Relay Auto-tuners in Modelica

Markus Björk

Robin Levenhammar

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6032
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2017

Abstract

The PID controller is by far the most popular controller for process control in industry today [Åström and Murray, 2008]. The combination of its relatively simple usage and its satisfactory results within several areas has gained its popularity [Visioli, 2006]. However, the tuning of the controller is crucial. An automatic tuning method was developed by [Åström and Hägglund, 1984] in the mid-eighties and auto-tuners has since then been diligently used in the process industry.

This thesis will focus on the implementation of an auto-tuner based on asymmetric relay feedback in the modeling language Modelica. Also, creating a good work flow and providing valuable information for the user. The auto-tuner provides an estimated low order model together with parameters for a PID controller, based on the AMIGO tuning rules.

After an introduction the report begins with a chapter containing a theoretical background concerning the PID controller, tuning methods, asymmetric relay autotuners and Modelica/Dymola. This is followed by an explanation of the method used, including a step-by-step description.

The report ends with chapters dealing with results, discussions and comments about future development. The result chapter is divided into different sections, each dealing with one of the processes used for testing. Every section presents the experimental results together with comparisons against other parameter set-ups. The last chapter contains a discussion regarding the overall results, both in terms of usage and performance.

The results, in terms of responses to load disturbances and step changes, were satisfactory for all processes that were tested in the thesis, although some processes required more preparatory work than others. As long as the process is in steady-state when the experiment begins, the resulting PID parameters turns out to work very good in comparison to the ones being used in the models today. Despite the good performance there are smaller things that needs to be further developed. For instance, a feature that is supposed to generate two plots automatically needs to be improved.

Acknowledgements

To begin with we would like to thank Modelon AB and our supervisor there, Håkan Runvik, for offering the opportunity to write this thesis for them. Håkan has been very helpful and easily accessible during the whole project, which has made things easier for us when struggling. For this we are grateful. The company offered us a course in Modelica which did help us in order to get going with it, therefore we would like to express our gratitude both to the company and the two lecturers Mathias Strandberg and Reza Dadfar.

We would also like to thank our supervisor, Josefin Berner, from the Department of Automatic Control at Lund University. Her willingness to help and answer questions regarding the method and implementation of it has without any doubt contributed in facilitating the work.

Finally we would like to thank Tore Hägglund for accepting the role as examiner.

Terminology

In this section frequently used expressions are stated, divided into sections.

Controller

- K: Proportional gain in a PID controller.
- T_i: Integral time constant in a PID controller.
- T_d: Derivative time constant in a PID controller.

Experiment

- d_1, d_2: Asymmetric relay levels.
- ε: The tolerance regarding the limit cycle.
- h: Hysteresis level.
- γ: A measure of the asymmetric level regarding the relay.
- ρ: Half period ratio used when calculating the normalized time constant.
- t_{on}: The time for which the relay output is equal to its maximum.
- t_{off}: The time for which the relay output is equal to its minimum.
- τ: Normalized time delay.

Models

- FOTD: First Order Time Delay.
- ITD: Integrated Time Delay.
- K_p: Process gain for a FOTD model.
- k_v: Process gain for an ITD model.
- L: Process time delay.
- T: Time constant.

Signals

- e: Control error.
- u: Relay output/control signal.
- u_0: Relay output when in steady-state.
- y: Process output.
- y_0: Process output when in steady-state.

Specification

- IAE: integrated absolute error.
- PO: Percentage overshoot.
- RT: Rise time.
- k_c: Critical/ultimate gain
- M_s: Maximum sensitivity
- t_c: Critical time period.

Contents

1

Introduction

1.1 Motivation

Modelica and the libraries within it are used by many people. People with varying background. Since the libraries contain blocks for several different purposes it is not always obvious how to use them. Control loops are often included when developing models, making blocks regarding control theory required. A person with lack of experience in this field will probably find it hard to use these blocks properly. One way to make it easier is to make use of an auto-tuner. With such a component a person with lack of knowledge in control theory still would be able to make use of it.

1.2 Scientific background

This thesis is based on an amount of references. The key references are mainly from [Berner, 2015], which is a licentiate thesis with focus on an automatic tuning method for PID controllers. The auto-tuner should be implemented in the modeling language Modelica, using Dymola as development environment and simulation tool. Modelica is a powerful modeling language where components from libraries can be used in solutions of varying kind [Modelica-association, 2014]. One can find a briefing concerning this later in the thesis. The company Modelon is working with this language and therefore a successful thesis resulting in a proper working method for auto-tuning would be appreciated. Knowledge regarding Dymola and Modelica has been obtained by attending a course and via its associated material, which was given by the company.

1.3 Contribution to the development of knowledge

In industry today auto-tuners are already used, including relay auto-tuners [Åström and Hägglund, 1984]. What this thesis mainly is aiming to contribute with is a helpful tool such that the auto-tuner can be used in a simulation environment. Since

the number of control loops can vary it is necessary to investigate if the experiment is able to perform regardless of this fact or if there is something that needs to be modified.

Another important goal with the tool is regarding its usability. Regardless if it is a person with many years of experience or a person with limited it should not be any problems to use it. This means that the graphic interface and the number of options are critical parts as well.

1.4 Goals

There are mainly three goals of this master thesis work. These are listed below.

- Implementation: An implementation of an auto-tuner should be done in Modelica based on [Berner, 2015].
- Workflow: The developed auto-tuner should make it easy for the user to work with PID controllers in Modelica regardless of his or hers previous knowledge in control theory.
- Evaluation: The auto-tuner should be evaluated in terms of performance through experiments on already existing models. Furthermore, the resulting PID parameters from the auto-tuner should be compared against other parameter set-ups such as the ones being used in the models today.

1.5 Delimitations

The experiment from the auto-tuner gives an estimated model of the process. Based on that model PID parameters can be determined. How they are obtained depends on what tuning rule one chooses to use. The auto-tuner block is restricted to one tuning rule, i.e. AMIGO tuning [Åström and Hägglund, 2006].

Another delimitation is regarding the models that are going to be used together with the auto-tuner. There will be no focus on developing new models since the libraries already contains appropriate models in order to evaluate the auto-tuner.

2

Background

The following chapter will introduce a number of topics which the thesis is built upon. The basics on PID control and its features are presented in Section 2.1. This is followed by 2.2 which covers the theory behind relay auto-tuners and the asymmetric relay feedback experiment. Lastly the software that has been central in this thesis work, Dymola and Modelica, is declared in Section 2.3.

2.1 PID control

Introduction

The by far most popular controller regarding process control in industry today is the PID controller [Åström and Murray, 2008]. The combination of its relatively simple usage and its satisfactory results within several areas has gained its popularity [Visioli, 2006]. This section will roughly explain the basics regarding the PID controller and its structure. A PID controller consists of three different control actions. In equation (2.1) the control signal, $u(t)$, is expressed as the sum of those three.

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{de(t)}{dt}\right) \tag{2.1}$$

$$1 : Ke(t) \qquad 2 : \frac{K}{T_i}\int_0^t e(\tau)\,d\tau \qquad 3 : KT_d\frac{de(t)}{dt}$$

The first part is the proportional control action and it is simply the current control error, $e(t)$, multiplied with a proportional gain, K. Next there is an integral action which is computed by multiplying the integral of the control error with a constant, $\frac{K}{T_i}$, where T_i is the integral time. Lastly there is a derivative action which is equal to the derivative of the control error multiplied with both the proportional gain, K, and the derivative time constant, T_d [Visioli, 2006].

Depending on what type of process one wish to control, the PID controller needs to be tuned in a proper way in order to get satisfactory performance. The three control parameters that are available for tuning are K, T_i and T_d.

Control actions

As one can expect the different control actions come with different impacts. Starting with a pure P controller, i.e. $T_i = \infty$ and $T_d = 0$, it will most likely end up having a steady-state error being not equal to zero. An increase of K results in decrease of the steady-state error. However, the increase will eventually make the system behave more and more oscillative when a certain limit is reached. Another way to get rid of the steady-state error is to make use of integral action which guarantees that it is zero when steady-state is reached. Depending on the value of T_i the error term will converge to zero with various speed. In terms of equation (2.1) one can easily conclude that a decrease of T_i implies a larger impact of the integral action. This in turn will speed up the convergence of the error. Just like the case with the proportional gain the adjustments of T_i has its restrictions. Decreasing the integral time too much will introduce undesirable oscillative behavior [Åström and Murray, 2008].

The derivative action can be used to damp oscillations due to large and small values on K and T_i respectively. The damping effect increases when increasing T_d. At some point though, it reaches a maximum and further increase of T_d will instead result in smaller and smaller effect. A large value on the derivative time may also introduce oscillations [Åström and Murray, 2008]. In Figure 2.1 three different simulations illustrate how tuning of the control parameters can affect the result.

Nonlinearities in the system such as actuators of various kind may cause problems when integral action is included. It is very likely that actuators come with limitations, e.g. a valve cannot be opened more than 100%. If the control error is large the calculated control signal may exceed limitations regarding the actuator. This in turn can put the actuator into saturation. If the calculated control signal becomes very large, i.e. much larger than the limit, it will take some time for it to return to a value that doesn't cause saturation. This phenomena is called integrator windup and it is avoided by using a method called anti-windup [Åström and Hägglund, 2006].

For high-frequency signals the derivative gain is ideally very large [Åström and Hägglund, 2006]. For this reason measurement noise tends to be amplified. If one consider a sinusoidal noise signal the conclusion that the impact from the amplification grows with higher frequencies can be drawn [Visioli, 2006]. To reduce the impact from measurement noise the derivative part can be implemented together with a filter whose task is to limit the amplification of content with high-frequency in the signal [Åström and Hägglund, 2006].

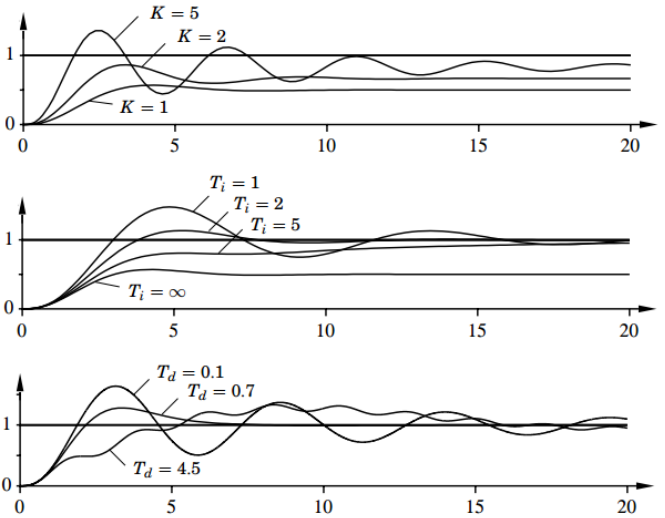

Figure 2.1: Three simulations of a closed-loop system. The upper plot illustrates the influence of K when working with a P controller. The middle plot shows the impact of T_i when $K = 1$ (PI control). In the lower plot the damping effect with varying values on T_d is shown. Here $K = 3$ and $T_i = 2$ (PID control) [Åström and Hägglund, 2006].

Tuning

The tuning part is crucial for design of PID controllers. A good guidance is available through the control specifications. Given those and since it is clear what impact the three control actions have on the control performance, one tuning method operators can make use of is the one called trial and error. Although it may sound easy it can be very tricky to come up with the optimal parameters and therefore this part really run the risk of being very time consuming. The result is understandably dependent on the operator in terms of how skilled he or she is [Visioli, 2006].

The fact that the tuning task only consists of three parameters has resulted in empirical techniques for direct adjustment of them. In total there are many different methods. Three of them are presented below, where the first two are the primary ones for this thesis.

Ziegler-Nichols method In 1942 two control engineers named John G. Ziegler and Nathaniel B. Nichols came up with two methods for determining the parameters for a PID controller. The methods take into account some features from process dynamics and simple formulas are used to calculate the parameters.

One of the two methods uses process information in terms of the open-loop step response to determine two parameters which together with an existing table gives the controller parameters (see Table 6.2 in appendix).

The other method uses the Nyquist curve of the process transfer function instead of the step response in order to get the necessary information about the process. Based on where the curve intersects with the negative real axis the ultimate gain and period is determined. The controller parameters are then calculated in the same manner as the previous method, using a given table [Ziegler and Nichols, 1942].

Although the Ziegler-Nichols methods have been actively used in industry for over 70 years regarding the tuning issue of PID controllers, it does come with severe drawbacks. The methods use deficient process information and the resulting closed loop systems are not good in terms of robustness [Åström and Hägglund, 2004].

AMIGO tuning (Approximate MIGO design) In [Åström and Hägglund, 2006] an alternative method is presented. The method is based on an open-loop step response, and takes a constraint regarding the robustness into account. They successfully came up with a simple tuning rule, named AMIGO, by investigating correlations between controller parameters and normalized process parameters, where the controller parameters were obtained by applying the MIGO design. MIGO stands for M-constrained Integral Gain Optimization and the primary goal with the design is to end up with good load disturbance responses. In order to fulfill the goal the main focus is to minimize the integrated control error. To guarantee robustness it uses a constraint on the maximum sensitivity, M_s [Panagopoulos et al., 2002].

λ-tuning Another method commonly used in process industry is the so called λ-tuning, developed in 1968 [Åström and Hägglund, 2006]. The method is a special case of pole placement and it is related to internal model control, IMC [Rivera et al., 1986]. By assuming a first order time delayed model, FOTD, it is possible to come up with either PI or PID controllers. If the design parameters are chosen in a good way the method can give appropriate results although the consequence of cancelling a process pole implies poor load disturbance responses for lag dominated processes [Åström and Hägglund, 2006].

2.2 Relay auto-tuners and the asymmetric relay feedback experiment

Development perspective

Auto-tuners in general have existed for many years. In the mid-eighties an effective way of tuning PID controllers by estimating the critical gain and the critical frequency on the Nyquist curve was found by [Åström and Hägglund, 1984]. Tuning PID controllers using the information about the critical gain and the critical time period was introduced by Ziegler and Nichols in the early forties. They proposed to find the critical gain through increasing the proportional gain in a simple proportional controller until oscillation occured. The gain that creates the oscillation is then called the critical gain k_c and the time period for the oscillation is called the critical

time period t_c, which also corresponds to a point in the frequency plane where the Nyquist curve intersects the negative real axis [Ziegler and Nichols, 1942].

Another way to determine k_c and t_c were found by [Åström and Hägglund, 1984]. Instead of iteratively increasing the proportional gain in a PID controller they applied a relay and the describing function. The fact that a relay in a closed loop together with a system having a phase lag of at least π radians may create an oscillation with time period t_c is stated in [Åström and Hägglund, 1984]. The critical gain k_c could then be determined by the describing function approximation.

Since that method only gives one point on the Nyquist curve, an improvement of that method has been provided. The idea of the improved relay auto-tuner experiment is instead to find a low order model describing the process. To reach that goal the idea is to take advantage of the normalized time delay, which is to be described later in this text. The relay experiment is done, like in the [Åström and Hägglund, 1984] experiment, by placing a relay in the closed loop. It can be represented with a block diagram with the relay parallel to an off-switched controller, as in Figure 2.2. Since it is placed in a closed loop it has the advantage to stay nearby the set point. During the experiment the switch is adjusted such that the relay is active, otherwise it is the controller that is in charge.

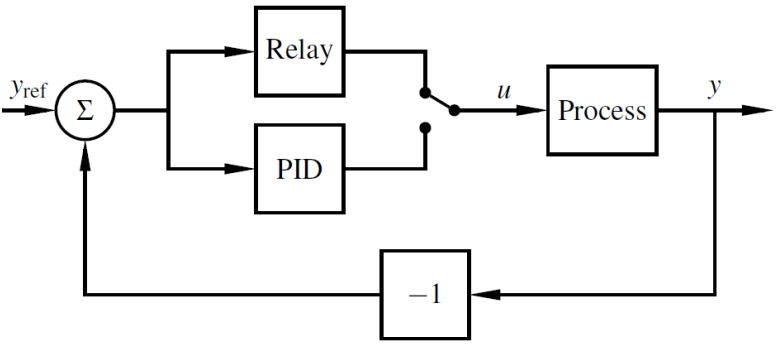

Figure 2.2: Block diagram with the relay implemented in the feedback loop [Berner, 2015].

The main idea

[Åström and Hägglund, 1984] used a symmetric relay, in which the upper and lower levels have the same magnitude regarding a given zero point u_0. The method presented here uses an asymmetric relay. The advantage with the asymmetric relay is that it provides a low order model. In [Berner et al., 2014] the two levels are defined as d_1 and d_2 and they are illustrated together with a hysteresis added in Figure 2.3.

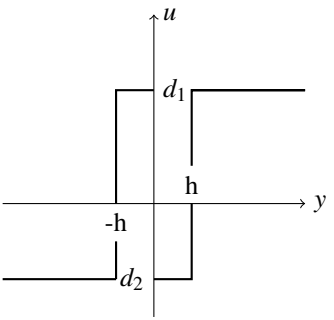

Figure 2.3: Asymmetric relay with hysteresis h and the amplitude levels d_1 and d_2.

When the experiment starts the process is assumed to be in steady-state. In other words, that the control signal and the output are staying at a certain working point $(u,y) = (u_0, y_0)$. When in steady-state, the goal is to make the process oscillate in a limit cycle behavior, like in [Åström and Hägglund, 1984], whereupon certain features can be investigated. This is done by generating a control signal from the relay, which according to [Åström and Hägglund, 1984] will make a process with a phase lag of at least π to oscillate with a time period t_c. Since it is a relay the signal will either be u_{on} or u_{off}, nothing in between. Every time the output signal passes through its equilibrium the control signal is switched, either from u_{on} to u_{off} or vice versa. Since that would introduce a heavy switching behavior in environments where noise is present a hysteresis belt with deviation h from y_0 is introduced as shown in Figure 2.3 and illustrated in Figure 2.4. With a hysteresis belt implemented the relay will not be as sensitive to noise as it would be without it. The upper and lower limits of the hysteresis band together works as a substitute to the working point y_0 when it comes to the switching behavior regarding the relay.

Estimated models

In [Berner, 2015] there are two types of representations describing a first order process. The first order time delayed model (FOTD), which has the transfer function represented as given in equation (2.2).

$$P(s) = \frac{K_p}{sT+1} e^{-Ls} \tag{2.2}$$

The other parametrization is the integrated time delayed model (ITD), represented in equation (2.3).

$$P(s) = \frac{k_v}{s} e^{-Ls} \tag{2.3}$$

Estimates of the model parameters (k_v, K_p, L, T) will be obtained from the relay feedback experiment. In the case with the FOTD model the parameters L and T

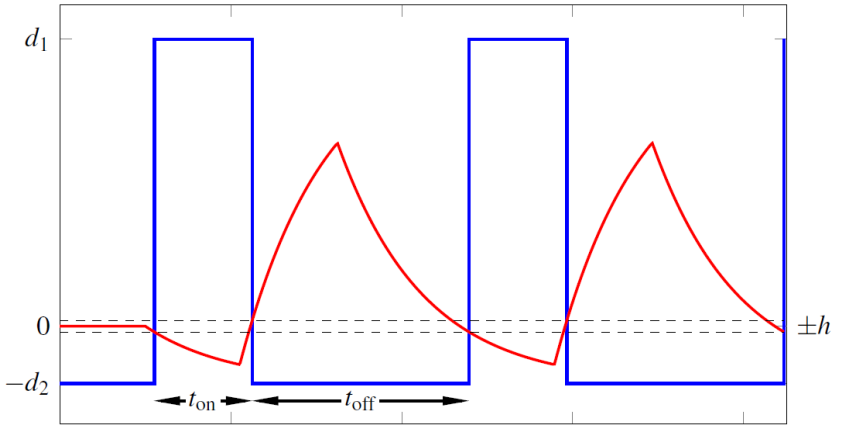

Figure 2.4: The relay experiment with a hysteresis belt with height h introduced [Berner, 2015].

relate to each other by a normalized time delay which is given by:

$$\tau = \frac{L}{L+T}. \tag{2.4}$$

Depending on the value that τ is assigned, a first order model can either be described as a FOTD or an ITD. From equation (2.4) one can see that T has less impact on τ if the value is close to one and vice verse. In [Luyben, 2001] the ratio between L and T was investigated. The curvature of the output signal was shown to behave more as a pure integrator if the ratio was small enough, i.e. the output signal behaved as a triangular wave. Such a wave can be interpreted as a ramp function for small values on the ratio $\frac{L}{T}$. A series of pulses (steps) into an integrator process will result in a series of triangular pulses, which is the reason why it can be approximated as an ITD model, if τ is small enough. If the ratio was large the behavior looked like a delayed pulse. To find the right model curve fitting is used, which is based on the value of the normalized time delay.

The experiment - a walkthrough

The first step in the experiment is to figure out the sign of the process and also the initial amplitude for the relay. The relay signal is ramped up exponentially until one of two possible events occur. Either the maximum value of the control signal is reached or the output signal hits one of the limits on the hysteresis belt. If the first scenario occurs the relay output stays constant until one of the limits is crossed. Depending on if it is the upper or lower hysteresis level that has been crossed the sign of the process gain either is stated as positive or negative. At this point the initial values of the relay levels d_1 and d_2 are determined using the current value

of the relay output and an asymmetry factor γ, which is defined in equation (2.7). Depending on the value of u_0 the two relay outputs u_{on} and u_{off} can be determined as follows:

$$u_{on} = u_0 + \text{sign}(K_p)d_1 \tag{2.5}$$

$$u_{off} = u_0 - \text{sign}(K_p)d_2. \tag{2.6}$$

Observe from equation (2.5) and (2.6) that u_{on} either can be the higher or the lower level depending on the sign of K_p. The same reasoning can be applied for u_{off}. It turns out that the ratio between the maximum and the minimum value of d_1 and d_2 (mathematically described in equation (2.7)), referred to as asymmetric level γ, has an impact on the estimated process gain [Berner, 2015].

$$\gamma = \frac{\max(d_1, d_2)}{\min(d_1, d_2)} \tag{2.7}$$

A larger value on γ will decrease uncertainties on the estimated process gain. At the same time there is a constraint on γ which has to be fulfilled:

$$\gamma \leq \frac{y_{maxdev}}{y_{mindev}}. \tag{2.8}$$

Where y_{maxdev} and y_{mindev} are the maximum and minimum deviations from the working point y_0 respectively. Those values represents levels which the output signal should lie in between. Since the requirement stated in equation (2.8) has to be met an arbitrary value on γ cannot be chosen.

