 Main file

% Script evaluating step response for PID controller using both PO and SIMC controller tunings
% Written by: Martin S. Foss, fall 2012

clear all
% close all
clc

% Adding "sharedFiles" to MatLab search directory
curDir = pwd;
mainDir = fileparts(curDir);
sharedDir = fullfile(mainDir,'sharedFiles');
addpath(sharedDir);

% for m = 1:8
modelId = 9; %m;
modelName = {'case1','case2','case3','case4','case5','case6',
             'case7','case8','case9','case10'};
msSet = 1.7; % define the Ms value

load(fullfile(mainDir,'dataFiles',
              ['resSimc_','num2str(modelName{modelId}),'.mat'])) % loading datafiles

% Collecting PO and SIMC tunings
index = find(res.POpid.minTuning.ms == msSet);
tuning{1} = res.POpid.minTuning.tuning(:,index(1));
index2 = find(res.POpi.minTuning.ms == msSet);
tuning{2} = res.POpi.minTuning.tuning(:,index2(1));
index3 = find(res.simcPID.simcMs <= msSet);
tuning{3} = res.simcPID.simcTuning(:,index3(1));
index4 = find(res.simcPI.simcMs <= msSet);
A.3 Obtain step responses

```matlab
35 tuning{4} = res.simcPI.simcTuning(:,index4(1));
36 gp = model(modelId); %loading the model
37 %Extracting parameters from the model
38 p = pole(gp); %poles
39 t1 = abs(p(2)); %time constant 1 (largest)
40 t2 = abs(p(2)/p(1)); %time constant 2 (smallest) = taud
41
42 % Plotting PO (PID) and SIMC (PID)
43 ysTime = 1; %define simulation parameters
dTime = 20;
stopTime = 40;

45 figure(modelId)
clf
47 colorSet = colormap('lines');
49 Y = [];
50 U = [];
51 T = [];
52
53 for i = [1 3]
54 subplot(2,1,1)
55 gc = controller(tuning{i}(1),tuning{i}(2),tuning{i}(3),t2);
56 sim('simModel') %running Simulink
57 r = plot(t,y); %plotting output vs. time
58 set(r,'Color',colorSet(i,:),'LineWidth',1.5);
59 axis([0 stopTime 0 2])
60 hold on
61 subplot(2,1,2)
62 r = plot(t,u); %plotting input vs. time
63 set(r,'Color',colorSet(i,:),'LineWidth',1.5);
64 axis([0 40 -.5 4])
65 hold on
66 Y{i} = y; %storing results
67 end
```
A.3  Obtain step responses

MATLAB SCRIPTS

U{i} = u;
T{i} = t;
end

%% Plotting PO (PI) and SIMC (PI)
for k = [2 4]
    subplot(2,1,1)
    [a gc] = controller(tuning{k}(1),tuning{k}(2));
    r = plot(t,y); %plotting output vs. time
    set(r,'Color',colorSet(k,:),'LineWidth',1.5);
    subplot(2,1,2)
    r = plot(t,u); %plotting inputs vs. time
    set(r,'Color',colorSet(k,:),'LineWidth',1.5);
    Y{k} = y; %storing results
    U{k} = u;
    T{k} = t;
end

subplot(211)
Ys = plot(t,ys,'--k','linewidth',1.5); %plotting setpoint
set(Ys,'Color',[0 0 0],'LineWidth',1.5,'LineStyle','--');

%% Printing info

msString={'$M_s$ $=$ $1.7$'};

stdFont = 16;
infoFontSize = 18;

subplot(211)
ylab = ylabel('Output, $y$');
set(ylab,'interpreter','latex','fontsize',infoFontSize)
set(gca,'fontsize',stdFont,'FontName','Times New Roman')

subplot(212)
ylab = ylabel('Input, $u$');
xlab = xlabel('Time, $t$');
set(xlab,'interpreter','latex','fontsize',infoFontSize)
set(ylab,'interpreter','latex','fontsize',infoFontSize)
set(gca,'fontsize',stdFont,'FontName','Times New Roman')

subplot(211)
[figx figy] = dsxy2figxy(gca,1.9,1.8); % (gca,3,3)
textBoxMs = annotation('textbox',[figx figy .07 .03],'string','...
msString{1},'
'interpreter','latex','fontsize',stdFont,'...
'color',[0 0 0],'FitBoxToText','on','LineStyle','none');

