ISSN 0280-5316
ISRN LUTFD2/TFRT--5800--SE

Implementation of an Autotunable Decoupling TITO Controller

Alfred Theorin

Department of Automatic Control
Lund University
July 2007

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

Document name
MASTER THESIS

Date of issue
July 2007

Document Number
ISRNLUTFD2/TFRT--5800--SE

Author(s)
Alfred Theorin

Supervisor
Michael Kwapisz and Mikael Petersson at ABB in Malmö.
Tore Hägglund at Automatic Control in Lund (Examiner)

Sponsoring organization

Title and subtitle
Implementation of an Autotunable Decoupling TITO Controller. (Implementering av en automatinställningsbar särkopplande TITO-regulator)

Abstract
In the process industry general TITO systems, i.e. systems with two inputs and two outputs, are usually assumed to be two separate systems. If the system is strongly coupled, the performance is generally poor, but usually nothing is done about this since there are no automatic means to improve it. The aim of this master's thesis is to develop a module with a completely automated design procedure which improves the performance of coupled TITO systems. For this to be possible an automated TITO system identification will be required, as well as automated decouple filter design and PID tuning for decoupled systems. Finally, to confirm that the developed module really works it is tested on a real process.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
79

Recipient's notes

Security classification

Contents

List of Figures

Acknowledgements

There are several persons who have helped me with the work on this thesis. First of all, I would like to thank professor Tore Hğglund for always being encouraging, knowledgeable and supportive. I would also like to thank Olof Garpinger for helping me to understand the PID design algorithm and coming with ideas to improve it. Also, thanks for lending me the water tank process.

For supporting me with everything I needed during development and testing I would like to thank the ACE division of ABB. Especially I would like to thank Michael Kwapisz for the many interesting discussions and ideas, and Mikael Petersson and Bengt Hansson for supporting me with their deep knowledge of the system.

Finally I would like to thank everyone who provided me with feedback on the report. Thank you!

Chapter 1

Background

Two input two output systems, or TITO systems for short, are systems where there are two properties to control and both control signals may affect both process values. TITO systems are usually assumed to be two separate single input single output systems, or SISO systems for short, which are controlled by separate controllers. This is easy to implement and there are no drawbacks with this approach when the couplings are relatively small. However for large couplings the performance may be very poor. The easiest way to check if a system is coupled is to apply a setpoint change on one process value and see if this causes variations on the other property. If it does, the system is coupled.

There are many systems in the industry where the couplings are too big for SISO control to be appropriate, still it is used. For these systems TITO control should be considered. One TITO control approach is to add a $2x2$ filter between the controllers and the process. Correctly designed the filter decreases or even cancels the couplings, making the TITO system appear as two completely separate SISO systems to the controllers.

In a factory there usually are thousands of controllers, so manual tuning of controller parameters and decouple filters is not an option. This master's thesis aims to implement a module with automated methods for tuning of both the $2x2$ filter and the controllers.

Chapter 2

Theory

2.1 Modelling

A TITO process can be modelled as a $2x2$-matrix with transfer functions as elements, see equation (2.1). If the system is properly connected the larger gains are in the diagonal and the couplings are described by p_{12} and p_{21}. If the coupling gains are much smaller than the diagonal gains there is no need for decoupling and a SISO approach should be used. Typical step responses for a coupled TITO system can be seen in figure 2.1.

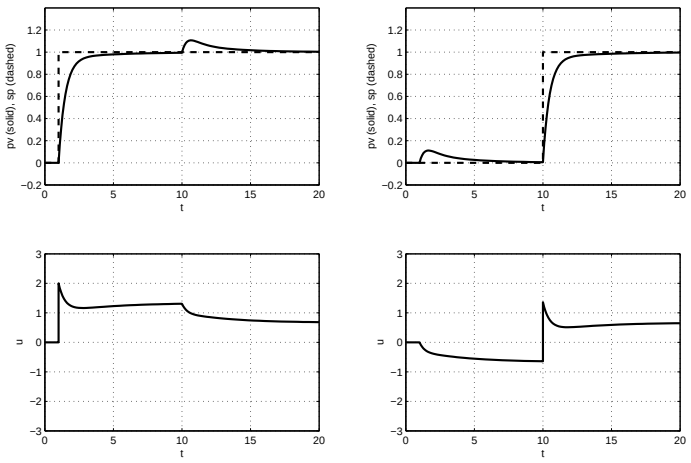

Figure 2.1: *The step responses for a simple TITO system with first order transfer functions. When the setpoint is changed for one process value, the other starts to vary, thus the system is coupled.*