In order to make the oscillation of the output signal stay in a preferable distance from the working point ($y_{mindev} \leq y \leq y_{maxdev}$) the relay amplitudes have to be adjusted. It is necessary that the oscillation stays above the hysteresis level. If equation (2.9) is fulfilled

$$min(d_1, d_2) \mid K_p \mid \geq \mu h, \tag{2.9}$$

for some value of $\mu > 1$, it can be guaranteed that the oscillations stays above the hysteresis level. Furthermore, it has been shown in [Berner, 2015] that small values on μ give less accurate estimation on the normalized time delay than larger values on μ. Since τ is an important parameter in the experiment it would not be desirable to accumulate an error in that variable.

The experiment continues until the process output signal reaches a limit cycle. Mathematically a limit cycle is defined as

$$x(t + t_0) = x(t), \text{ for some } t_0 > 0. \tag{2.10}$$

If two consecutive periods are sufficiently equal one can consider that the output signal has reached a limit cycle. To check whether they are approximately equal a tolerance level is introduced, specified by the parameter ε. The tolerance is typically very small and can be stated mathematically as:

$$\left| \frac{t_p - t_p^*}{t_p^*} \right| \leq \varepsilon, \tag{2.11}$$

where t_p and t_p^* is two consecutive period times. The tolerance ε is a user-defined parameter. A bad choice may accumulate error on the experimental result. This uncertainty has been investigated in [Berner, 2015]. The most crucial parameter in the FOTD model is the process gain K_p. It was shown that smaller values on ε gave smaller uncertainties in the experimental process gain. But there is a drawback with decreasing the tolerance to much since it will increase the experimental time. This means there is a trade-off one have to deal with regarding the two parameters. When simulating a process the experiment time is often not as crucial as it is in industry. However, the other parameter τ has been shown to stay almost independent for different values on ε. The default value was therefore choosen to 0.01, also listed in appendix, Table 6.8.

When the experiment has converged it is possible to gather data to analyze how long the relay has been on and how long it has been off. The variables regarding those times are t_{on} and t_{off} (Figure 2.5). They are necessary for later parameter calculations when dealing with different models. Another parameter that is available in the experiment is the integral of the output, i.e.

$$I_y = \int_t^{t+t_p} (y(\tau) - y_0)\, \mathrm{d}\tau, \tag{2.12}$$

where t_p has the duration $t_p = t_{on} + t_{off}$ [Berner et al., 2016b]. An illustration of the integral is shown in Figure 2.5. The output of the relay can be calculated in the same way. Since u is a square wave (see Figure 2.5), the integral calculation can be simplified as

$$I_u = \int_t^{t+t_p} (u(\tau) - u_0)\, \mathrm{d}\tau = (u_{on} - u_0)t_{on} + (u_{off} - u_0)t_{off} \tag{2.13}$$

The parameters in equation (2.2) and (2.3), i.e. k_v or K_p, L and T (FOTD), may be calculated through the results from the experiment. As mentioned earlier the value of τ decides what model is to be chosen. Since τ is unknown it has to be estimated. How to estimate it is shown in [Berner, 2015]. The half-period ratio, ρ, is used for that purpose. It is defined as

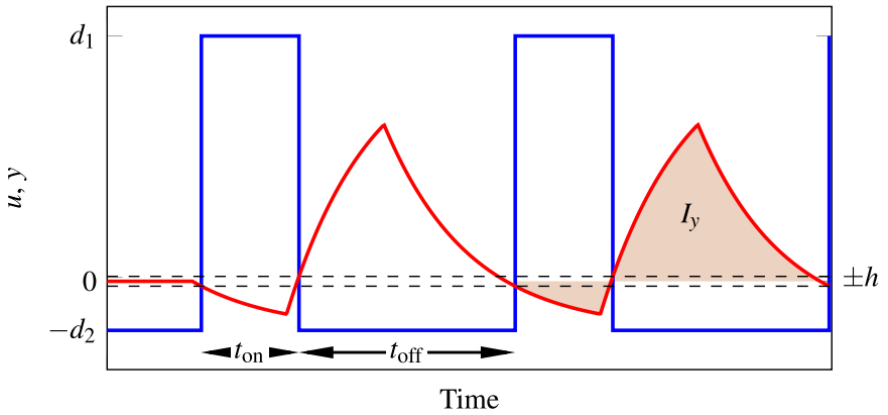

Figure 2.5: Illustration of the integral of the process output I_y. Here the red curve is the process output and the shaded area highlights the integral over one period of time [Berner et al., 2016a].

$$\rho = \frac{\max(t_{on}, t_{off})}{\min(t_{on}, t_{off})}, \tag{2.14}$$

which was shown in [Berner, 2015] to have an impact on τ through

$$\tau(\rho, \gamma) = \frac{\gamma - \rho}{(\gamma - 1)(0.35\rho + 0.65)}. \tag{2.15}$$

Since τ is estimated it will come with an error. However the estimated value is good enough to use for the calculations. If one wishes to read more on how to estimate τ, see [Berner, 2015].

Model choice

Earlier in this chapter it was discussed how the value of τ decided what model to choose [Luyben, 2001]. The unknown parameters in the models are determined in different ways. A large value (close to 1) on τ was said to indicate that an FOTD model is the best choice whereas an ITD model is the proper choice for small values of τ. In [Berner, 2015] the parameters for an FOTD model are given by the following equations:

$$K_p = \frac{I_y}{I_u} = \frac{\int_t^{t+t_p} y(\tau) - y_0 \, \mathrm{d}\tau}{(u_{on} - u_0)t_{on} + (u_{off} - u_0)t_{off}}, \tag{2.16}$$

$$T = \frac{t_{on}}{\ln(\frac{h/|K_p| - d_2 + e^{L/T}(d_1+d_2)}{d_1 - h/|K_p|})}, \tag{2.17}$$

$$L = T\frac{\tau}{1-\tau}, \tag{2.18}$$

and for an ITD model they are given by:

$$k_v = \frac{2I_y}{t_{off}t_{on}((u_{off}-u_0)+(u_{on}-u_0))} + \frac{2h}{t_{on}(u_{on}-u_0)}, \tag{2.19}$$

$$L = \frac{(u_{on}-u_0)t_{on} - 2h/k_v}{(u_{on}-u_0)-(u_{off}-u_0)}. \tag{2.20}$$

Observe the benefit with an asymmetric relay by looking at the derivation of K_p in equation (2.16). That derivation would not be possible for a symmetric relay, since it will always be zero. However, the integral I_u can be zero for an asymmetric relay as well. That requires that the ratio u_{on}/u_{off} is equal to the ratio $-t_{off}/t_{on}$. This implies that ρ and γ are equal according to their definition in equation (2.7) and (2.14). If ρ and γ are equal equation (2.15) would be zero, meaning that the process is of integrator type. In this case equation (2.19) and (2.20) should be used [Berner et al., 2016b].

To view the detailed derivations regarding the model parameters in equation (2.2) and (2.3) one can study [Berner, 2015]. The resulting values from equation (2.16), (2.17), (2.18) or from (2.19) and (2.20) can be used to tune the parameters in the controller using any preferable method. AMIGO tuning is the method that is to be used in this thesis, which relates its choice of PID parameters to the given parameters in equation 2.2 and 2.3. Another choice could have been λ-tuning for instance. In [Åström and Hägglund, 1984] k_c and t_c was given by the experiment, in that case a possible tuning method could be Ziegler and Nichols.

2.3 Modelica and Dymola

Modelica is an object-oriented modeling language that can either be programed using components from already existing Modelica libraries, or be coded in a text editor. It allows the user to integrate systems from many different territories, which makes the language useful for engineers from several areas [Modelica-association, 2014].

Dymola is a simulation and modeling tool developed to support the Modelica language. Modelica is powerful mainly because of its compatibility to integrate

modeling, analysis, simulation and results evaluation. Since it has open libraries users can easily create and develop own components or make use of existing components that may be of interest for a particular purpose.

This thesis work mainly uses Modelica to create a model for auto-tuning purpose. The resulting model will hopefully be a beneficial tool while working with tuning of PID controllers for different kinds of processes.

Model definition

A model is a type of a class in Modelica. When declaring a model there is a certain structure describing the necessary parts. To begin with all the variables and parameters that are to be used in the model should be declared. There is a concrete difference between the meaning of a parameter and a variable. A variable is allowed to change during a simulation whereas a parameter must be constant.

Next there is a part called equation. Here all the necessary equations regarding the model are stated. There are differences from Java, C or other programming languages when working with the equation part in Modelica. In Modelica it does not matter which side of the equality sign one puts the expression, i.e. the left hand side is not reserved to be the "assuming side". When defining an equation it is not defined which variable it is to be solved for. Lastly the order of the rows is irrelevant since everything is executed concurrently [Modelica-association, 2014].

As an alternative to the equation part one can introduce an algorithm section, which just like an equation part relates variables. Depending on your model the algorithm section can either replace the equation completely or complement it. The way to work within this section is more like standard programming, both considering the sign and the row order. The latter entails that it is more convenient to use an algorithm if the purpose is to write a sequential working code. An algorithm section separates inputs from outputs, which is a big difference in comparison to the equation part [Modelica-association, 2014].

Events in Modelica

In general, people who work with Modelica does not think in terms of sequential execution. Instead, one creates the model and then includes necessary equations for solving the problem, with no respect to the order. Having that in mind the structure of the auto-tuner will be special since it will work sequentially right through. Considering Section 2.2, the different parts in the experiment comes with dependencies regarding earlier results. A suitable approach is to refer to the different experiment parts as states.

A transition from one state to another can be considered as a discrete behavior. In Modelica this is something that creates a discontinuity in the solution. Modelica treats such a behavior as an event. An event is generated via a conditional expression that is fulfilled when the total condition is triggered, e.g. when the process output crosses the upper hysteresis level while the relay output is equal to its higher

value. The condition can embrace all kinds of variable types, integer, boolean etc. If it is specific times that triggers the condition, i.e. if a variable containing a real value is compared with the built-in time variable, the event is called a "time event". Condition types of various kind appear in the work with the auto-tuner.

In Modelica events are treated with statements such as if- and when-statements. The main difference is that when-statements only are active when the event is triggered, compared to the if-statement that is active as long as the conditional expression is true [Modelica-association, 2014].

Linearization tool

Modelica LinearSystems 2 is a library that includes packages with a bunch of methods associated to linear analysis. This is for instance useful in cases when the model is non-linear in order to make a linearization.

In order for the linearization to be pleasant the process needs to be in steady-state. Because of the linearization one should be aware of that the model only is valid around the selected point.

The linearization tool will later be used in order to find a reference model. Together with that model one is able to find a controller that is comparable to the controller obtained by the auto-tuner.

3

Method

In this chapter the implementation of the relay auto-tuner in Modelica is considered. Also the workflow regarding the block is discussed. Several assumptions are being made along the way and they are primarily based on [Berner, 2015]. In case of exception the current reference is stated.

The first thing to consider is what information the relay block should exchange with the rest of the world. In Modelica this is called interface or boundary condition. Since the equations in Chapter 2 need information about the process output, y, it is reasonable to use this as an input to the block. The second signal used as an input is the set point, y_s. In this way it is possible for the user to try the experiment for different steady-state levels. The signal generated by the auto-tuner, u, is defined to be an output.

A relay block has been constructed mostly through the "Modelica text" environment, but also through the diagram view in Dymola. The two inputs and the output discussed above are defined to be of the type real. The auto-tuner uses information provided by the process and utilizes this to do the mathematical calculations given in Chapter 2. In Figure 3.1 one can see what the resulting block looks like in the diagram view in Dymola.

Figure 3.1: The auto-tuner block with two inputs and one output. The inputs to the block are the process output, y, and the set point y_s. The output is the output signal generated by the auto-tuner, u.

The structure of the implementation is categorized into different states by reason of the experiment structure. Depending on what state is active the output from the auto-tuner is different. For simplicity an enumeration is used in order to define

suitable states for the experiment. Another way to define the states could be by introducing an integer variable. Since the number of states is known beforehand an enumeration easily can be used. One advantage with enumerations in comparison to an integer variable is the fact that it is possible to make it more readable and understandable.

This chapter will cover what type of actions that is being exerted in the different states. Mainly there are six different states, as follows: *Initiation*, *ExponentialGrowth*, *RelayOutput_ON*, *RelayOutput_OFF*, *Go steady state* and *Comparison*, illustrated in Figure 3.2. The second and third state differ from the others since the experiment will be switching between those two for some time, whereas the other states only will be active once. When starting the experiment it enters the first state, *Initiation*. It will not change state until the process output reaches a steady-state. This is necessary since the experiment requires steady-state to be able to achieve good results. When reached steady-state next state can be activated, more particularly the state is changed to *ExponentialGrowth*. The output is increased exponentially until certain events occur upon which the sign of the process gain and initial relay amplitudes can be determined, as stated in Chapter 2. The upcoming two states, *RelayOutput_ON* and *RelayOutput_OFF*, are alternated a non-fixed number of times during the experiment. The number of switches can vary for different experiments and processes. It will continue to jump between these two until convergence occurs. The next state is *GoSteadyState*, which is used in purpose of reaching steady-state again. The last state named *Comparisons* handles a comparison between the estimated model and the actual process. An illustrative overview of the different states can be studied in Figure 3.2.

Transitions between states are treated with when-statements in Modelica since it only has to be updated once when it is time to change state. Regarding the applied control signal u this needs to be set continuously throughout the experiment. In this case if-statements are used.

3.1 How to reach steady-state and check whether it is reached

Before the main part of the experiment starts, i.e. before *ExponentialGrowth* is active, the process output has to be in steady-state. For some processes this may take more time than it does for others. In a mathematical point of view, steady-state requires no variations at all. Practically it is approximated to be satisfied when there are only small variations. Small oscillations around a steady point can have different meanings for various processes. For some processes it can signify that steady-state has been reached while for others it cannot. Therefore this has to be handled with caution, otherwise the whole experiment might present bad results. According to [Berner, 2015], the experiment will not give satisfactory results if the process is not in steady-state when it is started. Of course this is not a problem if the process is in

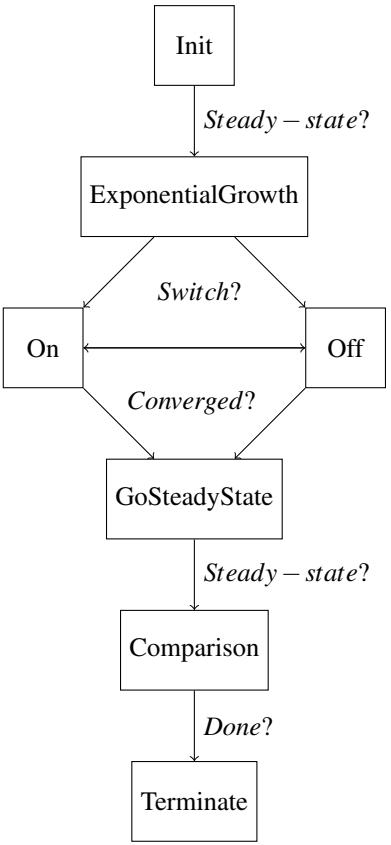

Figure 3.2: The relay may be in six different states. It starts by generating a constant or controlled signal. Then it continues in to exponential growth and from here there are two possibilities depending on which hysteresis level being crossed. The process is then alternated between the next two states (On/Off). When convergence is a fact, a control action makes the process output reach steady-state again whereupon the comparison can be made.

Input signal	Duration
Constant	Fixed
Controlled	Unfixed

Table 3.1: Describes the two main options concerning the steady-state issue.

steady-state initially. For instance, when dealing with a tank whose level should be controlled, it is necessary to bring the level to a desirable point beforehand.

Now when it is clear that steady-state is a prerequisite for the experiment, the next issue is regarding the knowledge of being in steady-state. Something needs to inform the auto-tuner that steady-state has been reached. One way to solve this is by guessing the time it might take. For instance one may believe that 30 seconds is enough. This method might work in some cases but in general it is very uncertain. A method more reliable would be to study the derivative of the process output. A small value indicates that the output is more or less constant.

The two methods mentioned above have been implemented in the auto-tuner making it possible for the user to decide what setting to use. Both methods are regarding the problem of knowing when steady-state is reached. In addition to this one can choose from two different ways of reaching the desired level. Either by applying a constant input or by using a PI controller (Constant or Controlled input in Table 3.1). The controller and the constant input that is used to reach steady-state uses the same output as the relay. This means that they are only active during the initiation. Another alternative would be to include a controller in the loop and let the relay work with the stabilized system. However, that is not the method that is used in this thesis.

Regardless of the choice for reaching steady-state one will need to come up with suitable values on the constant input or PI parameters. To determine those the user has to apply trial and error, where pre-existing knowledge of the process together with its complexity has an impact. In some cases Ziegler and Nichols tuning rules can be used with good results when deciding the PI parameters. To sum things up the user has the opportunity to choose from four different modes, see Table 3.1.

In Figure 3.3 one can have a look on a screen shot of the parameter window of the auto-tuner block. The choices are available through a drop-down list under the tab called initialization. As earlier mentioned the user can either choose a fixed time, t_{ss}, which determines when the program should go further and change state. The process output is then assumed to be in steady-state after this time. This setting requires some additional support from the user in order to set the fixed time to a reasonable value. This can be done by doing pre-simulations and study the steady-state behavior. This would of course not be possible in the industry. In that case an operator would instead have to wait for the process to reach steady-state and then push on the experiment start button. However, since the auto-tuner is used in a simulation environment this is not a problem and the method is therefore available

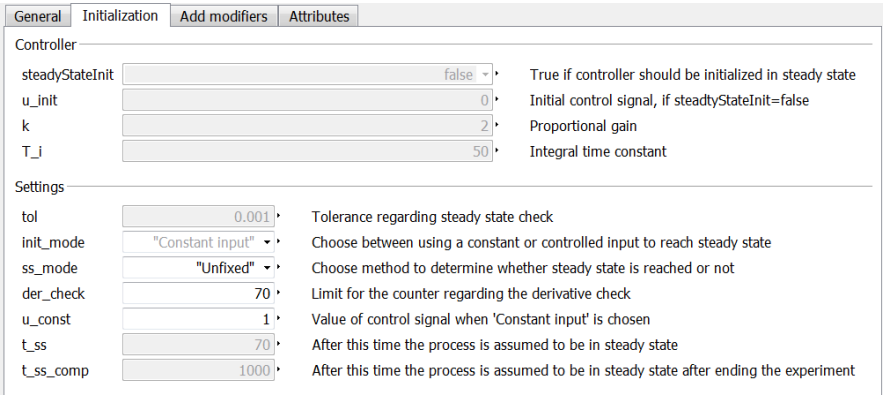

Figure 3.3: Initialization of the auto-tuner. it is divided into two separate groups, *Controller* and *Settings*. The *Controller* part treats the stabilizing controller if `"Controlled input"` is chosen. Under *Settings* it is possible to change what mode to use in order to reach steady-state etc. In appendix their default values are listed in Table 6.8

as an option.

The other alternative available from the drop-down list is the one based on the derivative. If the derivative is below a certain tolerance, implemented as tol, during a number of consecutive time steps the process is considered to be in steady-state. By default the test is executed each tenth of a second. For simplicity the derivative has been implemented as an approximation

$$\frac{dy}{dt} \approx \frac{\Delta y}{\Delta t}, \tag{3.1}$$

where Δt represents a small time step. Since it is possible to choose a small value on the time step it is not a disadvantage to choose this approximation. However, the method do come with some drawbacks regarding the accuracy. Since a curvature can consist of a number of extreme points it is important to make sure that the derivative is less than the chosen tolerance for a fair amount of time. Furthermore, problems may arise if the process consists of a long time delay, if the process has slow dynamics etc. It is possible for the user to choose how many times in a row the derivative check should be successful, observations from tests have shown that this clearly has an impact on the resulting performance.

3.2 Start-up

When steady-state is reached, an event is triggered making the state change from *Initiation* to *ExponentialGrowth*, see state diagram in Figure 3.2. The purpose of

the *ExponentialGrowth* state is to find the sign of the process gain and to decide the initial amplitudes of the relay. These are separately determined using two different functions. The first function uses the simulation time as input to exponentially increase the output signal from the relay. This is done until a certain event occurs. The event is triggered when the process output crosses one of the hysteresis levels, illustrated in Figure 3.5. The event is handled with a "when-statement" since it only needs to enter and update the state once. According to equation (2.5) and (2.6) the crossed level decides how the relay amplitudes should be designed. This is implemented in a second function named *RelayAmplitudes*. This function returns the initial asymmetric relay amplitudes (d_1 and d_2), the output signal of the relay (u_{on} and u_{off}) and the sign of the process gain. In order to avoid that the computed output exceeds or is below the allowed limits a saturation check is done. It is required that

$$u_{min} < u < u_{max}, \tag{3.2}$$

is satisfied to achieve desirable results. Values on u_{min} and u_{max} are set by the user through the auto-tuner parameter window illustrated in Figure 3.4. In the group called *Experiment parameters* one can also decide the asymmetric level γ, the hysteresis level h etc. Through observations a good value on h might be set a factor 100 less than the maximum value. When working in a simulation environment there is normally no noise to take into consideration, therefore it may work fine without a hysteresis as well. Without it however, processes with fast dynamics could have a negative impact on the result. To mimic the standard implementation found in [Berner, 2015], it is included in the implementation.