[figx figy] = dsxy2figxy(gca,10,1.75); % (gca,3,3)
textBoxCaseInfo = annotation('textbox',[figx figy .07 .03],'string','...
caseInfo{modelId},'
'interpreter','latex','fontsize',...
'stdFont','color',[0 0 0],'FitBoxToText','off','...
'LineStyle','none');

[figx figy] = dsxy2figxy(gca,1.9,1.55); % (gca,3,3)
textBoxSetPoint = annotation('textbox',[figx figy,.07 .03],'string','...
'\bf{\textbf{\textbf{setpoint}}},'
'interpreter','latex','...
'fontsize',stdFont,'color',[0 0 0],'FitBoxToText','...
'on','LineStyle','none');

%PO (PID)
y1 = find(Y{1}>=0.8);
[figx figy] = dsxy2figxy(gca,T{1}(y1(1)),Y{1}(y1(1)));
curve1 = annotation('textarrow',[figx+0.05 figx],[figy figy],...
'color',colorSet(1,:));
A.3 Obtain step responses

MATLAB SCRIPTS

```matlab
%PO (PI)
y2 = find(Y{2}>=0.25);
[figx figy] = dsxy2figxy(gca,T{2}(y2(1)),Y{2}(y2(1)));
curve2 = annotation('textarrow',[figx+0.05 figx],[figy figy],...
    'string',curveInfo{2},'interpreter','latex','fontsize',stdFont,...
    'color',colorSet(2,:));

%SIMC (PID)
y3 = find(Y{3}>=0.5);
[figx figy] = dsxy2figxy(gca,T{3}(y3(1)),Y{3}(y3(1)));
curve3 = annotation('textarrow',[figx+0.05 figx],[figy figy],...
    'string',curveInfo{3},'interpreter','latex','fontsize',stdFont,...
    'color',colorSet(3,:));

%SIMC (PI)
[y4val y4ind] = max(Y{4}(1:150));
[figx figy] = dsxy2figxy(gca,T{4}(y4ind),Y{4}(y4ind));
curve4 = annotation('textarrow',[figx+0.05 figx],[figy+0.02 figy],...
    'string',curveInfo{4},'interpreter','latex','fontsize',stdFont,...
    'color',colorSet(4,:));

subplot(212)

%PO (PID)
if modelId == 7
    [u1val u1ind] = min(U{1});
    [figx figy] = dsxy2figxy(gca,T{1}(u1ind),U{1}(u1ind));
curve1 = annotation('textarrow',[figx−0.05 figx],[figy figy],...
        'string',curveInfo{1},'interpreter','latex','fontsize',stdFont,'color',colorSet(1,:));
else
    u1 = find(U{1}(100:end)<=0.2);
    [figx figy] = dsxy2figxy(gca,T{1}(u1(1)+100),U{1}(u1(1)+100));
curve1 = annotation('textarrow',[figx−0.05 figx],[figy figy],...
        'string',curveInfo{1},'interpreter','latex','fontsize',stdFont,'color',colorSet(1,:));
end

%PO (PI)
if modelId == 4 || modelId == 5 || modelId == 7 || modelId == 8 || modelId == 9
```
A.3  Obtain step responses

MATLAB SCRIPTS

[u2val u2ind] = max(U{2});

[figx figy] = dsxy2figxy(gca,T{2}(u2ind),U{2}(u2ind));
curve2 = annotation('textarrow',[figx+0.05 figx],[figy+0.03 figy],...
    'string',curveInfo{2},'interpreter','latex',...
    'fontsize',stdFont,'color',colorSet(2,:));

else
    u2 = find(U{2}>=0.70);
    [figx figy] = dsxy2figxy(gca,T{2}(u2(1)),U{2}(u2(1)));
    curve2 = annotation('textarrow',[figx+0.05 figx],[figy−0.02 figy],...
        'string',curveInfo{2},'interpreter','latex','fontsize',...
        stdFont,'color',colorSet(2,:));
end