$$\begin{cases} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = P \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \\ P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}, \\ p_{ij} = \frac{B_{ij}(s)}{A_{ij}(s)} e^{-sL} \end{cases} \tag{2.1}$$

where $A_{ij}(s)$ and $B_{ij}(s)$ are polynomials.

The decouple filter for a TITO system can also be described by a 2x2-matrix with transfer functions as elements. With the decouple filter D added between the controllers and the process the complete system appears as PD to the controllers. If D decouples the system perfectly, PD is diagonal, see equation (2.2).

$$PD = L \tag{2.2}$$

where L is diagonal.

From linear algebra we have

$$A^{-1} = \frac{1}{\det(A)} \text{adj}(A) \tag{2.3}$$

From equation (2.2) and (2.3) we get

$$D = P^{-1}L = \frac{1}{\det P} \text{adj}(P)L = \text{adj}(P)\Lambda \tag{2.4}$$

where $\Lambda = \frac{1}{\det(P)} L$ and thus also is diagonal.

In [1] it is suggested to use the adjunct of P and then design the decouple filter by altering the diagonal matrix, Λ. Since Λ does not have to be implemented the only restrictions on Λ is that it must lead to an implementable decouple filter.

Since most target processes are well approximated by first order transfer functions with dead time, or FOWDT for short, they will be used for elements both in the process model and in the decouple filter. With FOWDT transfer functions the diagonal elements of the decoupled system will be the sum of two second order filters with dead time, denoted by G_{ij} in equation (2.5).

$$
\begin{aligned}
PD &= \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix} \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} \\
&= \begin{pmatrix} p_{11}d_{11} + p_{12}d_{21} & 0 \\ 0 & p_{21}d_{12} + p_{22}d_{22} \end{pmatrix} \\
&= \begin{pmatrix} G_{11} + G_{12} & 0 \\ 0 & G_{21} + G_{22} \end{pmatrix}
\end{aligned} \tag{2.5}
$$

2.2 Decouple Filter Design

Various assumptions may be made when designing a decouple filter, some result in simpler decouple filters with limited performance while other result in more complex decouple filters with potentially better performance. Common for all of them is that they require a process model and the better the model is the better the decoupling will be. A more complex decouple filter requires more information about the process, thus a more complex system identification is needed.

The most simple decouple filter is the static decouple filter, which only has static gains as filters, see [2]. It has very limited performance but handles inaccurate models pretty well. Beside the static decouple filter the two decouple design methods proposed in [1] and a static decouple filter with dead time have been implemented and evaluated.

2.2.1 Static Decouple Filter

The static decouple filter is a very simple decouple filter without any dynamics, i.e. the decouple filter's elements are simply static gains, see equation (2.6).

$$
D = P^{-1}(0) = \frac{1}{\det P(0)} \begin{pmatrix} p_{22}(0) & -p_{21}(0) \\ -p_{12}(0) & p_{11}(0) \end{pmatrix} \tag{2.6}
$$

Designing a static decouple filter for a process only requires the static gains of the process transfer functions since $P^{-1}(0) = P(0)^{-1}$.

The performance of a static decouple filter is limited to static states. This means that for instance the transients upon setpoint changes will not be improved noticeably. The big advantage for this kind of decouple filter is when you manage the control signals manually, i.e. Manual mode. Without a decouple filter both control signals have to be altered if one property is to be changed and the other is to remain constant. With a static decouple filter only one control signal has to be altered to accomplish this, making it easier for the operator to control the system.