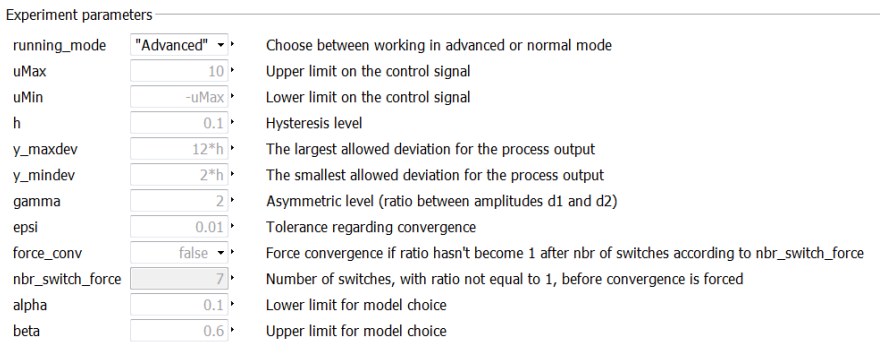

Figure 3.4: Experiment parameters that the user is able to change. In appendix their default values are listen in Table 6.8

Depending on if it is the upper or lower hysteresis level that is being crossed it is possible to determine the sign of the process gain. At the same time instant

the initial amplitudes regarding the relay are determined through equation (2.5) and (2.6). It has been implemented in such a way that d_1 always is γ times larger than d_2. The determined results are then used to set the output signal in the actual state.

With the sign of the process gain and the initial relay amplitudes given, the experiment moves on. By taking a look at the state diagram in Figure 3.2 one can see that there are two different alternative ways to go from *ExponentialGrowth*, either it is *RelayOutput_ON* or *RelayOutput_OFF*. It is the sign of the process gain that decides what the next state becomes. According to equation (2.5) and (2.6) a positive sign indicates that ON is the upper level and OFF is the lower level. The experiment then alternates between *RelayOutput_ON* and *RelayOutput_OFF*. The triggering occurs when the process output passes one of the hysteresis levels. For instance, if the sign is positive and the process output crosses the upper level (coming from below) the state should change to *RelayOutput_OFF*.

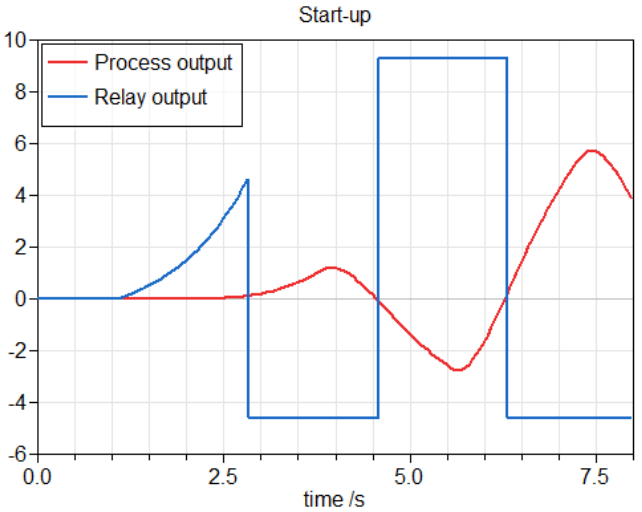

Figure 3.5: Illustration of the start-up in the experiment. The blue curve represents the output from the auto-tuner and the red the process output. The control signal increases exponentially to begin with. When the process output passes through the hysteresis band the relay amplitudes are set. In this case u_{on} is the upper level and u_{off} is the lower since the process output goes in positive direction.

3.3 Adjustment of the relay amplitudes

To get a more gentle oscillation on the process output the amplitudes are adjusted. This is done until the oscillation aligns within a desired amplitude interval. The

interval limits are set by y_{mindev} and y_{maxdev} and the values 2h and 12h respectively are used, as in [Berner, 2015]. As long as equation (2.8) is satisfied other suitable limits can be chosen.

The adjustments of the amplitudes are done with an algorithm which takes the process output during one time period into account. Depending on what value the largest deviation from y_0 assumes, the relay amplitudes are either increased, decreased or their values are retained. This procedure is done until the largest deviation during one time period ends up within the desired limits. To prevent the signal from decreasing to much, a lower limit on how much it can be scaled during each iteration is set.

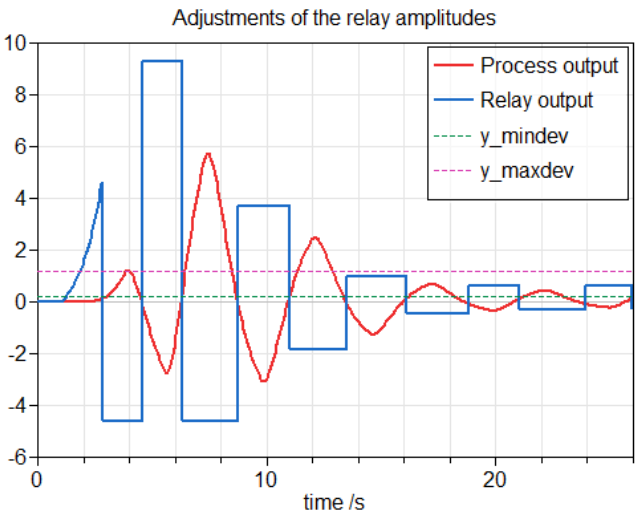

Figure 3.6: Illustration of the adjustment of the relay amplitudes in order to align within the preferable region. The region is set to be y_{mindev} (green line) and y_{maxdev} (magenta line). By default $y_{maxdev} = 12h$ and $y_{mindev} = 2h$.

As mentioned above the algorithm regarding the adjustments uses the largest deviation from y_0 during a time period. To be able to come up with this value the process output has to be compared with its previous value continuously throughout one period. The problem is solved by using an own type called `Comparator` constructed in C and appurtenant functions. In total there are four functions regarding `Comparator`. Firstly there is one named `createComparator()` which is called by the constructor in Modelica. When the constructor is called memory concerning one instance of `Comparator` is allocated and the internal variables are set to zero. Correspondingly, the memory allocated by that instance has to be deallocated. This is done by the function `deleteComparator(void* type)` which is called by the

destructor in Modelica.

In the equation part (Modelica text mode) the function `updateAndGetMax(void* type,double y)` is called continuously. It simply compares the current value on the process output, y, with an internally stored value representing the current maximum value. If the input y is larger than the stored value, the latter is updated and returned. Otherwise the stored value is returned.

Due to the fact that the adjustment algorithm uses values from separate time periods the stored max value needs to be reset right before it begins another loop. This is done by calling on the function `reset(void* type)`. Without the reset function the maximum deviation would not be matched correctly to the different periods. According to Figure 3.6 the process output in the beginning is much larger than in the end. If the stored max value is not reset in this case, the consequence is that the wrong value will be used in the algorithm.

3.4 Convergence check

If equation (2.11) is satisfied, the oscillation of the process output is assumed to have reached a limit cycle, i.e. two consecutive periods are sufficiently equal. To examine whether this is the case, two consecutive periods need to be calculated. On top of that an acceptable limit, i.e a tolerance, regarding the difference between the periods needs to be considered. In [Berner, 2015] the tolerance level is examined and it is shown that an appropriate choice on ε is 0.01. This choice is confirmed by studying the error in τ and process gain respectively. It proves that a smaller value on the tolerance does not improve the accuracy but instead increases the experiment time. Using a larger value on the tolerance decreases the experiment time but at the same time the resulting process gain is not as good as with ε equal to 0.01. Based on this, ε is defined to be 0.01 by default.

When measuring a time period the first thing one has to deal with is when to begin the measurement. In Modelica this is kind of tricky since parameter values are not accessible during a simulation in the same way as they are in Matlab for instance. There is no such memory feature. To solve this problem an appropriate way to go is to make use of the relay output and its switching behavior. The relay output is switched every time the process output crosses one of the hysteresis levels. The easiest way to see this is to take a look at Figure 2.4 in Chapter 2. When the relay (blue curve) is set to its lower level and the process output (red curve) crosses the lower hysteresis level the relay output is changed from low to high. In connection with this event the current time is stored in a vector with two slots, one intended for the start time and one for the end time regarding one period. The time is available through a global variable. Two separate vectors are used to store times, one is regarding switches from low to high and the other is the opposite case. The purpose of having two instead of one is to be able to compare two consecutive periods every half period. This feature may decrease the duration of the experiment. In Figure 3.7

an example is shown where one can see how the time periods change with time. In the end the difference between the two consecutive time periods is so small that convergence is assumed.

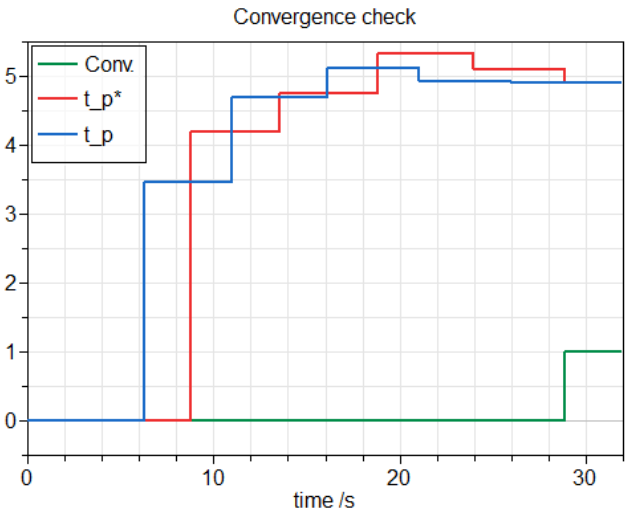

Figure 3.7: Illustrates the time period measurements used for the convergence check during an experiment. t_p and t_p^* are period times of two consecutive periods. When they are sufficiently equal, convergence is assumed. Just before 30 seconds this happens, see the green curve.

3.5 Measuring and derivation of t_{on}, t_{off}, I_u and I_y

When the process output has reached a limit cycle, i.e. when equation (2.10) and (2.11) are satisfied, convergence is assumed. At the same time some values are stored for further calculations. As mentioned in the previous section it is not possible to access complete data concerning variables in Modelica. For that reason the same idea as above is used when measuring t_{on} and t_{off}. Times connected to specific relay switches, e.g. low to high, are used to store start and end times regarding t_{on} and t_{off}. In order to minimize the duration time for the experiment the calculation of the respective times are done continuously, even though they come to use first at convergence.

Consider the situation shown in Figure 2.4, where the times t_{on} and t_{off} are specified. To begin with the relay output is changed from low to high. At this moment the current time is stored and used to describe two things. Firstly it represents the start time for a new measurement of t_{on}, secondly it represents the end time for

t_{off}. When the relay after some time returns from high to low the current time once again is stored and used for updating times regarding the calculation of t_{on} and t_{off}. In Figure 3.8 one can see the calculations of t_{on} and t_{off} respectively. In the end of the simulation they are held constant.

Observe that the relays *ON* state not necessarily is equivalent with having the relay output set to its upper level. The appearance depends on the sign of the process gain, as stated in equation (2.5) and (2.6).

In addition to the times discussed above two integrals are calculated, based on equation (2.12) and (2.13). Due to the same reasoning concerning the total duration time the calculation is done continuously throughout the experiment, until convergence occurs. Dealing with equation (2.12) and the fact that convergence may occur every half period it is required to have two separate integral calculations for the output, one being lagged by a half period with respect to the other. To calculate the integrals in Modelica one can take advantage of the built-in function `der()`. The usage is shown in equation (3.3).

$$der(I_y) = y, \tag{3.3}$$

Where I_y is the integral of the process output y [Modelica-association, 2014]. Modelica will give you two things depending on what variable that is known, the function solves the problem differently for different cases. Consider I_y to be known, then y is derived by taking the derivative of I_y. On the other hand, when y is known, Modelica solves the problem by taking the integral of y. In the experiment y is known and therefore the latter version is applied.

At the same instant as a new period is started the integral value must be reinitialized to be 0. In connection with this event the calculated value is stored in another variable just to make sure it is not overwritten. In Figure 3.9 one can see how the two integrals develop over time. Notice how they are forced to zero.

Equation (2.13) states how to calculate the integral of the relay output, i.e. I_u. Since the on and off times for the relay is being calculated in this section as stated above and the adjustment of u_{on} and u_{off} were done previously in Section 3.2, the calculation of I_u is straightforward according to equation (2.13).

3.6 Calculation of parameters

To calculate the parameters for both the estimated model and the PID controller the results from Section 3.5 are needed. In the code there is a flag which informs whether the experiment has converged or not. When convergence is achieved a function call is made. This function uses previously calculated variables to determine ρ and τ, according to equation (2.14) and (2.15). Depending on the estimate of τ, either parameters of a FOTD or ITD model are computed from equation (2.16), (2.17), (2.18) or (2.19), (2.20) respectively. In conclusion, the model parameters computed

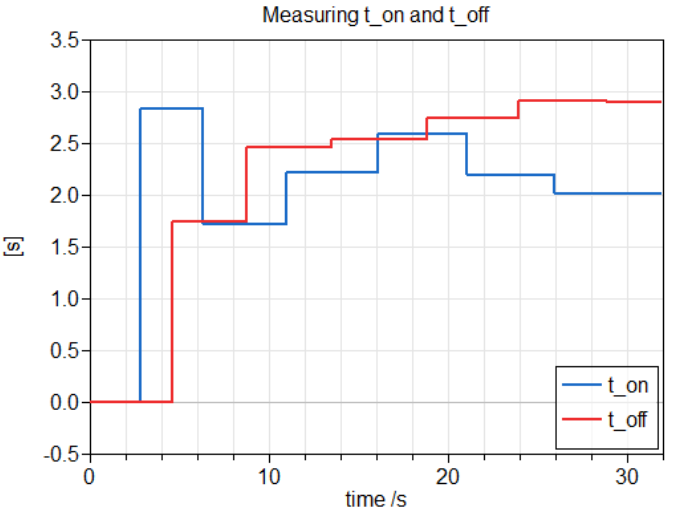

Figure 3.8: Illustrates the continuous measurements of t_{on} and t_{off} during an experiment. Their values are stored when convergence is achieved.

in the step above are used as inputs when determining the final PID parameters using the AMIGO tuning rule.

3.7 Workflow

One of the main goals with this project is to create a comfortable workflow for the user. This can be done by looking into and optimize specific parts. For instance the handling of information is taken into consideration. The way to present the results from the experiment can differ. Another aspect is what type of settings that is necessary from a user perspective. Should it be possible to use the auto-tuner in some type of advanced or normal mode etc.

Comparison between the estimated process and the real

One relevant question to answer after the experiment is how well the estimated model imitates the actual process. This is done by using the relay once again and switching its output. The signal from the relay is coupled to both the process and the estimated model, according to Figure 3.10. Lastly there is a comparison of the two outputs, resulting in a plot looking like the one in Figure 3.11. By observing the comparison one can draw conclusions concerning the accuracy of the estimated model. Since the PID parameters are based on the estimated model a bad estimation clearly would have a negative impact on the resulting parameters. In other words, if the user notice bad control performance when using the resulting PID parameters

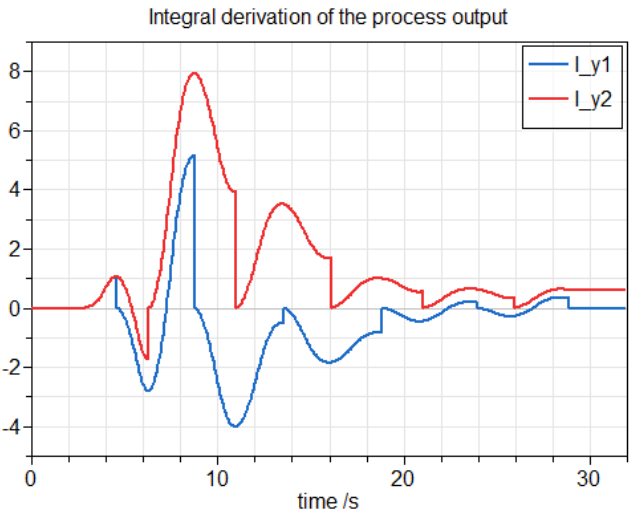

Figure 3.9: Shows the two integrals regarding the output. Notice both how they are forced to zero and that the red curve is lagged with respect to the blue. Observe the blue curve and its value just before it is reset to zero the last time. That value is very close to what the red curve were just before it was reset the last time. This indicates convergence. Depending on when convergence occurs one of the integrals is used for further calculation.

from the experiment he or she can check whether it is a problem with the estimated model or not.

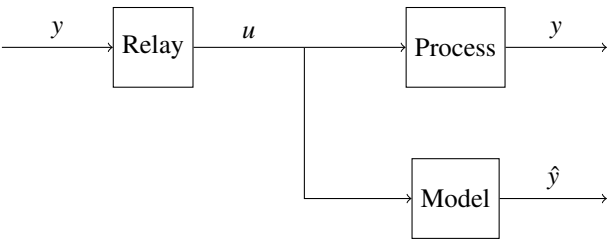

Figure 3.10: Comparison between the estimated model and the real model in a block diagram view. Here u represents the relay output, y is the process output and $\hat{y}$ is the output generated by the estimated low-order model.

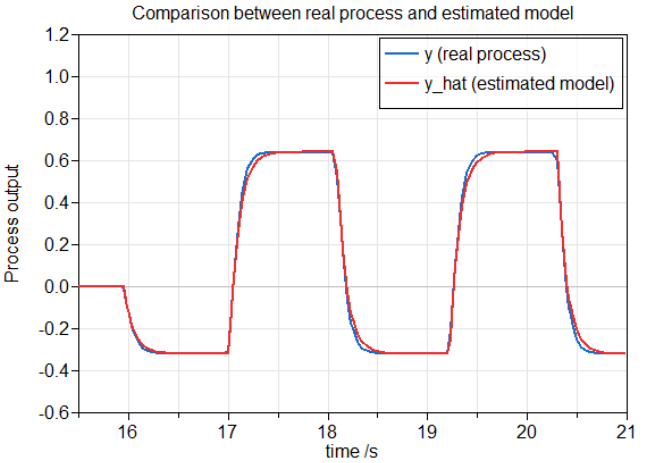

Figure 3.11: Comparison between the estimated model and the real model.

Generating a text file

As mentioned above the estimated model and its parameters are crucial parts of the experiment. Therefore, this information is written to a text file placed in the workspace together with additional results that may be useful for the user to look at. This is done automatically just before the simulation is terminated. The estimated model parameters of interest are T(FOTD), L and K_p or k_v. In addition to those the calculated PID parameters are also presented in the file. The name of the text file is "$info\_file$" by default. An example of a generated text file is shown in Figure 3.12. To start with the PID parameters are presented. This is followed by a simple figure of the model block and its parameters.

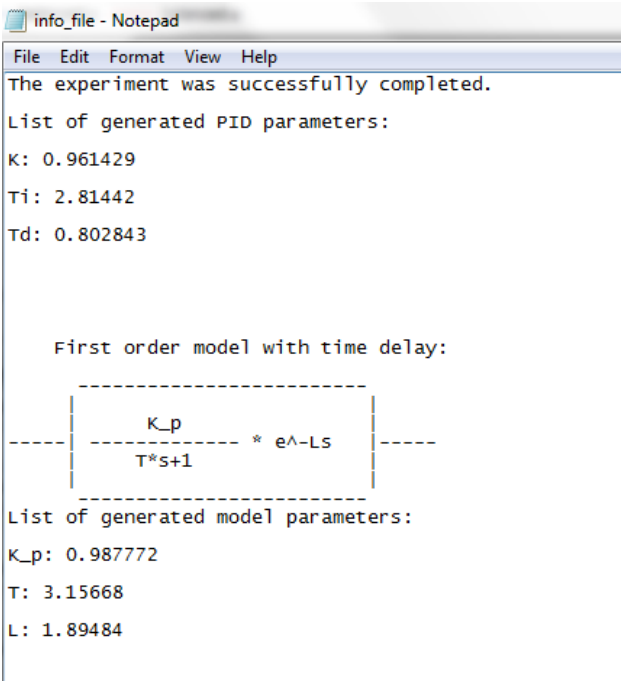

Figure 3.12: When the experiment is terminated an information file is generated containing controller and model parameters.

Information from the message window

Another presentation of the crucial parameters, from the experiment, is given through the message window. Dependent on the initiation, a quantity of parameters are displayed in the window. There are two ways to initiate how much information one wishes to see in the window. These are respectively normal mode or advanced mode. The default mode is normal. It simply displays PID parameters and the estimated model parameters. Choosing the advanced track instead gives more information about the experiment. It still provides the same information as normal mode but the number of displayed experiment parameters is extended. For instance one gets information about the experimental time and the value on τ. An example is shown in Figure 3.13.

If one wishes to run more than one auto-tuner during a certain simulation each result is displayed in the window. For each instance created they are separated with its unique name.

```
--------------------------------------------------------------------------------
Information regarding further analysis...

To generate plots showing reference following and step
response both for the process and the estimated model
use the function called 'plot'.

For more information see file called 'info_text_LinearModel.autotuner'
in workspace.
--------------------------------------------------------------------------------

-------------------------LinearModel.autotuner--------------------------

A textfile info_file_LinearModel.autotuner.txt was successfully created
List of generated PID parameters:

K: 0.681841

Ti: 1.4889

Td: 0.512521

The system has been estimated as a first order time delayed model,
with parameters:

K_p: 0.999262

T: 1.40394

L: 1.31253

Remaining information:

The experiment converged after: 42.4825 s
Tau was estimated to be: 0.483176
```

Figure 3.13: When the experiment is terminated information is shown in the message window in Dymola.