%SIMP (PID)
[u3 u3maxInd] = max(U{3}(1:150));

[figx figy] = dsxy2figxy(gca,T{3}(u3(1)+100),U{3}(u3(1)+100));
curve3 = annotation('textarrow',[figx+0.05 figx],[figy+0.02 figy],...
    'string',curveInfo{3},'interpreter','latex',...
    'fontsize',stdFont,'color',colorSet(3,:));

%SIMP (PI)
[u4maxVal u4maxInd] = max(U{4}(1:150));

[u4 u4maxInd] = find(U{4}(u4maxInd:end)<=1.12);

[figx figy] = dsxy2figxy(gca,T{4}(u4(1)+u4maxInd),U{4}(u4(1)+u4maxInd));
curve4 = annotation('textarrow',[figx+0.05 figx],[figy+0.02 figy],...
    'string',curveInfo{4},'interpreter','latex','fontsize',stdFont,....
    'color',colorSet(4,:));

%S IMP Saving figures
pause

saveas(r,['stepFigures\stepResponse_','...
    num2str(modelName{modelId}),'.eps'],psc2')
saveas(r,['stepFigures\stepResponse_','...
    num2str(modelName{modelId}),'.fig'])

end

restoredefaultpath
A.4 Obtain parallel tuning parameters

A.4.1 Main file

% Script calculating tuning parameters (parallel form) for PO(PID), PO(PI), SIMC–PID and SIMC–PI
% Written by: Martin S. Foss, fall 2012

% clc
clear all

% Adding "sharedFiles" to MatLab search directory
curDir = pwd;
mainDir = fileparts(curDir);
sharedDir = fullfile(mainDir,'sharedFiles');
addpath(sharedDir);

%%
for m = 1:9
modelId = 9; % m

fprintf('Finding tuning parameters (parallel form) \n')
fprintf('Case: %g\n',modelId)
fprintf('***************************\n')
fprintf(\n')

msSet = 1.7; % define Ms value

modelName = {'case1','case2','case3','case4','case5','case6','case7','case8','case9'};

load(fullfile(mainDir,'dataFiles',['resSimc_',num2str(modelName{modelId}),'.mat'])) % loading data files

% Collecting PO and SIMC tunings
index = find(res.POpid.minTuning.ms == msSet);
tuning{1} = res.POpid.minTuning.tuning(:,index(1));
index2 = find(res.POpi.minTuning.ms == msSet);
tuning(2) = res.POpi.minTuning.tuning(:,index2(1));

index3 = find(res.simcPID.simcMs <= msSet);
tuning(3) = res.simcPID.simcTuning(:,index3(1));

index4 = find(res.simcPI.simcMs <= msSet);
tuning(4) = res.simcPI.simcTuning(:,index4(1));

x = [1 2 3 4];

tuningParallel = []; %setting up result matrix
for i=1:length(tuning)
    Kc = tuning{i}(1); %finding Kc
    taui = Kc/tuning(i)(2); %calculating tau_i
    if i == 2 || i == 4 %calculating tau_d
        taud = NaN;
    else
        taud = tuning{i}(3)/Kc;
    end
    tuningParallel(:,i) = [Kc; taui; taud]; %storing results
end
res.tuningPara = tuningParallel;

%% Saving results
save([mainDir, '\dataFiles\','resTunPar_','num2str(modelName{modelId}),...
'.mat'],$,'res');
save(['tuningResults\','resTunPar_','num2str(modelName{modelId}),...
'.mat'],$,'res');
% end

restoredefaultpath
A.5  Shared files

The following m.files are used of several of the other files.