2.2.2 FOWDT Decouple Filter

A method for designing a more advanced decouple filter is proposed in chapter 3.2 in [1] and will be referred to as the FOWDT decouple filter since the elements are first order filters with dead time, see equation (2.7). The method chooses Λ in equation (2.2) to minimize unnecessary dead times and dynamics.

$$p_{ij} = \frac{K_{ij}}{T_{ij}s + 1} e^{-sL_{ij}} \tag{2.7}$$

First Λ is chosen to be the identity matrix, which means $D = \text{adj}(P)$. By changing Λ_{ii}, common dead times, poles and zeroes of D_{i1} and D_{i2} are removed.

When poles are removed there is a risk of making the decouple filter impossible to implement, i.e. having a non causal filter with more zeroes than poles. With a FOWDT model there are no zeroes so this cannot happen.

Example 2.2.1. *Consider the process*

$$P = \begin{pmatrix} \frac{1}{3s+1}e^{-2s} & \frac{0.5}{2s+1}e^{-2s} \\ \frac{1}{s+1}e^{-s} & \frac{1}{s+1}e^{-3s} \end{pmatrix}$$

Begin with $\Lambda = I$*,*

$$\text{adj}(P)\Lambda = \text{adj}(P) = \begin{pmatrix} \frac{1}{s+1}e^{-3s} & -\frac{0.5}{2s+1}e^{-2s} \\ -\frac{1}{s+1}e^{-s} & \frac{1}{3s+1}e^{-2s} \end{pmatrix}$$

Common dead times are removed with

$$\Lambda = \begin{pmatrix} e^{s} & 0 \\ 0 & e^{2s} \end{pmatrix}$$

$$\text{adj}(P)\Lambda = \begin{pmatrix} \frac{1}{s+1}e^{-2s} & -\frac{0.5}{2s+1} \\ -\frac{1}{s+1} & \frac{1}{3s+1} \end{pmatrix}$$

Finally the common pole in column 1 is removed with

$$\Lambda = \begin{pmatrix} (s+1)e^{s} & 0 \\ 0 & e^{2s} \end{pmatrix}$$

$$D = \text{adj}(P)\Lambda = \begin{pmatrix} e^{-2s} & -\frac{0.5}{2s+1} \\ -1 & \frac{1}{3s+1} \end{pmatrix}$$

2.2.3 Approximate FOWDT Decouple Filter

This decouple filter was proposed in [1] for reduction of dynamics in a FOWDT decouple filter. For each column the time constants T_{1i} and T_{2i} are denoted T_l and T_s where $T_l \geq T_s$. One of the transfer functions will be

$$\frac{K_1}{T_s s + 1} e^{-sL_1} \tag{2.8}$$

The other transfer function will be approximated by using

$$\frac{K_2}{T_l s + 1} e^{-sL_2} \approx \frac{K_2}{(T_s s + 1)((T_l - T_s)s + 1)} e^{-sL_2} \tag{2.9}$$

Thus the pole $T_s s + 1$ will be common and can be canceled by modifying Λ. The approximation results in a decouple filter which has less dynamics since at least two elements will be static gains, possibly with dead times.

2.2.4 Static Decouple Filter With Dead Times

One way to extend the static decouple filter in section 2.2.1 is to add dead times, see equation (2.10). Unnecessary dead times can then be removed in the same way as in the FOWDT decouple filter, see section 2.2.2. By doing this the decouple filter will be able to handle differences in dead times better than the ordinary static decouple filter without adding dynamics to the decouple filter.

$$p_{ij} = K_{ij} e^{-sL_{ij}} \tag{2.10}$$

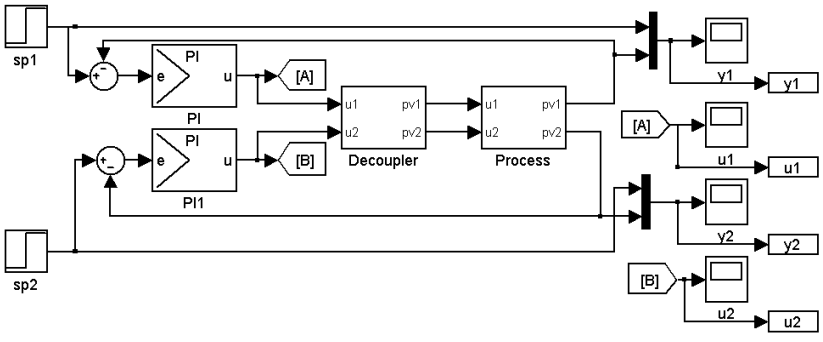

Figure 2.2: *SimuLink model for TITO simulations.*

2.2.5 Decouple Filter Simulations

To decide which decouple filters are worth using and when to use which, simulations have been done in SimuLink. The processes used are symmetric and controlled by two identical PI-controllers.