Documentation

In order to present information about the auto-tuner a HTML documentation has been done. This is accessible when the block is selected. It is a description of the model and the parameters within. Changeable parameters as shown in Figures 3.3 and 3.4 are also described.

4

Experiments and Results

The following section contains several tests with the purpose of evaluating how well the auto-tuner works together with processes of different kind, with various complexity. Mainly four models were chosen. The first one was taken from the large batch in [Åström and Hägglund, 2006] containing 134 common processes from industry. It is a linear model and it was used to verify the performance of the auto-tuner in an early stage.

Furthermore, three non linear processes were chosen. Two from a thermodynamics library and one from an aerodynamics library. Processes with various properties concerning stability, dynamics and number of control loops were chosen to see what the auto-tuner could handle.

To compare the results different methods have been used. To measure how well the generated PID parameters work when a constant load disturbance is applied, the integrated absolute error, IAE, is used.

$$IAE = \int_0^\infty |e(\tau)| \, \mathrm{d}\tau. \tag{4.1}$$

To analyze the behaviour during step changes a combination of the overshoot in percentage and rise time is investigated. They are related to the bandwidth. A system with large bandwidth proves to have a faster response than a system with a small.

The percentage overshoot is defined as:

$$\mathrm{PO} = \frac{y_{max,peak}}{y_{step}}, \tag{4.2}$$

where $y_{max,peak}$ is the peak value of the step response and y_{step} is the step size. Both variables are presented as deviations from the start value. The rise time of a step change is defined as:

$$\mathrm{RT} = t_{0.9y_{step}} - t_{0.1y_{step}}, \tag{4.3}$$

where $t_{0.9y_{step}}$ and $t_{0.1y_{step}}$ are time points at which the process output has reached a level corresponding to 90% and 10% of the total step change.

To generate a reference model Modelica LinearSystems2 is used. The resulting model is then used to compute the maximum sensitivity, M_s, which is defined as:

$$M_s = \max_{0 \leq \omega \leq \infty} \frac{1}{1 + P(i\omega)C(i\omega)}. \tag{4.4}$$

It is the transfer function between the process output load disturbance and the output. It describes how sensitive the closed loop system is for errors in the model. The inverse of M_s gives the minimal distance between the Nyquist curve and the critical -1 point at the x-axis. Meaning that if M_s is large the distance is small and the process is consequently more sensitive. By definition M_s can not be smaller than 1 since the complementary sensitivity function is

$$T = 1 - S = \frac{P(s)C(s)}{1 + P(s)C(s)}, \tag{4.5}$$

which also is called the closed loop system. As one can see in equation (4.5) the sum of S and T must always be equal to 1 [Åström and Hägglund, 2006].

To compare the results generated by the auto-tuner a function, `pidtune()`, from Matlabs control system toolbox is used. The toolbox refers to [Åström and Hägglund, 2006]. It derives PID parameters for the reference model based on a phase margin larger than 60 degrees. The reference model used is generated by Modelica LinearSystems 2.

For one of the processes found below Skogestad's half rule were used for a later comparison with already existing tuning techniques for PID controllers. That rule is used in purpose of reducing the order of the transfer function, generated by Modelica LinearSystems2. Usually a high order model is given and to use the AMIGO tuning method a low order model is required. If one wish to know more about Skogestad's half rule, read for instance [Skogestad, 2003].

4.1 Linear process

About the process The process stated in equation (4.6) was retrieved from a test batch, presented in [Åström and Hägglund, 2006], containing 134 common processes regarding process control.

$$P_1(s) = \frac{1}{(0.7s + 1)^2} e^{-s} \tag{4.6}$$

A representative view of the experiment is shown in Figure 4.1, taken from the diagram view in Dymola. The experiment starts up in non steady-state.

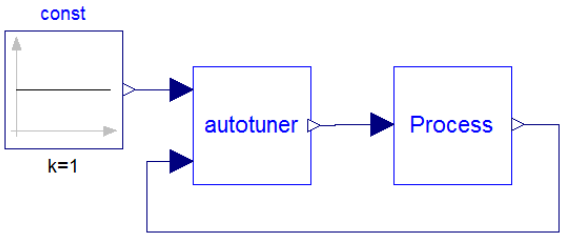

Figure 4.1: Test on a linear process.

Setup Before starting the simulation some parameters in the auto-tuner are adjusted, although the default values in Table 6.8 are used for almost all. The parameters that were changed for this experiment are listed in Table 4.1. It would work with a controlled input as well in this case. The value on *der_check* and *u_const* was chosen empirically.

Initialization	
Parameter	**Choice**
init_mode	Constant input
ss_mode	"Unfixed" (derivative check)
der_check	70
u_const	1

Table 4.1: Modified parameters for the linear process.

Experimental result In Figure 4.2 the whole experiment flow is presented in terms of the output from the relay and process, starting with the initialization and ending after convergence has occured. Observe the different parts, i.e. the exponential ramping of the relay output, the adjustments of the amplitudes and how the process output ends up in a nice oscillation.

From the plots in Figure 4.2 one can notice that the process reaches steady-state after approximately 5 seconds. The experiment does not start until 15 seconds though. This implies that it would be possible to decrease the number of derivative checks necessary before beginning the experiment, i.e. decreasing *der_check* in Table 4.1.

Furthermore, in Figure 4.3 the comparison between the estimated model and the actual process is showed. Both Figures 4.2 and 4.3 are generated by a script called "Plot".

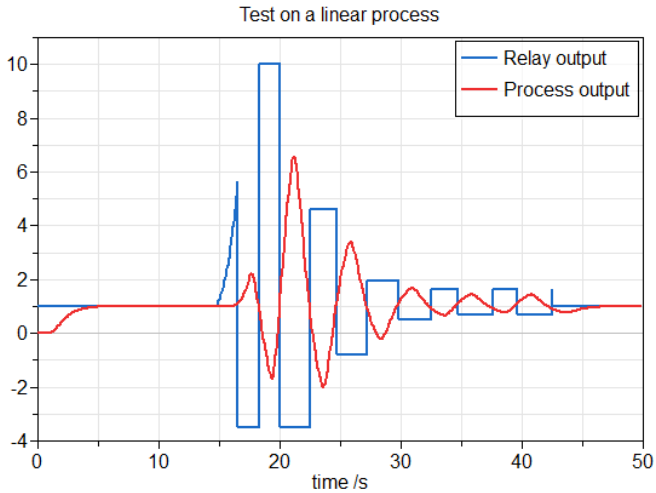

Figure 4.2: The relay experiment. The red curve is the process output and the blue curve is the the relay output.

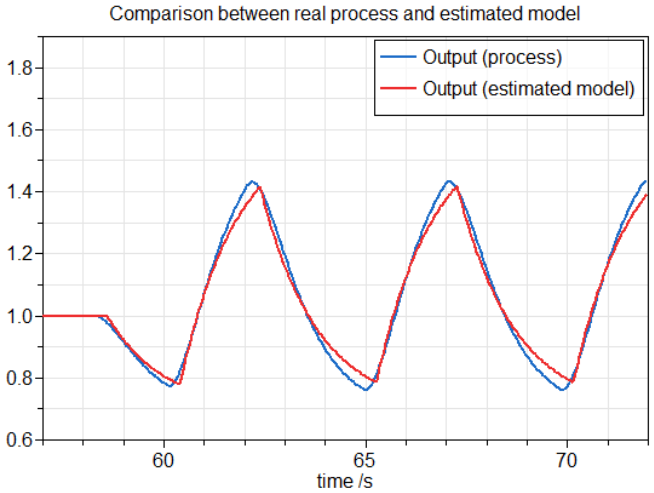

Figure 4.3: Comparison between the estimated model and the real process. The red curve represents the output from the estimated model and the blue curve represents the output from the real process.

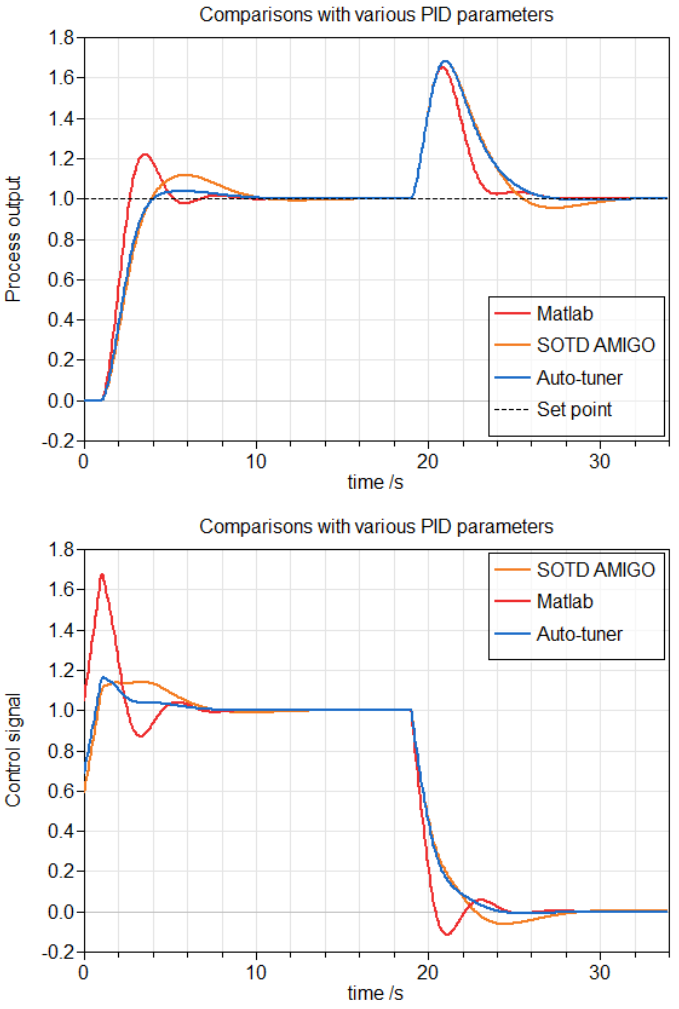

Figure 4.4: Comparisons between different PID parameters. Both regarding an initial step and a constant load disturbance occurring after 18 seconds. In the upper plot the process output is compared whereas the control signal is shown in the lower.

When the simulation is done a message window pops up in Dymola. Results are displayed under the tab "simulation". Advanced mode was chosen initially which means that some extra results are presented. In Table 4.2 the experiment results regarding the linear process are listed together with data concerning the performance when using different parameter set-ups. The set-ups used in the comparison comes from both Matlab and the AMIGO tuning rules. Since the mathematical model was

known it was straightforward to use the function `pidtune()`. By the same reasoning the AMIGO tuning rules were applied together with the given second order system. A comparison on how the different parameter set-ups handles an initial step and a constant load disturbance is shown in Figure 4.4.

Comparisons with various PID parameters					
Method	K	T_i	T_d	RT	PO
Matlab	1.045	1.68	0.42	1.14 s	21.80 %
Auto-tuner	0.69	1.49	0.51	1.87 s	3.65 %
SOTD AMIGO	0.58	1.17	0.68	1.93 s	11.66 %
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Matlab	1.61	1.92		Model type	FOTD
Auto-tuner	2.21	1.49		K_p	0.999
SOTD AMIGO	2.33	1.38		T	1.40
				L	1.31
				Converged	42.48 s
				τ	0.48

Table 4.2: A summary on results regarding the linear process. The upper part presents different PID parameters and performance data in terms of rise time etc. In the lower part results regarding the experiment is listed.

4.2 Non linear processes

The processes in the following section are non linear. In order to have a reference model for these examples the already existing library in Dymola, LinearSystems2, is used, which derives a linear model at a certain time point.

First thermodynamic process

About the process This test is on a thermodynamic process. The main purpose is to control the steam temperature by injecting water into a mixed volume. The controller task is to regulate the valve so that the steam temperature becomes the desired. By opening the valve, more water from the source is injected to the mix. Since it is a valve it cannot be opened more than 100% or closed more than 0%. Therefore the limits for the valve are specified by 1 and 0 respectively. In Figure 4.5 one can see what the process looks like in Dymola. In the upper right corner the controller has been replaced by the auto-tuner, which is connected to the measured steam temperature and the set point.

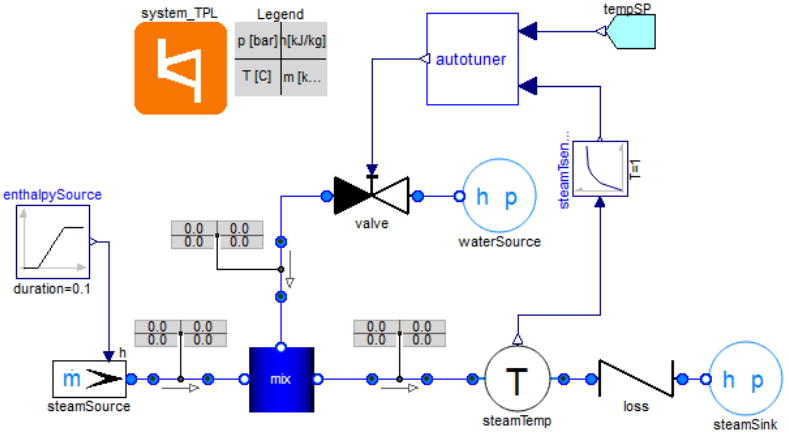

Figure 4.5: The model steam temperature control with the auto-tuner inserted in the loop.

Setup Like in the first experiment, parameters in the auto-tuner block are set before the experiment starts. In Table 4.3 one can see the parameters that were adjusted in this case. Recall the characteristics for a valve and notice the limits on the control signal, i.e. the values on *uMin* and *uMax*.

To perform the experiment equation (3.2) is required to be satisfied. Since *uMin* is equal to 0, an offset needs to be added to the control signal. To gain the steady-state level a constant input is applied and a derivative check examines whether the signal has reached steady-state or not. One can expect that the process reaches steady-state after a certain time, i.e. it is stable, since the process deals with mixing water in order to achieve a certain temperature.

Experiment parameters	
Parameter	**Value**
uMax	1
uMin	0

Initialization	
Parameter	**Value**
init_mode	Constant input
ss_mode	"Unfixed" (derivative check)
der_check	30
u_const	0.6
Stabilizing Controller	
k	-1
T_i	1

Table 4.3: Setup steam temperature control.

Experimental result The relay experiment is shown in Figure 4.6 where the oscillation of the process output is in the upper part and the relay output is in the lower.

In the figure one can see that the process output goes in the opposite direction compared to the direction of the relay output, indicating that the process has negative gain.

Figure 4.7 shows a comparison on the estimated model against the real process in terms of the output. In this case the output from the estimated model differs a bit from the real process. However, it turns out that the computed PID parameters result in satisfactory control behaviour regardless of the moderate model.

In Figures 4.8 and 4.9, one can see comparisons on the control performance when five different parameter set-ups are being used. Both in terms of a step change and a constant load disturbance. Because τ assumed the value 0.1 in this experiment, i.e. the limit that decides whether the estimated model should be a FOTD or ITD, two simulations were made with the auto-tuner. In the first case a FOTD model was estimated whereas an ITD model was estimated in the second. The two set-ups are compared against the set-up being used in the model today and set-ups derived by the control system toolbox in Matlab and Skogestad's half rule respectively.

In Table 4.4 data regarding the performance using the different parameter set-ups are listed together with model specific information generated by the experiment.

Comparisons with various PID parameters					
Method	K	T_i	T_d	RT	PO
Current	-1	1	–	2.19 s	0 %
Auto-tuner (FOTD)	-0.51	0.23	0.028	1.87 s	0.14 %
Auto-tuner (ITD)	-0.62	0.39	0.024	1.87 s	0.09 %
Matlab	-0.09	0.63	0.089	1.88 s	0.37 %
Skogestad	-0.61	0.26	0.022	1.87 s	0.12 %
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Current	0.28	1.65		Model type	FOTD
Auto-tuner (FOTD)	0.14	1.4		K_p	-8.07
Auto-tuner (ITD)	0.18	1.26		T	0.49
Matlab	2.084	1		L	0.055
Skogestad	0.12	1.43		Converged	22.75 s
				τ	0.10

Table 4.4: A summary on results regarding the Steam Temperature Control model. The upper part presents different PID parameters and performance data in terms of rise time etc. In the lower part results from the experiment are listed.

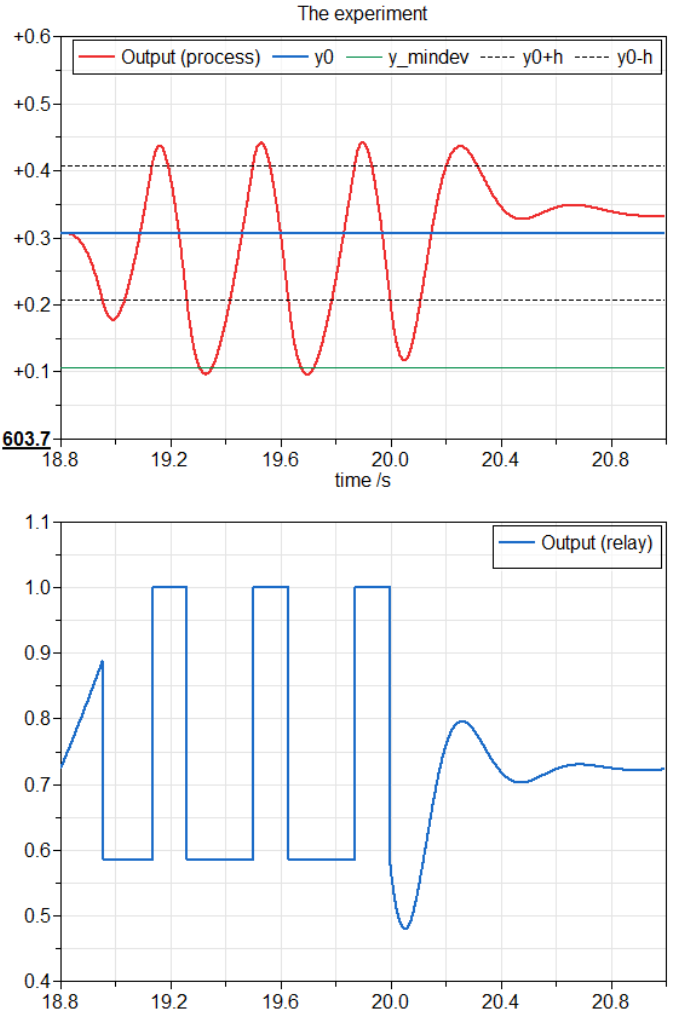

Figure 4.6: The relay experiment. In the upper plot the process output is shown in red. In the bottom plot the relay output is shown in blue.

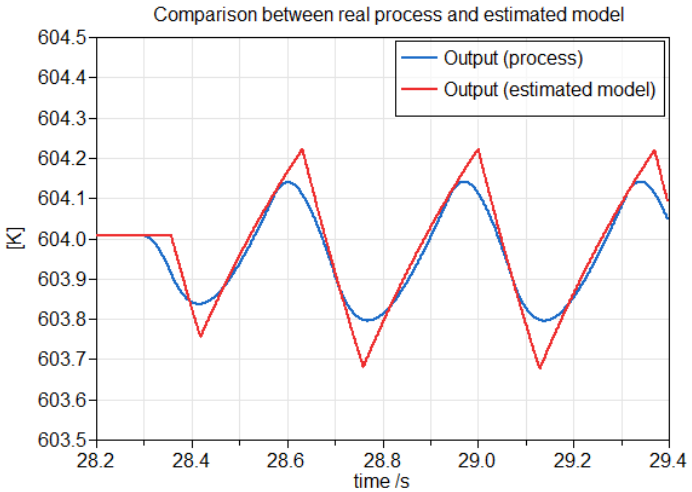

Figure 4.7: Comparison between the estimated model and the real process. The red line represents the estimated model and the blue line represents the real model.

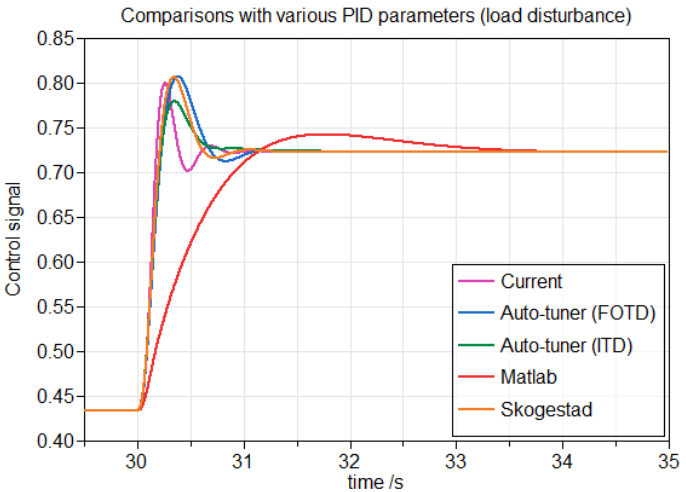

Figure 4.8: Shows the control signal when the different parameter set-ups are being used. The constant load disturbance appears after 30 seconds.