A.5.1  Cases - second order models

% Function calculating second order transfer functions for the different % cases % Written by: Martin S. Foss, fall 2012
function gp = model(caseNumber)
s=tf('s');
tau1 = 1;
switch caseNumber
  case 1
    tau2 = 0.5*tau1;
    theta = 1;
  case 2
    tau2 = 0.8*tau1;
    theta = 1;
  case 3
    tau2 = 0.3*tau1;
    theta = 1;
  case 4
    tau2 = 0.5*tau1;
    theta = tau2/1.5;
  case 5
    tau2 = 0.8*tau1;
    theta = tau2/1.5;
  case 6
    tau2 = 0.3*tau1;
A.5 Shared files

MATLAB SCRIPTS

A-32

theta = tau2/1.5;

34 case 7
35 tau2 = 0.5*tau1;
36 theta = tau2/2;
37
38 case 8
39 tau2 = 0.8*tau1;
40 theta = tau2/2;
41
42 case 9
43 tau2 = 0.3*tau1;
44 theta = tau2/2;
45 end
46
47 gp = 1/((tau1*s+1)*(tau2*s+1));
48 gp.outputd = theta;

A.5.2 Cases - first order models

% Function calculating first order transfer functions, using the half rule,
% for the different cases
% Written by: Martin S. Foss, fall 2012
function gp = modelPI(caseNumber)
s=tf('s');
tau1 = 1;

switch caseNumber
    case 1
        tau2 = 0.5*tau1;
        theta = 1;
    case 2
        tau2 = 0.8*tau1;
        theta = 1;
    case 3
        tau2 = 0.3*tau1;
        theta = 1;
end
tau2 = 0.3*tau1;
theta = 1;

\textbf{case 4}
\begin{align*}
\text{tau2} &= 0.5\times\text{tau1}; \\
\text{theta} &= \text{tau2}/1.5;
\end{align*}

\textbf{case 5}
\begin{align*}
\text{tau2} &= 0.8\times\text{tau1}; \\
\text{theta} &= \text{tau2}/1.5;
\end{align*}

\textbf{case 6}
\begin{align*}
\text{tau2} &= 0.3\times\text{tau1}; \\
\text{theta} &= \text{tau2}/1.5;
\end{align*}

\textbf{case 7}
\begin{align*}
\text{tau2} &= 0.5\times\text{tau1}; \\
\text{theta} &= \text{tau2}/2;
\end{align*}

\textbf{case 8}
\begin{align*}
\text{tau2} &= 0.8\times\text{tau1}; \\
\text{theta} &= \text{tau2}/2;
\end{align*}

\textbf{case 9}
\begin{align*}
\text{tau2} &= 0.3\times\text{tau1}; \\
\text{theta} &= \text{tau2}/2;
\end{align*}
\textbf{end}

\text{s1} = s + 0.5*tau2;
\text{Theta} = \text{theta} + 0.5*tau2;

gp = 1/(s1*(s+1));
gp.outputd = Theta;

\textbf{A.5.3 Controller}

\textit{function for generating the controller transfer function}
\textit{on the form gc = K + I/s}
A.5 Shared files

% Written by: Martin S. Foss, fall 2012

function [gc gcPI varargout] = controller(K,I,D,taud,varargin)

s = tf('s');

if nargin == 3
    gc = K + I/s + D*s; %PID Controller
elseif nargin == 2
    gcPI = K + I/s; %PI controller
    gc = NaN;
else
    gc = K + I/s + (D*s)/(0.01*taud*s+1); %PID controller with
    %derivative filter
end
return

A.5.4 Integral absolute error

function iae = iae(t,y)
% This is the function iae.m
% Simple integration routine made for computing IAE of time signal using
% trapez integration
% y - time signal vector
% t - time signal vector

% Initialize
i = 1;
npoints = length(t);
iae = zeros(1,size(y,2));

% Integrate
while i < npoints,
    yavg = (y(i,:) + y(i+1,:))/2;
    dt = t(i+1) - t(i);
    int = abs(yavg)*dt;
    iae = iae + int;
    i = i + 1;
end
% Function for calculating the Ms value
% written by: Chriss Grimholt 11 jan. 2012
function ms = ms(gp, gc)
ms = max(abs(freqresp(feedback(1,gp*gc),logspace(-4,4,40000))));
return
B SIMULINK Model

Figure B.1: SIMULINK model