For the process in equation (2.11) simulation results are shown in figure 2.3, 2.4 and 2.5.

$$P = \begin{pmatrix} \frac{1}{s+1} & \frac{0.5}{2s+1} \\ \frac{0.5}{2s+1} & \frac{1}{s+1} \end{pmatrix} \tag{2.11}$$

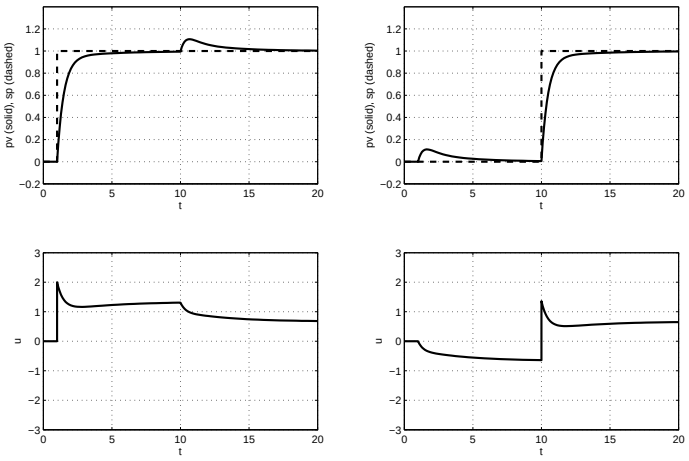

Figure 2.3: *The step responses for the system in equation (2.11) without a decouple filter. Typical effects of coupling can be seen at the setpoint changes.*

With a static decouple filter the coupling at setpoint changes is not affected notably, simply because the setpoint change is not of static nature. What is more interesting is to look at the control signals. Setpoint changes without a static decouple filter causes offsets in both control signals, see figure 2.3. With a static decouple filter there is only a transient on the other control signal which is a great advantage when controlling any or both of the values manually, i.e. Manual mode.

With a FOWDT decouple filter there are no signs of coupling at all. This is because the model does not suffer from any mismatches which means that the process transfer function matrix is completely diagonalized by the decouple filter. A disadvantage with this decouple filter is that it is a lot harder to find controllers which work well, compared to without a decouple filter or with a static decouple filter.

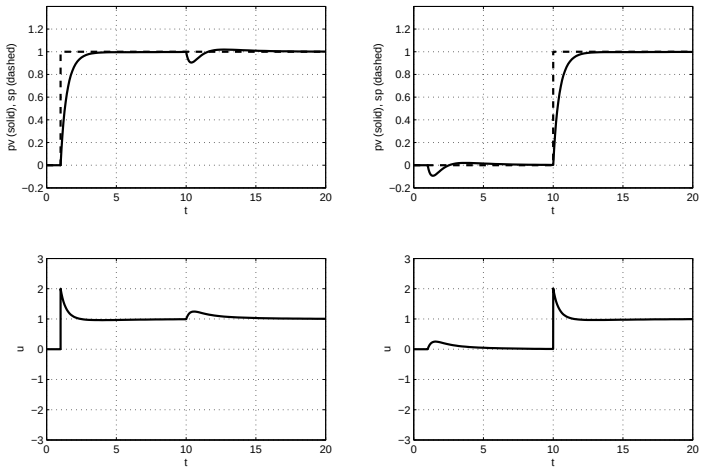

Figure 2.4: *Step responses for the system in equation (2.11) with a static decouple filter.*