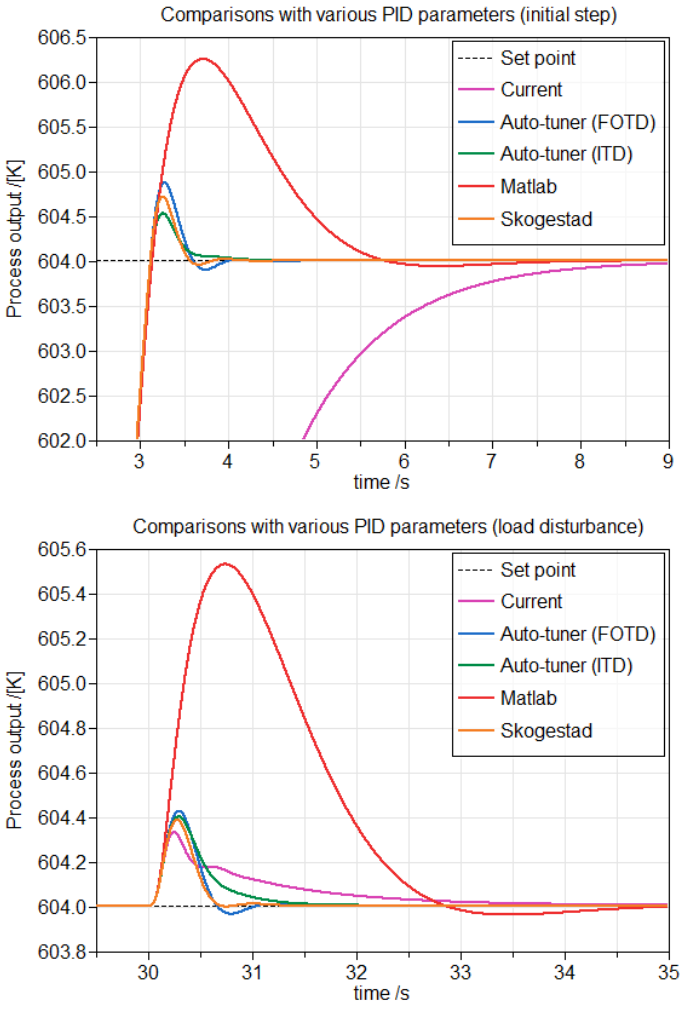

Figure 4.9: Comparisons on the process output when different parameter set-ups are used. In the upper plot responses to a step change is shown whereas the response to a constant load disturbance is shown in the lower.

Second thermodynamic process

About the process In this model there are two variables one wishes to control, namely the pressure and level in the drum. In this experiment the focus is on the latter parameter, i.e. the level. It is controlled by regulating the valve, which in turn controls the amount of water being transferred into the drum from the water source. Again, one has to be aware of limitations when using a valve, i.e. it cannot be opened more than 100% or closed more than 0%.

In Figure 4.10 one can see what the process looks in Dymola. The controller has been replaced with the auto-tuner in the upper left half of the model. The valve being controlled is called "feedValve" and the water source is shown to the left. This model is unstable.

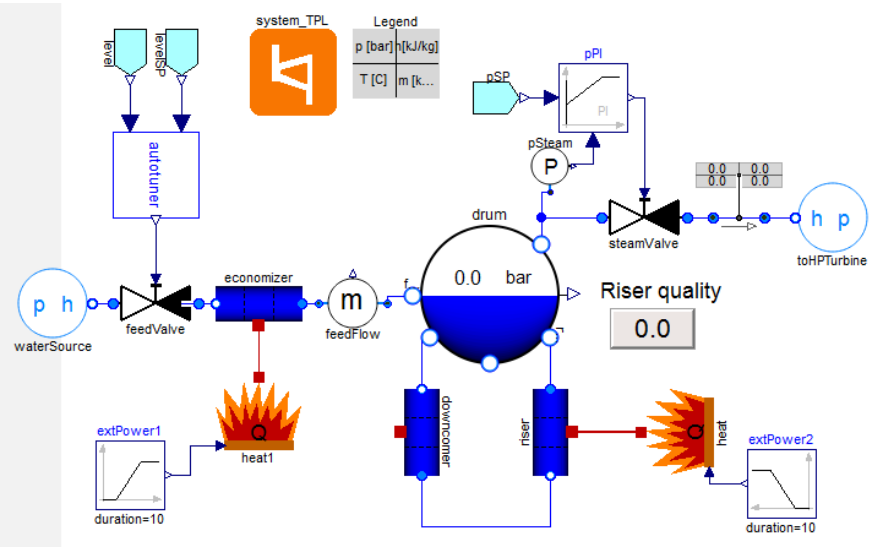

Figure 4.10: The model Natural Circulation Boiler with the auto-tuner inserted in the loop.

Setup In Table 4.5 one can see which parameters that were adjusted before starting the simulation. Unlike the cases with the previous processes it was necessary to set the boolean "force_conv" to true in this case. The reason for this is further discussed in Section 5.1. Since the controlled actuator once again is a valve, limits regarding the control signal needs to be adjusted to 1 and 0 respectively. Another parameter that differs from previous experiments is "init_mode". Since the process is unstable the controlled input is chosen.

Experimental result The relay experiment is shown in Figure 4.11, where the process output and its oscillation can be seen in the upper plot and the relay output

Experiment parameters	
Parameter	**Value**
uMax	1
uMin	0
force_conv	true

Initialization	
Parameter	**Value**
init_mode	Controlled input
ss_mode	"Unfixed" (derivative check)
der_check	30
Stabilizing Controller	
k	2
T_i	50

Table 4.5: Setup for Natural Circulation Boiler.

in the lower. Notice how the process output changes direction directly when the relay output switches, i.e. the process has very fast dynamics. The black dashed lines represents the hysteresis levels. The green line represents the lower limit concerning the process output and it is clear that there is a problem in this case. The process output will never end up in the desirable area. This will be discussed further in Chapter 5.

The estimated model is compared with the real process in Figure 4.12. The result is quite good but do notice how the model output gradually moves upwards. The reason for this is treated in the Chapter 5. The resulting control performance is not affected by this phenomena.

In Figures 4.13 and 4.15 the control performance is shown for different parameter set-ups. Both in terms of a step change in set-point and a constant load disturbance. There are two plots showing the load disturbance response. One of them is just a scaled version of the first one. This is because the response when using the parameters generated by the auto-tuner was so much smaller in comparison to the response when the pre-defined parameters were used. In Figure 4.14 the control signal during the disturbance is shown.

When using the linearized model from Dymola and Matlab in order to obtain PID parameters the results were not satisfactory. The result showed a large steady-state error and there was no response to the load disturbance. This is why only two curves are compared in this case.

In Table 4.6 data regarding the performance using the different parameter set-ups are listed together with model specific information generated by the experiment.

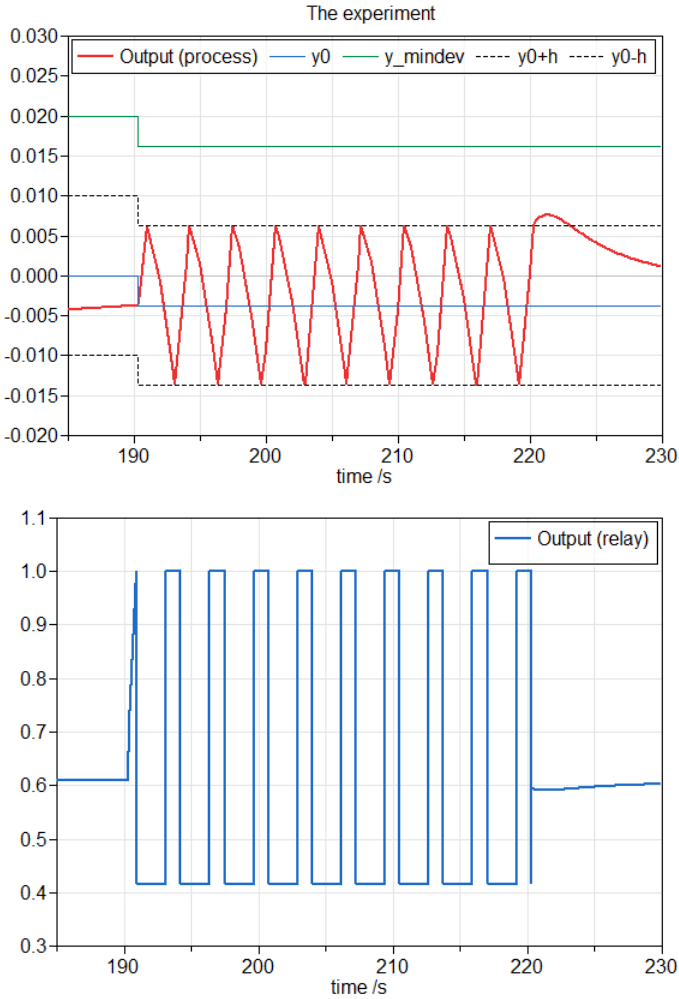

Figure 4.11: The relay experiment. In the upper plot the process output is shown. Notice how it changes direction in the very same moment as it crosses the hysteresis limit. Consequently it cannot reach the level representing the minimal deviation from y0, y_mindev. In the bottom plot the relay output is shown.

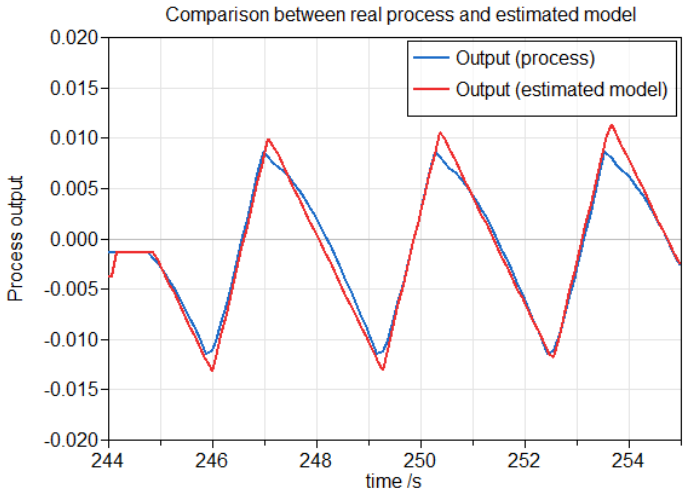

Figure 4.12: Comparison between the estimated model and the real process. The red curve represents the estimated model and the blue curve represents the real model.

Comparisons with various PID parameters					
Method	K	T_i	T_d	Rise time	Overshoot
Current	1	25	-	13.27 s	34.74 %
Auto-tuner	73.51	0.89	0.055	3.21 s	3.48 %
Matlab	504	$6.73 \cdot 10^6$	$1.68 \cdot 10^6$	-	-
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Current	1.14	3.63		Model type	ITD
Auto-tuner	0.00067	1.016		K_v	0.055
Matlab	-	1.65		L	0.11
				Converged	220.268 s
				τ	0.0147006

Table 4.6: A summary on results for the Natural Circulation Boiler. The upper part presents different PID parameters and performance data in terms of rise time etc. In the lower part results from the experiment are listed.

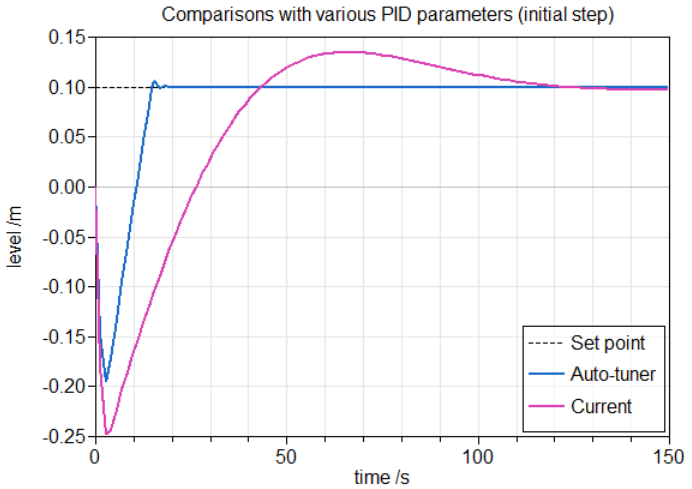

Figure 4.13: Shows the response to a set-point change when different parameter set-ups are used.

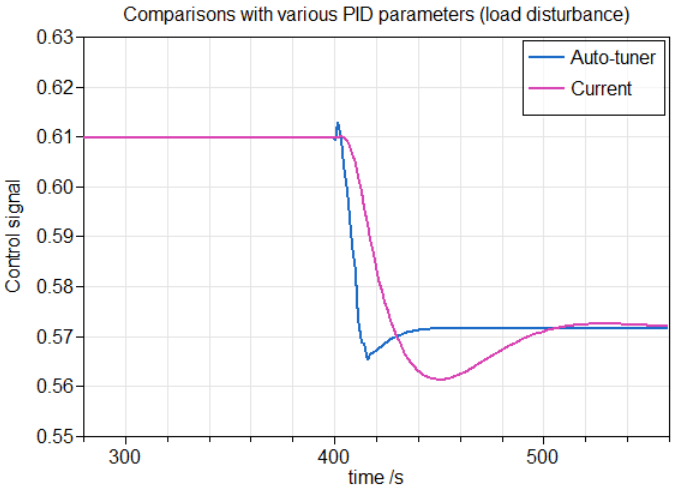

Figure 4.14: Shows the control signal during a constant load disturbance when different parameter set-ups are used. The disturbance appears after 400 seconds.

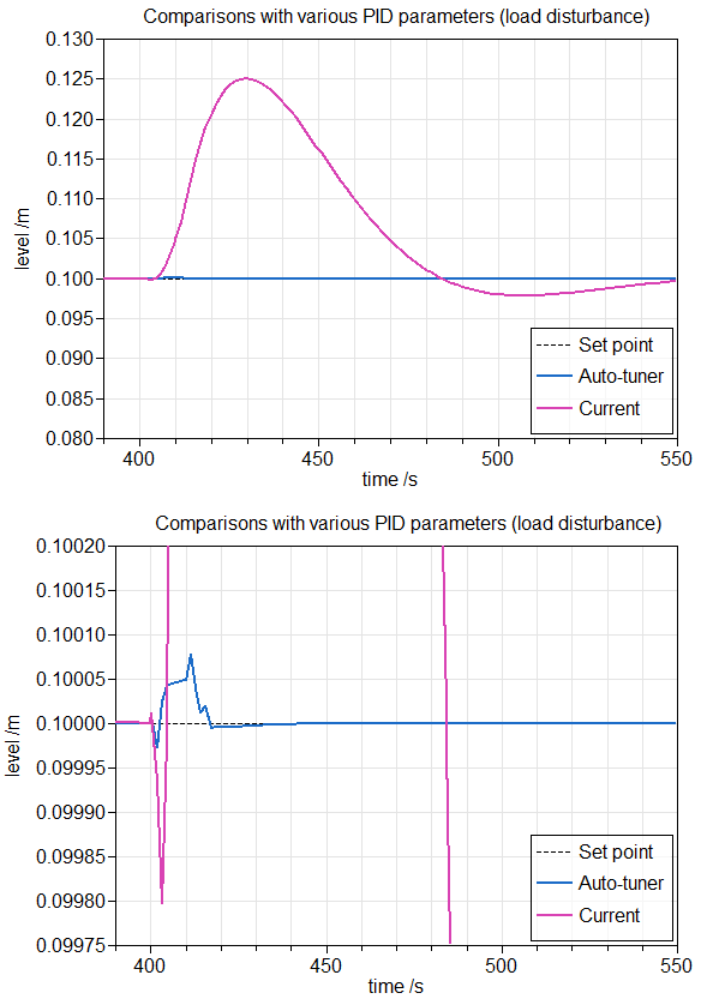

Figure 4.15: Shows the response to a constant load disturbance when different parameter set-ups are used. The plot in the bottom is a scaled version of the upper. Notice how small the response is when using the parameters generated by the auto-tuner.

Aircraft dynamic process

About the process This is a model which describes the dynamics of an aircraft. It contains several components such as an aircraft, an atmosphere which defines the atmospheric conditions, an autopilot controller which controls roll, yaw and vertical velocity, a world component which defines the orientation of the coordinate system as well as the gravity. There is also an initialization component which initiates the aircraft, one component for including different quantities like mass, size etc. That together describes the aircraft. Lastly there is a ground component which specifies ground conditions. This example mainly focus on maneuvers in the air and therefore the aircraft is initialized when being there. Another thing to notice is that the orientation of the coordinate system is defined such that a negative input from the vertical velocity controller results in an increased altitude.

Figure 4.16 illustrates what the model looks like in Dymola, with all the components described above represented. The component that is connected to the aircraft with yellow connections is the autopilot controller. If one enters that component, i.e. goes down one level, a model of three control loops is shown. This is demonstrated in Figure 4.17. Due to the fact that there are multiple control loops in terms of roll, yaw and velocity, this is an interesting challenge for the auto-tuner.

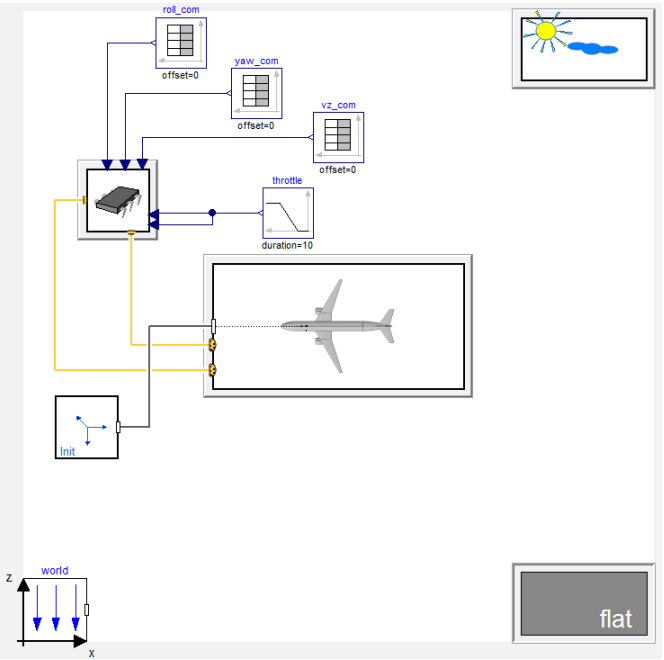

Figure 4.16: Overview of the model from the top level.

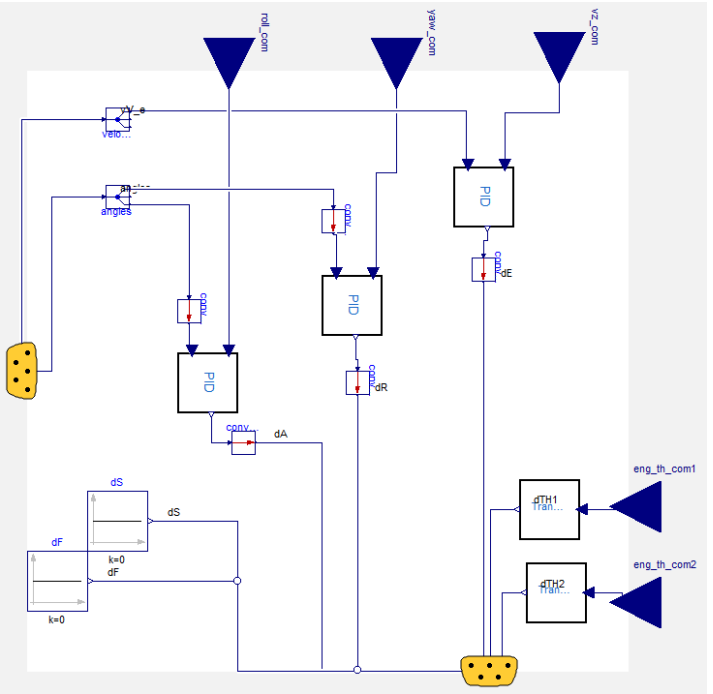

Figure 4.17: Control system regarding the autopilot (starts from left side). Controls roll angle, yaw angle and vertical velocity.

Setup For this experiment three different initiations are needed, one for each control loop. In Tables 4.7, 4.8 and 4.9 parameters that were adjusted before simulating the experiments are shown. Characteristics for each actuator decides the limits on the control signal, i.e. the values on $uMin$ and $uMax$. They are different for all three cases.

Since the systems are unstable a controlled input for each experiment is needed to reach the steady-state level. The parameters for the initiation controllers were given through the trial and error method. Together with that method the experimental start time was obtained through observations.

Experiment parameters	
Parameter	**Value**
uMax	10
uMin	-25

Initialization	
Settings	
Parameter	**Choice**
init_mode	Controlled input
ss_mode	"Fixed" (time check)
t_ss	25
Stabilizing Controller	
u_init	0
K	-0.3
T_i	0.8

Table 4.7: Setup for the vertical velocity experiment.

Experiment parameters	
Parameter	**Value**
uMax	25
uMin	-25

Initialization	
Settings	
Parameter	**Choice**
init_mode	Controlled input
ss_mode	"Fixed" (time check)
t_ss	140
Stabilizing Controller	
u_init	0
K	-0.8
T_i	3

Table 4.8: Setup for the roll experiment.

Experiment parameters	
Parameter	**Value**
uMax	30
uMin	-30

Initialization	
Settings	
Parameter	**Choice**
init_mode	Controlled input
ss_mode	"Fixed" (time check)
t_ss	120
Stabilizing Controller	
u_init	0
K	-0.5
T_i	2

Table 4.9: Setup for the yaw experiment.

Experiment result The results are categorized into three parts, one for each control loop. The PID parameters were obtained in two ways. Firstly by replacing one of the existing controllers with the auto-tuner, then by replacing all three at the same time. In the latter case three experiments ran in parallel during the simulation, generating three different parameter set-ups.

Figures describing the relay experiments concerning each control loop are shown in Figures 4.18, 4.23 and 4.28. Observe that the process output goes in the opposite direction in comparison to the direction of the relay output in all three cases. This indicates that the processes have negative gains. For the roll angle the adjustment of the relay amplitudes were forced to be interrupted due to fast dynamics.

Figures 4.19, 4.24 and 4.29 show how the estimated models follow the process outputs for the different cases. Since they all are ITD models they are quite similar to each other.

The other figures show comparisons regarding control performance for different parameter set-ups. Two of the set-ups are results from the auto-tuner. They are named "Auto-tuner (single)" and "Auto-tuner (multiple)" respectively, where multiple means that three auto-tuners ran in parallel during the simulation. Furthermore, the set-ups currently being used by Modelon are called "Current" and the ones obtained by the control system toolbox in Matlab are called "Matlab".

Vertical velocity Results regarding the relay experiment, comparison test, load disturbance response, step response, reference following, control signal and measurement values for equation (4.1), (4.2), (4.3) and (4.4) for the vertical velocity are shown in Figures 4.18, 4.19, 4.20, 4.21, 4.22 and Table 4.10.

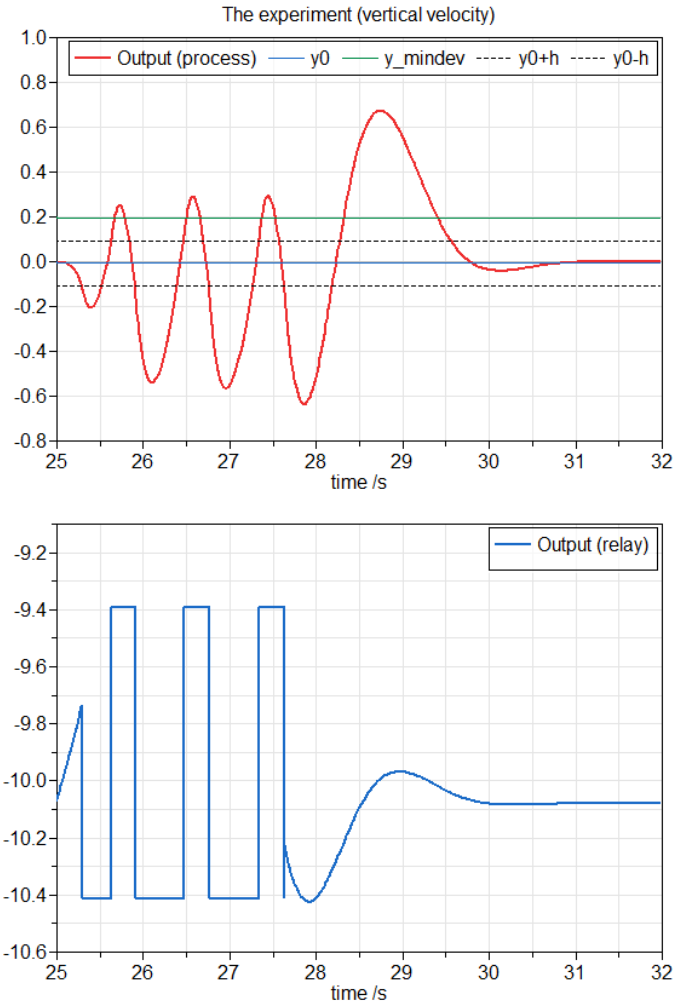

Figure 4.18: The relay experiment for the vertical velocity. In the upper plot the process output is shown in red. In the lower the relay output is shown in blue.

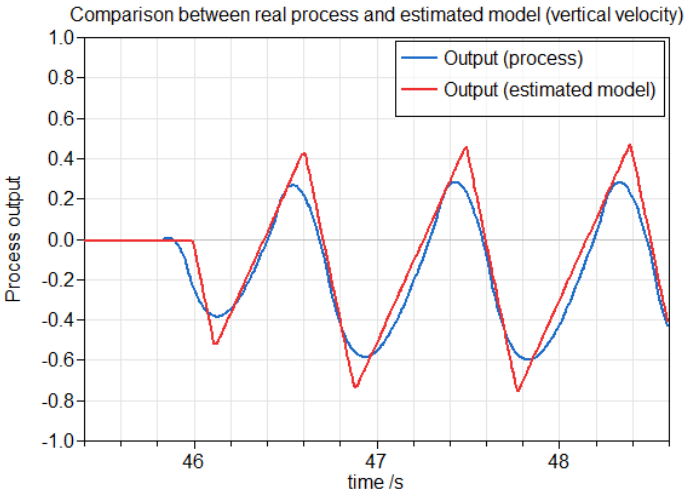

Figure 4.19: Comparison between the estimated model and the real process. The red curve represents the estimated model and the blue curve represents the real model.

Comparisons with various PID parameters					
Method	K	T_i	T_d	RT	PO
Current	-0.3	0.8	0.3	$0.45\,s$	$17.8\,\%$
Auto-tuner(one)	-0.44	1.30	0.081	$0.34\,s$	$19.5\,\%$
Auto-tuner(all)	-0.44	1.31	0.082	$0.342\,s$	$19.5\,\%$
Matlab	$-2.25 \cdot 10^{-5}$	$1.25 \cdot 10^{3}$	303.43	-	-
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Current	3.58	1.12		Model type	FOTD
Auto-tuner(one)	2.32	1.32		K_v	-6.26
Auto-tuner(all)	2.98	1.32		L	0.16
Matlab	-	1.0084		Converged	$27.63\,s$
				τ	0

Table 4.10: A summary on the results regarding the vertical velocity control. Firstly data on the different PID parameters and certain info such as overshoot etc. are listed, then specific info about the estimated model is presented.

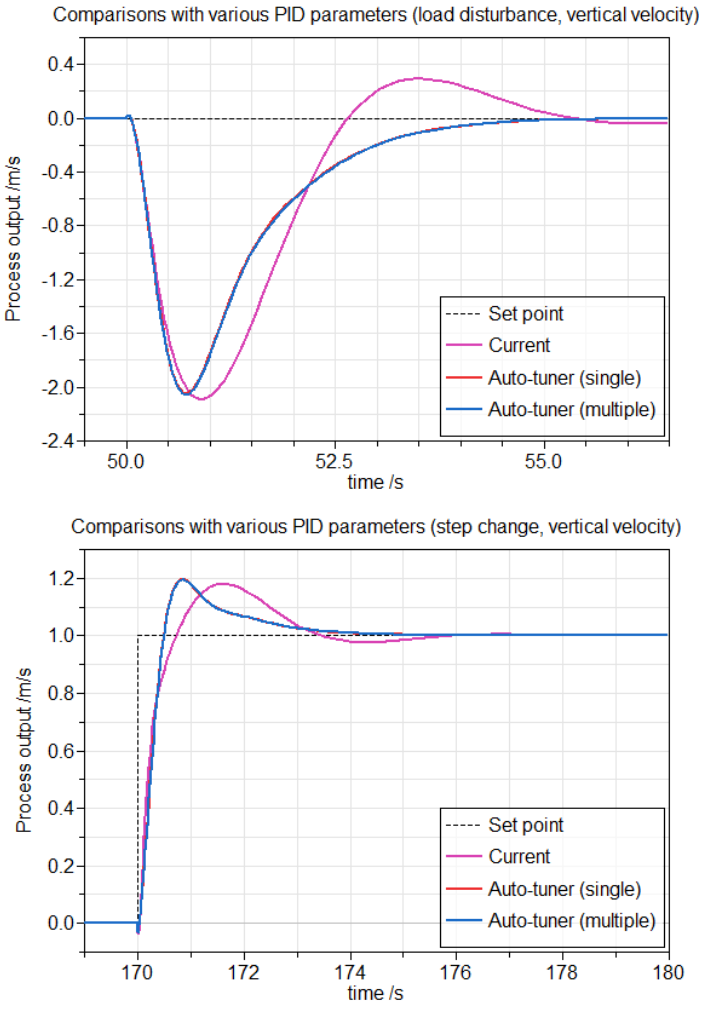

Figure 4.20: Shows the response to a constant load disturbance and a step change when different parameter set-ups are used. Notice that there is almost no difference on the results when one or three auto-tuners are used.

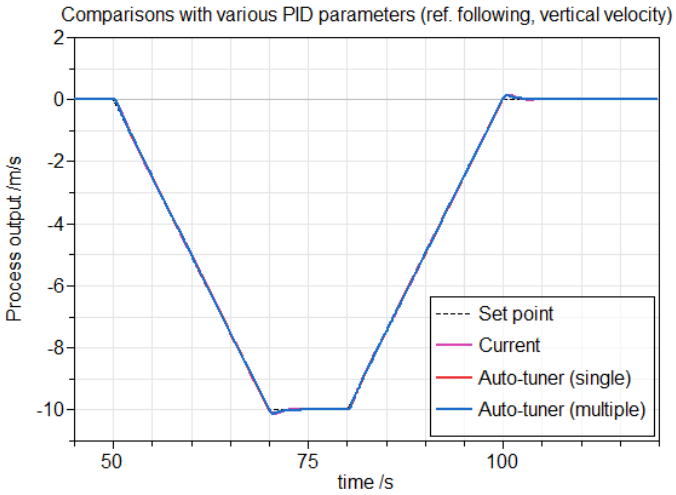

Figure 4.21: Shows the reference following when different parameter set-ups are used. As one can see, they are all following the set point well.

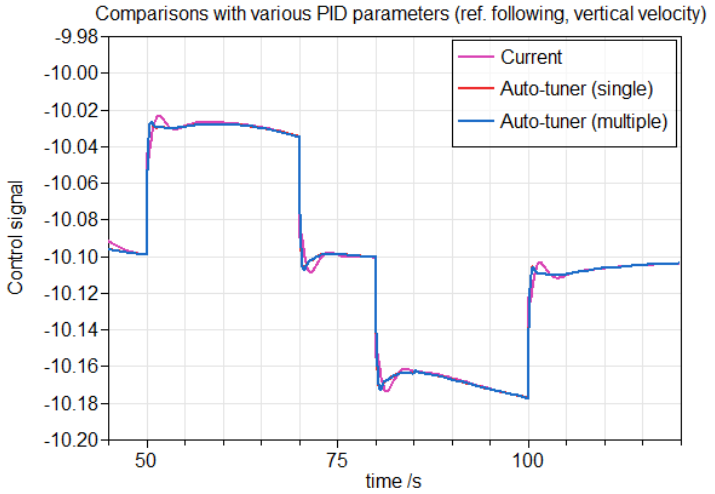

Figure 4.22: Shows the control signal when different parameter set-ups are used.

Roll Results regarding the relay experiment, comparison test, load disturbance response, step response, reference following and measurement values for equation (4.1), (4.2), (4.3) and (4.4) for the roll are shown in Figures 4.23, 4.24, 4.25, 4.26, 4.27 and Table 4.11.

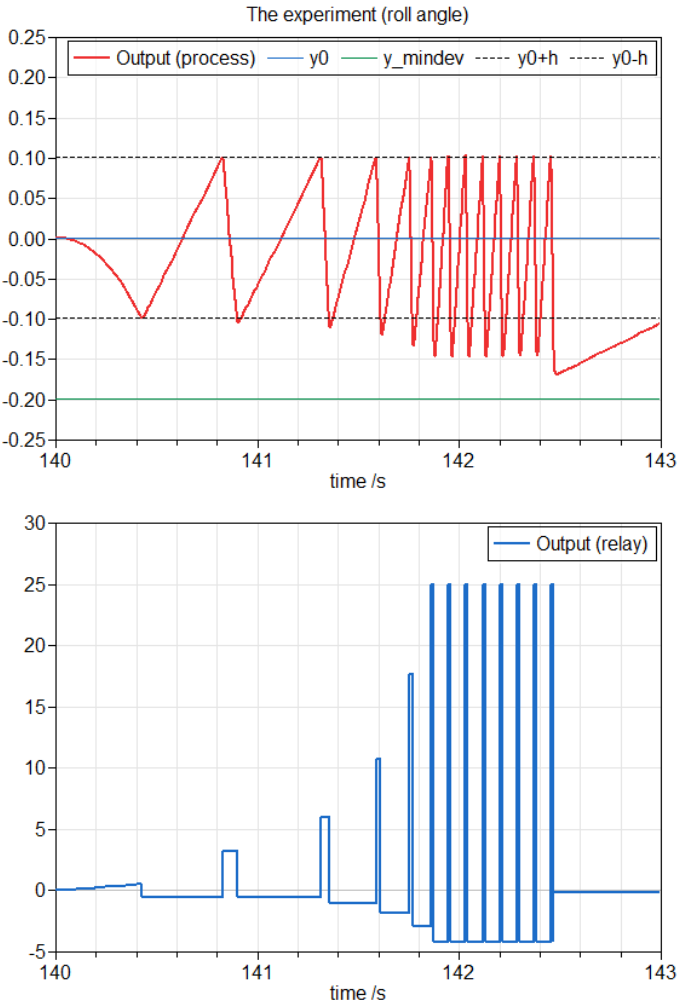

Figure 4.23: The relay experiment for the roll angle. In the upper plot the process output is shown in red. In the lower the relay output is shown in blue.

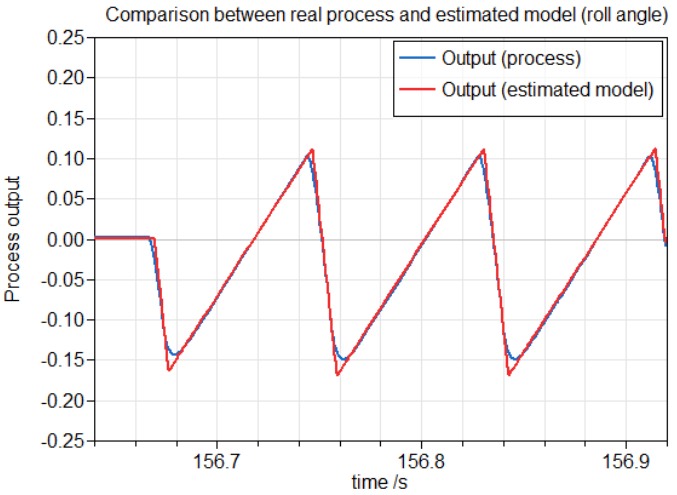

Figure 4.24: Comparison between the estimated model and the real process. The red curve represents the estimated model and the blue curve represents the real model.

Comparisons with various PID parameters					
Method	K	T_i	T_d	Rise time	Overshoot
Current	-0.8	3	0.8	$2.24\,s$	12 %
Auto-tuner(one)	-159.61	0.024	0.0015	$0.035\,s$	64 %
Auto-tuner(all)	-147.13	0.027	0.0017	$0.035\,s$	59.4 %
Matlab	-3.86	2.89	0.029	$0.52s$	6.6 %
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Current	8.66	1.19		Model type	FOTD
Auto-tuner(one)	0.00018	1.11		K_v	-0.93
Auto-tuner(all)	0.00022	1.079		L	0.0030
Matlab	1.42	1		Converged	$141.86\,s$
				τ	0

Table 4.11: A summary on the results regarding the roll angle. Firstly data on the different PID parameters and certain info such as overshoot etc. are listed, then specific info about the estimated model is presented.

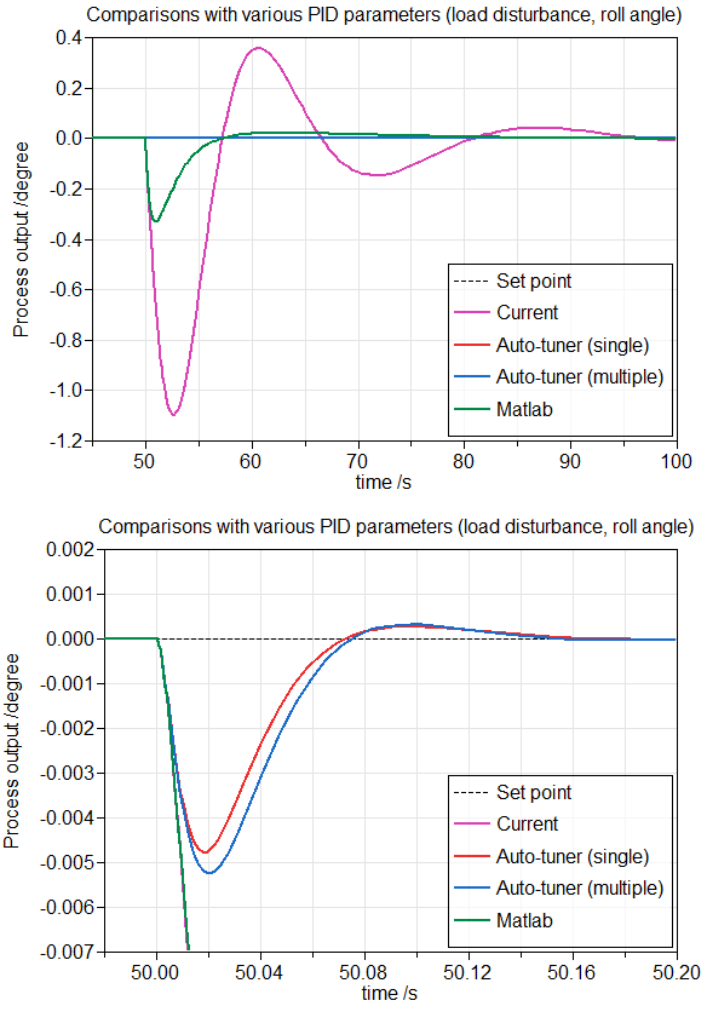

Figure 4.25: Shows the response to a constant load disturbance when different parameter set-ups are used. The plot in the bottom is a scaled version of the upper. Notice how small the response is when using the parameters generated by the auto-tuner. In this case the results are slightly better when only one auto-tuner is used.

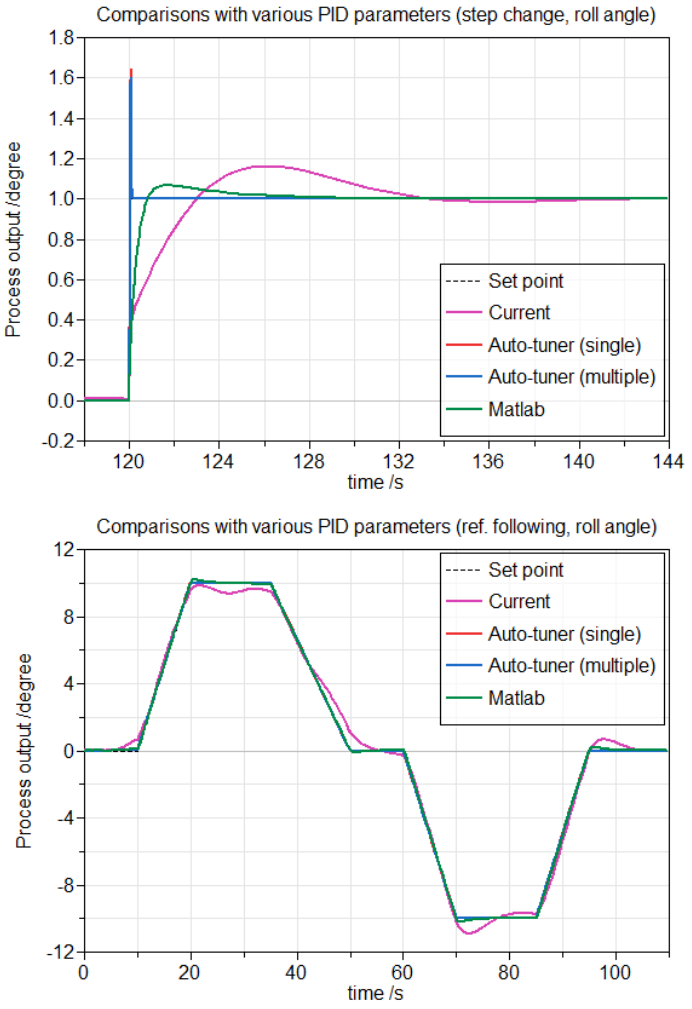

Figure 4.26: Shows the response to a step change and reference following when different parameter set-ups are used. The number of auto-tuners does not affect the results in this case. It might be hard to see the red and the dotted line. The reason for that is that they are located on the top of each other, behind the blue line.

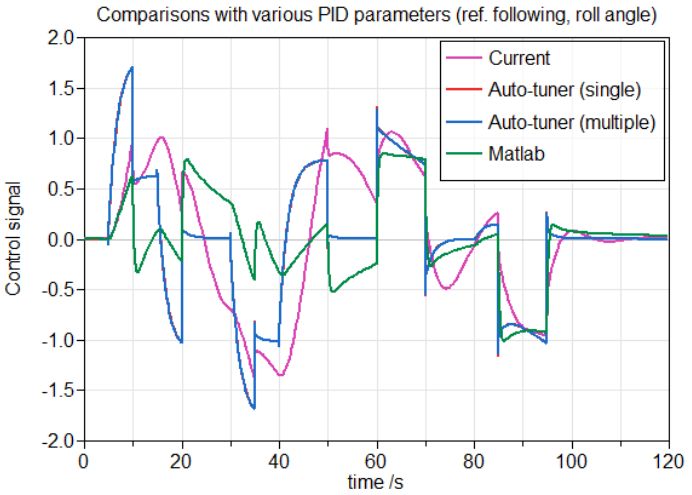

Figure 4.27: Shows the control signal when different parameter set-ups are used.

Yaw Results regarding the relay experiment, comparison test, load disturbance response, step response, reference following, control signal and measurement values for equation (4.1), (4.2), (4.3) and (4.4) for the yaw are shown in Figures 4.28, 4.29, 4.30, 4.31, 4.32 and Table 4.12.

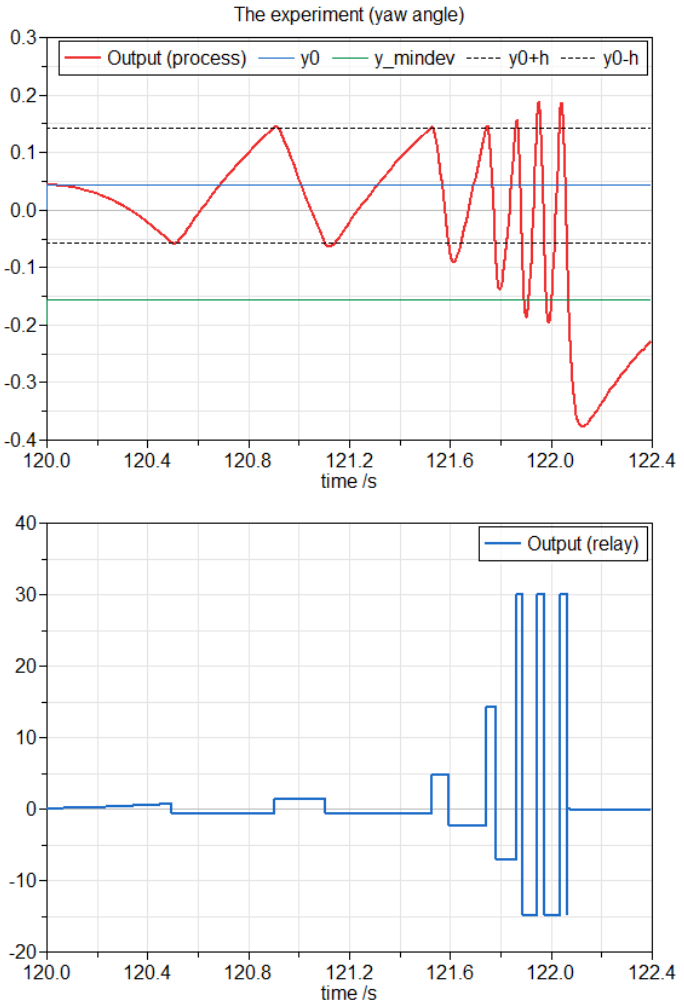

Figure 4.28: The relay experiment for the yaw angle. In the upper plot the process output is shown in red. In the lower the relay output is shown in blue.

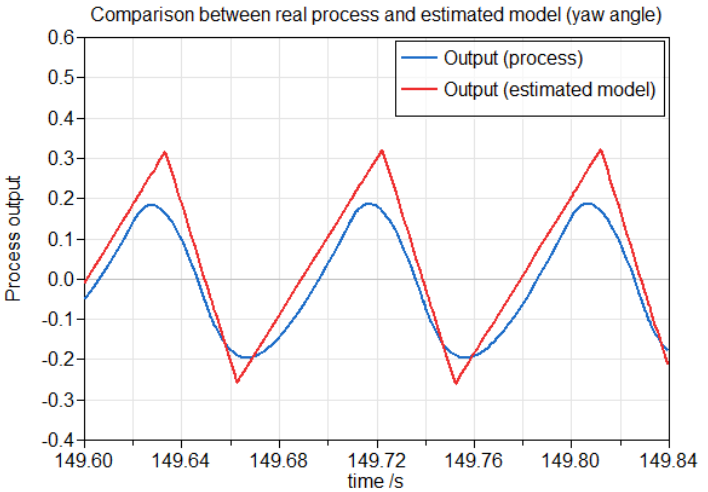

Figure 4.29: Comparison between the estimated model and the real process. The red curve represents the estimated model and the blue curve represents the real model.

Comparisons with various PID parameters					
Method	K	T_i	T_d	RT	PO
Current	-0.5	2	0.2	$4.93\,s$	32.8 %
Auto-tuner(one)	-54.066	0.10	0.0063	$0.04\,s$	37.6 %
Auto-tuner(all)	-68.40	0.088	0.0055	$0.039\,s$	38.2 %
Matlab	-0.45	26.79	0	$14.25\,s$	16%
				Experimental results	
Method	IAE	M_s		**Parameter**	**Value**
Current	7.21	1.22		Model type	ITD
Auto-tuner(one)	0.0019	1.26		K_v	-0.66
Auto-tuner(all)	0.0013	1.33		L	0.013
Matlab	61.15	1.006		Converged	$122\,s$
				τ	0.013

Table 4.12: A summary on the results regarding the yaw angle. Firstly, data on the different PID parameters and certain info such as overshoot etc. are listed, then specific info about the estimated model is presented.

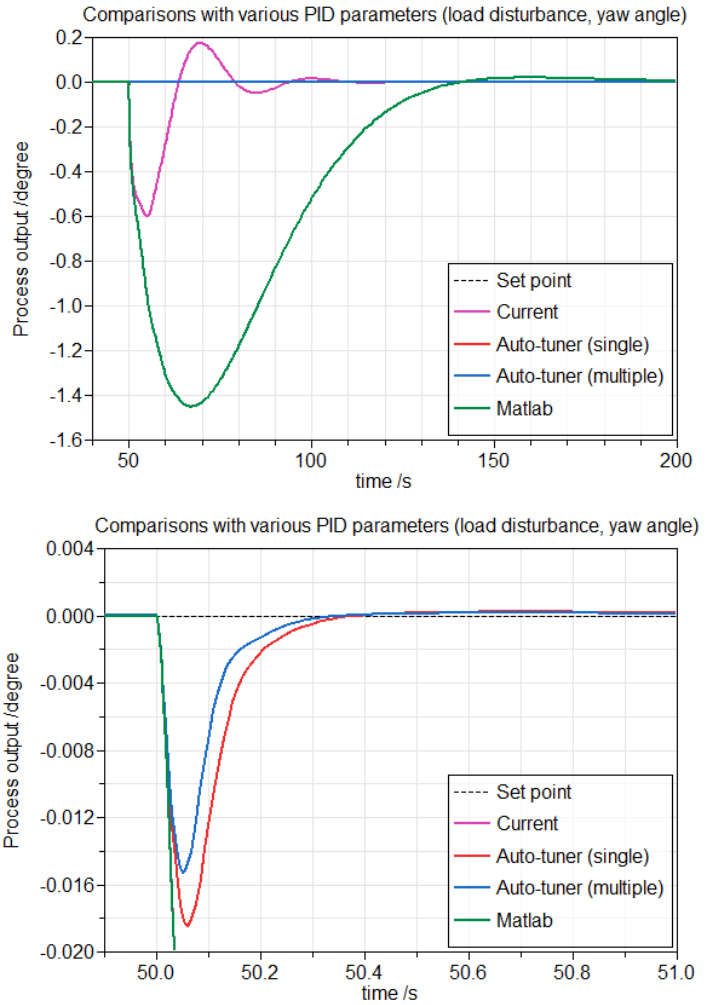

Figure 4.30: Shows the response to a constant load disturbance when different parameter set-ups are used. The plot in the bottom is a scaled version of the upper. Notice how small the response is when using the parameters generated by the auto-tuner. In this case the results are slightly better when all three auto-tuners is used.

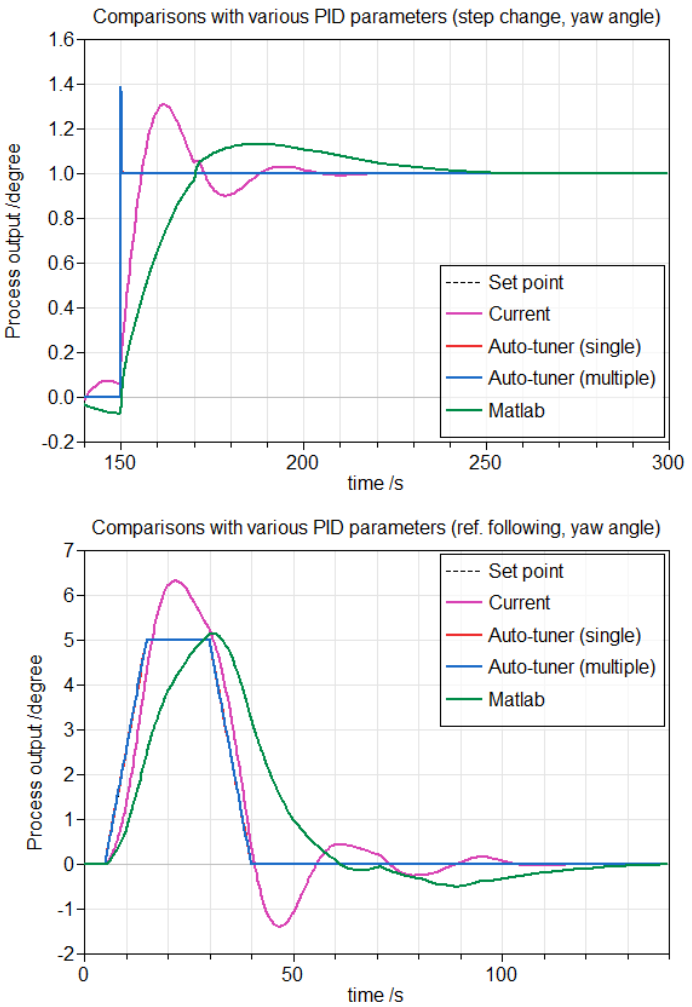

Figure 4.31: Shows the response to a step change and reference following when different parameter set-ups are used. The number of auto-tuners does not affect the results in this case. Notice especially how much better the set-ups generated by the auto-tuners are in comparison to the other two. It might be hard to see the red and the dotted line. The reason for that is that they are located on the top of each other, behind the blue line.

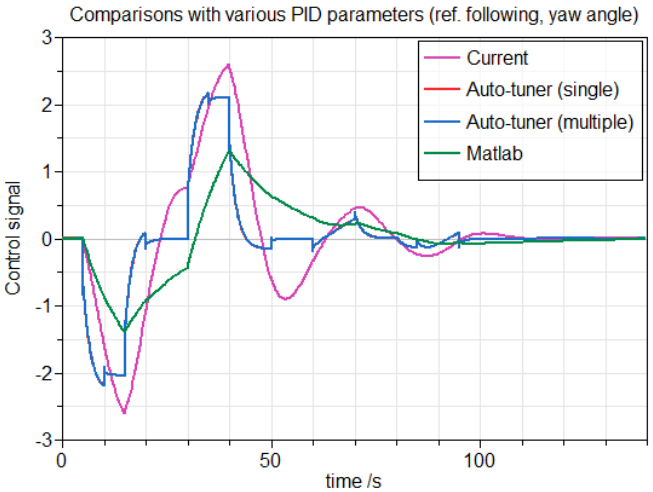

Figure 4.32: Shows the control signal when different parameter set-ups are used. Even though the reference following were a lot better when using the results from the auto-tuners, the control signal still looks good.

5

Discussion and Conclusion

5.1 Introduction

Auto-tuners are a very popular tool in industry and it has existed there for many years. It has been requested to investigate its usefulness in simulation environment as well and now it exists as a helpful tool in the modeling language Modelica. In comparison to industry there are some parts that differ when working with auto-tuners in simulations. Generally, simulations have the advantage that there is no time limits and there is not the same costs involved. Additionally, if things go wrong it is possible to re-run the simulation without risking the process in any way. It means that the experiment can run for a longer time without any consideration on time. This is beneficial since a harder constraint can be put on the convergence check and thereby allow the experiment to oscillate longer in purpose of obtaining a better estimate of the model.

Another advantage that comes from the fact that it is possible to run the experiment multiple times without affecting the process in a negative way, is that one can use several simulations in order to find out if and when the process actually reaches steady-state.

When working with the implemented auto-tuner it is possible to compare the estimated model against the real process. This is another advantage with the solution in Modelica in comparison to industry.

In this chapter restrictions and conditions on the experiment are discussed. That includes the steady-state requirement, information about processes where the experiment does not converge by itself and also about the user interface. The chapter then continues with a part that discusses the results obtained in Chapter 4. Finally, further developments are discussed.

5.2 Restrictions and conditions

The auto-tuner provides an estimated low order model together with control parameters for a PID controller. As one can expect the results may differ for various

processes. In some cases it can be very pleasant whereas for some cases it will not perform enough. For processes that was tested in this thesis the result turned out to be satisfactory. Since only a limited number of processes have been used in this thesis it needs to be further tested in order to guarantee a good performance generally.

The steady-state requirement

One crucial part that has been discussed in this thesis already is the one that states that the control signal and process output have to be in steady-state when the experiment begins, i.e. $(u, y) = (u_0, y_0)$. In Chapter 3 different options regarding this problem are dealt with. It requires some knowledge about the process to apply the auto-tuner.

The implementation contains two options for reaching steady-state. If one chooses the alternative meaning that a constant control signal will be applied, the result depends on the process gain. Applying this method on an unstable process may make the result diverge. On the other hand, if the user uses the controlled input instead, suitable control parameters are still required in order to succeed.

In our case when dealing with the steady-state issue for the different models it was possible to make use of existent controllers and their parameters instead of applying Ziegler and Nichols method for instance. This has of course simplified this part significantly. Depending on the complexity of the model it might get very tricky to come up with good values. This is a drawback we are aware of.

Processes where the experiment doesn't converge

For some processes the experiment does not converge. In order for convergence to occur it is necessary that the adjustment of the relay amplitudes is finished. Since the calculation of the estimated gain uses the integral of the process output together with the integral of the relay output, it is required that the values of the integrals are the correct ones. If an adjustment on the relay amplitudes takes place the calculations may not use the right integral values and consequently the computed gain would be wrong. That is why the adjustments has to be done before convergence is examined.

The adjustments are done in order to make the oscillation stay in a preferable region. The region is decided by y_{maxdev} and y_{mindev} (as shown in Figure 3.4). During the adjustments of the relay amplitudes many things may happen. The first scenario is that the oscillation stays below y_{mindev} and one of the relay amplitudes is saturated, which implies that the process output cannot reach above the limit y_{mindev}. As a consequence the experiment will not converge. The limits y_{mindev} and y_{maxdev} are by default set to $2h$ and $12h$ respectively. One method to circumvent this is to decrease y_{mindev} to a value closer to h in the general settings. Another alternative is to force the experiment to execute the convergence check even though the output signal is not aligned within the preferred region. When changing y_{mindev} and y_{maxdev} it is important that equation (2.8) is satisfied.

Critical processes with a risk of facing the problem discussed in the previous paragraph are those with low order, significantly fast dynamics and/or with a phase lag less than π at high frequencies. This can for instance be a first order model without any time delay as well as a pure integrator model. The fast dynamics makes the process output change its direction at the very same moment as it reaches the hysteresis level. This is why the oscillation never aligns within the preferable region. In [Luyben, 2001] it was shown how the output of such a process behaves during relay control. The ratio $\frac{L}{T}$ gives information about the shape of the curvature. For a first order process, without any delay, the ratio is 0 (in fact small) and therefore a triangular behavior is expected according to [Luyben, 2001]. Some first order processes with a time constant 0.01, 0.1, 1, 10 and 100 has been investigated. Principally they all present the same behavior as in Figure 5.1. The one in Figure 5.1 has the time constant T = 10. As one may see the output signal never reaches above the y_{mindev}-level.

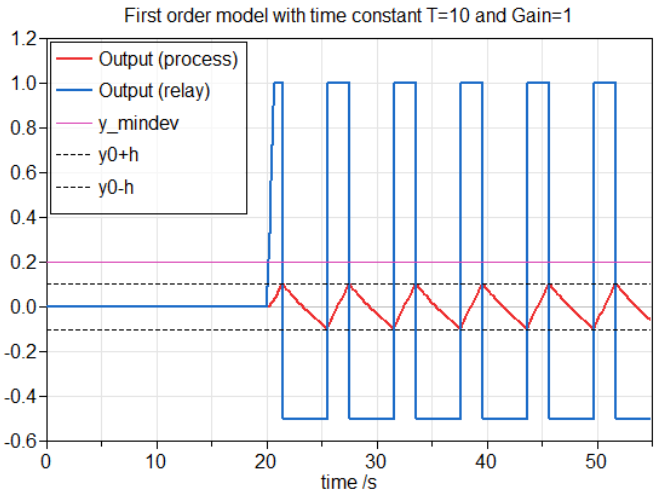

Figure 5.1: Illustrates the case when the process output cannot reach the desired level, y_{mindev}. The blue curve is the relay output, the red is the process output.

The problem explained above was observed while working with the auto-tuner. Two problematic processes are Natural Circulation Boiler and the part of the aircraft example handling the yaw. A discussion regarding those can be read in Section 5.2.

User perspective

From a user perspective it might be a lot to focus on. First of all the initialization is important if the experiment is going to provide good result. That includes decisions about when the experiment is in steady-state as well as parameter settings. The

latter includes constraints on the experiment and parameters used to initiate the experiment. Some default values are set (see Table 6.8) but they may be different depending on what process one is working with. Two parameters that often has to be modified are $uMax$ and $uMin$. They decide in what region the control signal is allowed to be in. In Chapter 4 these parameters varied for all the experiments.

One thing that may seem strange is that, for some processes, the initialization needs a controller to reach a steady-state level. This means that one has to come up with controller parameters before the experiment, which is supposed to generate good parameters, has run. The parameters used for reaching steady-state does not need to be perfect, the only purpose is reaching steady-state. And when it does, the experiment can begin and new parameters will be generated. The usage of a controlled input is only needed together with processes that are unstable and where steady-state is not reached in another way. This is definitely something that would be wise to keep working on.

The user gets a documentation about how the auto-tuner works through an information tab in Dymola. It describes what kind of parameters that are available for change as well as general information about the auto-tuner. After the experiment a file is created and a window pops up with experimental result. The information is about the estimated model, the PID parameters and some experiment parameters.

Some users might prefer another tuning rule than AMIGO. Therefore the experiment presents the estimated model as well in order to give the user different choices. Also, there is a comparison that continues after the experiment in order to get a hint if the estimated model is good or not. Additional information would be a bode plot and a Nyquist curve. However, those features are not included in the implementation.

5.3 Discussion on results

Performance

The tuning rule used in this thesis has been AMIGO. Since the experiment generates an estimated model another preferable tuning rule can be chosen. Some tuning rules are listed in Chapter 2, section "tuning". Another rule to choose might for instance be λ-tuning. It relates, as well as AMIGO, the PID parameters to K_p, K_v, L and T given from the estimated model, which makes it a good choice as well. When selecting tuning rule one should know that different tuning rules belongs to a certain specification. The advantage with the AMIGO rule is that it takes into account robustness by putting a constraint on the maximum sensitivity, $M_s < 1.4$. The constraint is related to the model that the AMIGO rule is used on, in this case the estimated model. If the model is a good approximation of the real process, it means that the sensitivity is less than 1.4. However, for the processes shown in Chapter 4 the approximations were not always good. Even if that was the case the sensitivity never became larger than 1.46. More than robustness the controllers proved

to be fast. That was given through the rise time numbers. It also relates to a large bandwidth.

The comparison between the estimated model and the real process has not always proved to be good. One should know that the estimated model is of first order but the real process can be of much higher order. However, the PID parameters turned out to perform well even for the less accurate models. It means that the comparison test is not always telling the whole truth about the outcome from experiment, but it may be good as a guideline. One should at least be observant in those cases where the estimated model is far away from the process.

Another thing that makes the comparison test sometimes give strange result is for processes where the estimated model is an ITD model. If τ is less than α the model is estimated as an ITD. But a pure ITD model is when τ is 0. If $0 \leq \tau \leq \alpha$ the equality

$$\frac{u_{on}}{u_{off}} = -\frac{t_{off}}{t_{on}}, \tag{5.1}$$

does not hold. This means that the integral

$$y(t) = k_v \int_0^{t_p} (u(\tau) - u(0))\,\mathrm{d}\tau \tag{5.2}$$

does not becomes zero. It means that an offset is applied to the estimated model for each period, which makes the comparison look ugly. Unfortunately that is something one can't get rid of. The estimated PID parameters may be good anyway.

Pre-defined methods

The already existing tool in Modelica LinearSystems 2 provides a full linear analysis. With that tool one can have a linear model of the process. You may therefore ask yourself why that is not enough? Unfortunately in most cases the generated linear model is of high order. Most tuning rules require a low order model to calculate PID parameters. In that sense model reduction must be applied to use a preferable tuning rule. Model reduction by balanced truncation is a way to do it. That method use information about Hankel singular values. A small Hankel singular value indicates that a state is weakly controllable/observable. Consequently it can be truncated without creating a big impact on the input-output relation [Glad and Ljung, 2003]. Another way is to use Skogestad's half rule to achieve a FOTD model [Skogestad, 2003]. That was done in the steam temperature control example, before it was tuned with AMIGO rule. In that case the model turned out to be of order 3 (see appendix). The IAE-values obtained from the auto-tuning experiment and from Modelica Linearsystems 2 were about the same. For the other cases Skogestad's half rule did

not give desirable results. It may be due to that the other processes were unstable. In [Skogestad, 2003] only stable processes are considered together with integrated processes.

To reduce the unstable models given in this thesis balanced truncation can be applied. Transfer functions for processes in Chapter 4 are listed in appendix. As one can see they are all of high order and some of them are unstable. For those that are unstable they need to be separated into a stable part and an unstable part before model reduction is applied, i.e

$$P(s) = P_{stable}(s) + P_{unstable}(s). \tag{5.3}$$

After model reduction is performed on the stable part one adds them both together. Due to the fact that some processes in appendix contain more than one unstable pole they can't be reduced to a lower order than number of unstable poles, since the unstable poles contain a lot of energy.

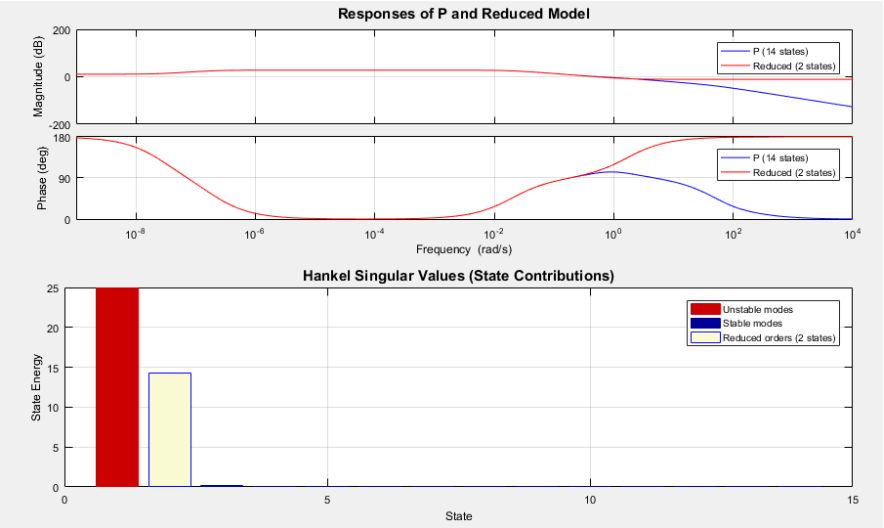

Figure 5.2: Balanced truncation on yaw.

With that reason a tuner from control system toolbox in Matlab was chosen instead. It worked for all of the processes except the Natural boiler example and the vertical velocity, in the aircraft example.

Experiments

In Chapter 4 some experiments were made. Overall the controllers based on the auto-tuner turned out to perform better than the pre-tuned controller as well as the PID tuner in Matlab.

The purpose with the first experiment was to present how the auto-tuner works for a linear, stable system. The process was in that case of second order with time delay. The estimated model and the real process followed each other well and the tuning result was good compared to the other result. Based on Figure 4.4 and Table 4.2 can it be read out that the Matlab version was more aggressive while the other two were more robust. The results were similar to those that were tuned with the AMIGO rule. In perspective of the robustness, the generated controller was a little bit worse. However, good results from the auto-tuner may depend on a good estimated model, presented in Figure 4.3. Due to the fact that the process was linear the estimated model would be good for other points in space as well.

In the first thermodynamic process, i.e. the steam temperature control example, the purpose was to present how the auto-tuner worked for a non linear stable system. In that experiment the model following, presented in Figure 4.7, was not as good as in the first experiment. One explanation is that the real process is of much higher order compared to the estimated model. However, the controller handled a step change as well as a load disturbance better than the current controller and the controller obtained from Matlab control system toolbox. Skogestad gave similar result as the auto-tuner. In fact the performance of the auto-tuner was much more aggressive and robust compared to the current.

One thing to observe through Table 4.4 is that τ was estimated to 0.1. That number is critical, since it is the limit (for default values of α and β) between a FOTD model and an ITD model. Due to that fact the alpha value was changed, in order to compare the impact of another model estimation. However, Table 4.4 shows that the result became similar in this case. The ITD model was little more robust and some percentage faster.

Considering the second thermodynamic process, i.e. the natural circulation boiler, this was an example of a non linear and unstable process. When working with it we also concluded that the dynamics were very fast, which in turn introduced us to a problem regarding the convergence. Observe that the process is of high order in this case. By implementing the boolean flag that was discussed in Chapter 3 we were able to force the experiment to check for convergence even in this situation. Since this is not how the auto-tuner normally works it was not clear whether it had any impact on the results or not. The resulting control behaviour presented in Figures 4.15 and 4.13 speaks for itself. When using the PID parameters generated by the auto-tuner the response to both a step change and a constant load disturbance is a lot more satisfactory. For instance it was necessary to change the scale in the lower plot in Figure 4.15 in order to be able to see the impact from the disturbance. Although the generated controller is more aggressive than the current, the control

signal in Figure 4.14 proves that this is not a problem. Additionally the data concerning rise time, overshoot and IAE in Table 4.6 also proves the auto-tuner's good performance.

In experiment 4, i.e. the aircraft experiment, the purpose was to investigate how well the auto-tuner worked in a multivariable system. Additionally it is an interesting example since the area of aerodynamics is not an area where auto-tuners traditionally are being used. When working with the system the auto-tuner experiment was made on one loop at the time in comparison with all running in parallel. Overall it turned out that the values for roll and yaw differed in the two experiments, while the vertical velocity almost remained unchanged. This indicates possible cross-couplings in the system. Relative gain array (RGA) is a good measure for interactions in a system [Glad and Ljung, 2003]. If the system is represented on the form

$$\begin{bmatrix} y_{roll} \\ y_{velocity} \\ y_{yaw} \end{bmatrix} = \mathbf{P} \begin{bmatrix} u_{velocity} \\ u_{yaw} \\ u_{roll}, \end{bmatrix} \tag{5.4}$$

where $\mathbf{P}$ is the transfer matrix, the RGA is possible to calculate. Under static conditions the RGA-matrix of the system, at the linearized point, is given by:

$$RGA = \begin{bmatrix} 0 & 0.6362 & 0.3638 \\ 1 & 0 & 0 \\ 0 & 0.3638 & 0.6362 \end{bmatrix}, \tag{5.5}$$

where

$$\begin{aligned} u_1 &= u_{velocity} \qquad u_2 = u_{yaw} \qquad u_3 = u_{roll} \\ y_1 &= y_{roll} \qquad y_2 = y_{velocity} \qquad y_3 = y_{yaw}. \end{aligned}$$

By observing equation (5.5) one can see that there is a cross-coupling between yaw and roll while the vertical velocity is independent, based on static properties. However, even if the system contained interactions the auto-tuner generated good result in both cases when dealing with a constant load disturbance. In the step response case they both gave a fast response, but a large overshoot. For another tuning that may be different. It is possible that the overshoot is a result of a too large step change. However, the result was good when the purpose was to reference follow a ramp change.

Another interesting thing to observe are how bad the estimated models follow the true models in Figures 4.19 and 4.29. A suggestion is to estimate higher order models, for instance SOTD models (Second Order Time Delay). That would probably give a better model following, but not necessarily better control performance. It could have caused another result of the large overshoot and there aggressiveness. A large overshoot may cause problems. Therefore restrictions on fast set point

changes are necessary in this case since aggressive controllers are aggressive will cause problems if noise were added. Aggressive controllers are generally noise sensitive and are therefore not preferable in that sense. Therefore it matters in what type of environment the parameters are used in. In this case it is in a noise free simulation environment.

An interesting thing that was not presented in the result part was how the controller, obtained from the experiment, handled a larger set-point change. In that case, the result was not very satisfactory for roll. A larger set-point change caused an oscillating behavior as Figure 5.3 shows. It may be since the estimated model is only valid in a region around the stationary point, where the experiment is performed. A step change that goes outside of that region makes the model invalid and though also the controller that is based on the model. However, when the goal was to track a ramp the result was more satisfactory as one can see in the result. One can imagine that a large step change is a heavy stress on a process. A more common thing in reality would be a ramp as illustrated in the result part.

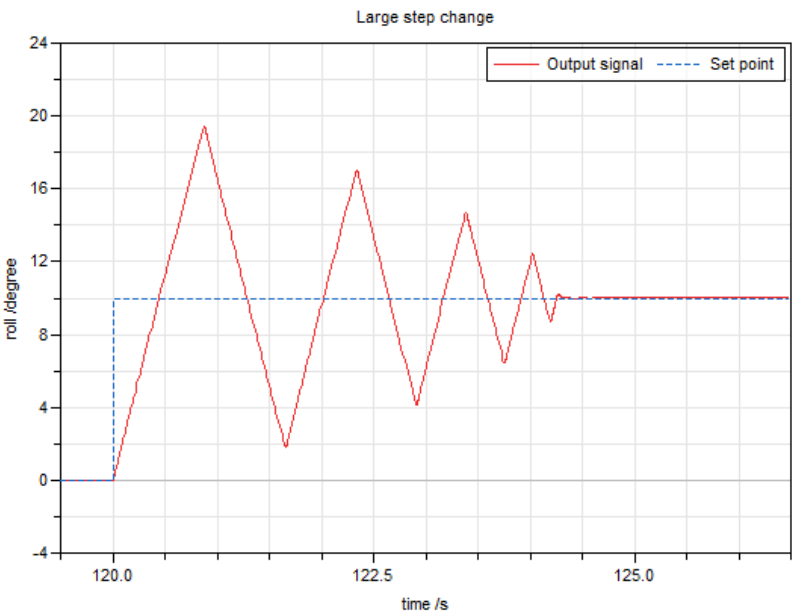

Figure 5.3: Shows how a large set-point change affects the results when working with the roll angle (aircraft model).

5.4 Further development

Although the auto-tuner performs good in more or less every test it is advisable to continue the development with it over time.

As stated previously the models we have used for evaluating the auto-tuner has been borrowed from already existing Modelica libraries. Hence, conclusions regarding the difficulties when using the auto-tuner while developing a new model cannot be drawn. For instance, our idea of making use of Ziegler and Nichols method in order to reach steady-state before the experiment is started has not been tested properly. One thing to investigate is whether it is possible or not to let the user start the simulation and then decide when to start the experiment manually. Ideally, the user should not need to think about this issue at all.

One feature that was mentioned in Section 4.1 was regarding a script function that generated Figure 4.2 and 4.3. The function does not always work since it is dependent on that the relay and process output both are aligned at the same level, more or less. When working with the steam temperature control process the function did not work very well since the set point for the process output was just over 600 while the relay output were between 1 and 0. In order to make this function helpful regardless of what process is being used, one has to come up with a solution to this.

Regarding systems with several control loops it might be a good idea to investigate it more. During the thesis it was tested on only one process. When testing it on several processes drawbacks might turn up. For instance, a systematic work procedure when dealing with systems of this type, would be wise.

For processes of higher order the comparison between the estimated model and the real model is pretty bad. For such processes it might be a good idea to estimate a higher order model, for instance a SOTD model. In order to perform it a modification of the implementation is needed in the part concerning Calculation of parameters, described in Chapter 3. A method for this purpose is described in [Berner, 2015].

Bibliography

Åström, K. J. and T. Hägglund (1984). *Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins*. Automatica, p. 645, SwePub, EBSCOhost, viewed 9 February 2017.

Åström, K. J. and T. Hägglund (2004). *Revisiting the Ziegler–Nichols step response method for PID control*. Journal Of Process Control, 14, pp. 635-650, ScienceDirect, EBSCOhost, viewed 23 March 2017.

Åström, K. J. and T. Hägglund (2006). *Advanced PID Control*. pp. 64-69, 73, 76-78, 159-163, 186-189, 225-241. ISA - The Instrumentation, Systems and Automation Society.

Åström, K. J. and R. Murray (2008). *Feedback Systems : An Introduction For Scientists And Engineers*. n.p.: Princeton, N.J. : Princeton University Press, cop. 2008, pp. 23-24, 110-124, 155-189, Library catalogue (Lovisa), EBSCOhost, viewed 8 February 2017.

Berner, J. (2015). *Automatic Tuning of PID Controllers based on Asymmetric Relay Feedback'*. Licentiate Thesis, SwePub, EBSCOhost, viewed 8 February 2017.

Berner, J., K. J. Åström, and T. Hägglund (2014). *Towards a New Generation of Relay Autotuners*. Paper presented at 19th IFAC World Congress, 2014, Cape Town, South Africa, viewed 8 February 2017.

Berner, J., K. J. Åström, and T. Hägglund (2016a). *Asymmetric relay autotuning – Practical features for industrial use*. Control Engineering Practice, 54, pp. 231-245, ScienceDirect, EBSCOhost, viewed 10 April 2017.

Berner, J., K. J. Åström, and T. Hägglund (2016b). *Improved relay autotuning using normalized time delay*. 2016 American Control Conference (ACC), p. 1869, Publisher Provided Full Text Searching File, EBSCOhost, viewed 10 February 2017.

Glad, T. and L. Ljung (2003). *Reglerteori : Flervariabla Och Olinjära Metoder*. n.p.: Lund : Studentlitteratur, 2003 (Lund : Studentlitteratur), Library catalogue (Lovisa), EBSCOhost, viewed 25 April 2017.

Luyben, W. L. (2001). *Getting More Information from Relay-Feedback Tests*. Industrial & Engineering Chemistry Research, 40, 20, pp. 4391-4402, Science Citation Index, EBSCOhost, viewed 8 February 2017.

Modelica-association (2014). *A Unified Object-Oriented Language for Systems Modeling*. Language Specification Version 3.3 Revision 1, viewed 23 February 2017.

Panagopoulos, H., K. J. Åström, and T. Hägglund (2002). *Design of PID controllers based on constrained optimisation.* IEE Proceedings - Control Theory and Applications, 149:1, pp. 32–40., viewed 3 April 2017.

Rivera, E. D., M. Morari, and S. Skogestad (1986). *Internal model control 4. PID controller design*. Ind. Eng. Chem. Proc. Des. Dev., 25, pp. 252-262, viewed 4 April 2017.

Skogestad, S. (2003). *Simple analytic rules for model reduction and PID controller tuning*. Journal of Process Control, vol. 13, pp. 291-309. Available from: 10.1016/S0959-1524(02)00062-8. 25 April 2017.

Visioli, A. (2006). *Practical PID Control*. [Electronic Resource], n.p.: London : Springer-Verlag London Limited, 2006., pp. 1-18, Library catalogue (Lovisa), EBSCOhost, viewed 8 February 2017.

Ziegler, J. and N. B. Nichols (1942). *Optimum Settings for Automatic Controllers*. trans.ASME 64:1,viewed 9 February 2017.

6

Appendix

6.1 Amigo tuning rule

Model	PI parameters	PID parameters
FOTD	$\mathrm{K} = \frac{0.15}{K_p} + (0.35 - \frac{LT}{(L+T)^2})\frac{T}{K_p L}$ $T_i = 0.35L + \frac{13LT^2}{T^2+12LT+L^2}$	$K = \frac{0.2L+0.45T}{K_p L}$ $T_i = \frac{0.4L+0.8T}{L+0.1T}L$ $T_d = \frac{0.5LT}{0.3L+T}$
ITD	$K = \frac{0.35}{k_v L}$ $T_i = 13.4L$	$K = \frac{0.45}{k_v L}$ $T_i = 8L$ $T_d = 0.5L$

Table 6.1: Tuning table AMIGO [Åström and Hägglund, 2006]

6.2 Ziegler and Nichols

Workflow

- Increase the proportional gain in a simple proportional controller until oscillation occurs.
- The gain that creates the oscillation is then called the critical gain k_c and the time period on the oscillation is called the critical time period t_c.
- The parameters in the controller are then given by table 6.2.

Parameter/Controller	P	PI	PID
K	$0.5K_c$	$0.45K_c$	$0.6K_c$
T_i	∞	$\frac{t_c}{1.2}$	$\frac{t_c}{2}$
T_d	0	0	$\frac{t_c}{2}$

Table 6.2: Tuning table Ziegler and Nichols, [Ziegler and Nichols, 1942].

6.3 Transfer functions

In the following section transfer functions of the experimental processes are stated. Since many of them are of high order they are expressed as two polynomials, A(s) and B(s), i.e.

$$P(s) = K\frac{B(s)}{A(s)} \tag{6.1}$$

The denominator is the A polynomial and the numerator is the B polynomial and K is the process gain. Furthermore each polynomial are factorized, i.e.

$$B(s) = B_0B_1\cdots B_n = (s+b_0)(s+b_1)\cdots(s+b_n), \tag{6.2}$$
$$A(s) = A_0A_1\cdots A_m = (s+a_0)(s+a_1)\cdots(s+a_m), \tag{6.3}$$

where m is the order of the denominator and n is the order of the numerator.

Steam Temperature Control

The parameters in equation 6.1 and 6.3 for the steam temperature control example are given through Table 6.3. Observe that the transfer function is a linear approximation, only valid around the linearized point.

Parameter	**Value**	a_i	**Value**	b_j	**Value**
K	-298.63	a_0	6762	b_0	4454
		a_1	11.19		
		a_2	1		

Table 6.3: Steam temperature control

Natural Circulation Boiler

The parameters in equation 6.1 and 6.3 for the Natural Circulation Boiler example are given through Table 6.4. Observe that the transfer function is a linear approximation, only valid around the linearized point.

Parameter	**Value**	a_i	**Value**	b_j	**Value**
K	0.017052	a_0	$-6.313 \cdot 10^5$	b_0	$-1.31 \cdot 10^5$
		a_1	130.7	b_1	130.5
		a_2	6.176	b_2	6.214
		a_3	4.417	b_3	0.804
		a_4	4.108	b_4	0.642
		a_5	1.693	b_5	-0.2194
		a_6	0.7938	b_6	$-7.897 \cdot 10^{-6}$
		a_7	0.7017	b_7	1.4090 + 0.2995i
		a_8	0.1774	b_8	1.4090 - 0.2995i
		a_9	$1.806 \cdot 10^{-8}$	b_9	4.3165 + 0.5078i
		a_{10}	0.6710 + 17.9318i	b_{10}	4.3165 - 0.5078i
		a_{11}	0.6710 - 17.9318i		

Table 6.4: Natural Circulation Boiler

Flight Manuver

The parameters in equation 6.1 and 6.3 for the vertical velocity, yaw and roll are given through Tables 6.5, 6.6 and 6.7. Observe that the transfer functions are a linear approximation, only valid around the linearized point.

Parameter	Value	a_i	Value	b_j	Value
K	1.2219	a_0	0.00751	$b_{0...1}$	0
		a_1	$1.523 \cdot 10^{-8}$	b_2	-31.57
		a_2	$10^{-5} \cdot$ (0.0437 + 0.1482i)	b_3	3.12
		a_3	$10^{-5} \cdot$ (0.0437 - 0.1482i)	b_4	0.08
		a_4	3.4920 + 2.1368i		
		a_5	3.4920 - 2.1368i		

Table 6.5: Vertical Velocity

Parameter	Value	a_i	Value	b_j	Value
K	-45.135	a_0	6515	b_0	6519
		$a_{1...6}$	1000	b_1	1005
		a_7	225.6	b_2	990.2
		a_8	51.99	b_3	205.7
		a_9	1.165	b_4	0.6757
		a_{10}	-0.01937	b_5	$-2.706 \cdot 10^{-8}$
		a_{11}	$1.946 \cdot 10^{-7}$	b_6	9.9550 + 0.1949i
		a_{12}	85.0500 + 34.7922i	b_7	9.9550 + 0.1949i
		a_{13}	85.0500 - 34.7922i	b_8	1004
				b_9	1000
				b_{10}	87.4 + 37.11i
				b_{11}	87.4 - 37.11i

Table 6.6: Yaw

Parameter	Value	a_i	Value	b_j	Value
K	-291.64	a_0	321.1	$b_{0...5}$	999.9
		$a_{1...6}$	1000	b_6	1.23
		a_7	1.497	b_7	0.1327 + 0.2983i
		a_8	-0.003252	b_8	0.1327 - 0.2983i
		a_9	0.0187 + 0.3036i	b_9	50.4 + 14.79i
		a_{10}	0.0187 - 0.3036i	b_{10}	50.4 - 14.79i
		a_{11}	50.6500 + 14.7844i		
		a_{12}	50.6500 - 14.7844i		

Table 6.7: Roll

6.4 Default values

General		**Initialization**	
Parameter	Default value	Parameter	Default value
Experiment parameters		**Init Controller**	
running_mode	Normal	steadyStateInit	false
alpha	0.1 uInit	0	
beta	0.6	k	2
uMax	10	Ti	50
uMin	-uMax	**Settings**	
h	0.1	tol	0.001
y_maxdev	12h	init_mode	Constant input
y_mindev	2h	ss_mode	Unfixed
γ	2	der_check	30
γ	2	u_const	0
ε	0.01	t_ss	70
force_conv	false		
nbr_switch_force	7		

Table 6.8: Default values

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
June 2017

Document Number
ISRN LUTFD2/TFRT--6032--SE

Author(s)
Markus Björk
Robin Levenhammar

Supervisor
Håkan Runvik, Modelon AB
Josefin Berner, Dept. of Automatic Control, Lund University, Sweden
Tore Hägglund, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Relay Auto-tuners in Modelica

Abstract
The PID controller is by far the most popular controller for process control in industry today [Åström and Murray, 2008]. The combination of its relatively simple usage and its satisfactory results within several areas has gained its popularity [Visioli, 2006]. However, the tuning of the controller is crucial. An automatic tuning method was developed by [Åström and Hägglund, 1984] in the mid-eighties and auto-tuners has since then been diligently used in the process industry.

This thesis will focus on the implementation of an auto-tuner based on asymmetric relay feedback in the modeling language Modelica. Also, creating a good work flow and providing valuable information for the user. The auto-tuner provides an estimated low order model together with parameters for a PID controller, based on the AMIGO tuning rules.

After an introduction the report begins with a chapter containing a theoretical background concerning the PID controller, tuning methods, asymmetric relay autotuners and Modelica/Dymola. This is followed by an explanation of the method used, including a step-by-step description.

The report ends with chapters dealing with results, discussions and comments about future development. The result chapter is divided into different sections, each dealing with one of the processes used for testing. Every section presents the experimental results together with comparisons against other parameter set-ups. The last chapter contains a discussion regarding the overall results, both in terms of usage and performance.

The results, in terms of responses to load disturbances and step changes, were satisfactory for all processes that were tested in the thesis, although some processes required more preparatory work than others. As long as the process is in steadystate when the experiment begins, the resulting PID parameters turns out to work very good in comparison to the ones being used in the models today. Despite the good performance there are smaller things that needs to be further developed. For instance, a feature that is supposed to generate two plots automatically needs to be improved.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-93

Recipient's notes

Security classification