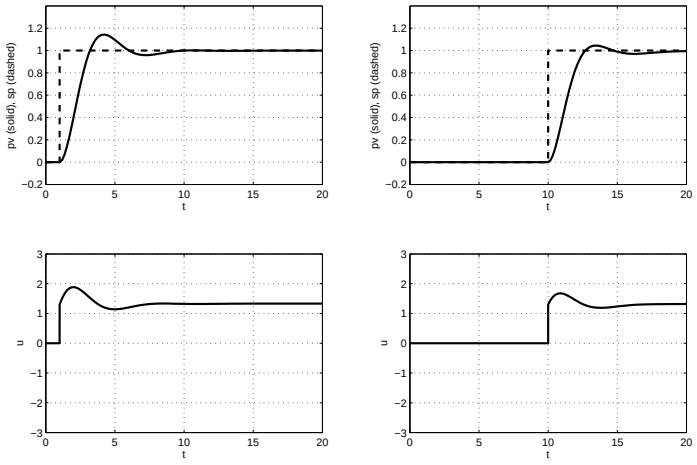

Figure 2.5: *Step responses for the process in equation (2.11) with a FOWDT decouple filter. Since the process is a first order model without any mismatch the process is completely decoupled.*

2.3 Controller Design

Adding a decouple filter increases the complexity of the system remarkably. An algorithm for PID design for this kind of systems is proposed in [1]. It is time consuming and typically takes more than one minute to execute on a 2.66GHz Intel P4 CPU.

The algorithm is unsuitable for embedded systems with limited resources and relatively low clock frequency. It is still preferable since it is the only available automatable design method which is able to handle systems of this complexity. Careful splitting of the design procedure will be necessary since it cannot be performed in one single scan, i.e. sample time.

2.3.1 Optimization Criterion

The proposed PID design algorithm minimizes the Integrated Absolute Error, IAE, for a step load disturbance, see equation (2.12).

$$IAE = \int_0^\infty |e(t)|dt = \int_0^\infty |r(t) - y(t)|dt = \int_0^\infty |y(t)|dt \tag{2.12}$$

where $e(t)$ is the control error, $r(t)$ is the reference signal, here set to zero, and $y(t)$ is the process value.

2.3.2 Robustness Condition

The optimization is performed under a given robustness condition.

There are two separate robustness factors, one for the sensitivity function, M_s, and one for the complementary sensitivity function, M_t. These limit the maximum gain for each function and appear as circles in the Nyquist plot. As long as the Nyquist curve stays outside these circles the desired robustness is guaranteed.

Choosing $M = M_s = M_t$ these circles can be replaced by a single circle which encircles them both, the M-circle, see figure 2.6.

$$M_{center} = -\frac{2M^2 - 2M + 1}{2M(M-1)} \tag{2.13}$$

$$M_{radius} = \frac{2M - 1}{2M(M-1)} \tag{2.14}$$

M is always greater than 1 and the closer to 1 it is, the more robust the system

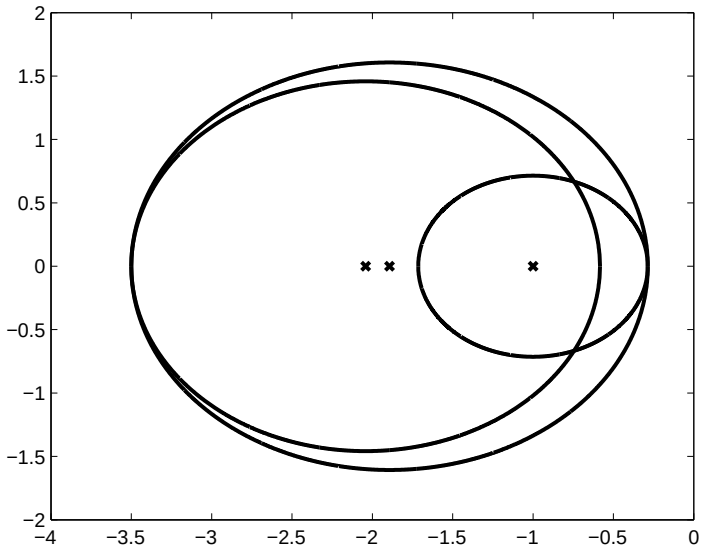

Figure 2.6: *The M_s-circle to the right and M_t-circle to the left are both encircled by the M-circle. Here $M = M_s = M_t = 1.4$.*

is. It can be shown that with $M \to 1$, the M-circle covers the entire left half plane.

Example 2.3.1. *Consider $M \to \infty$. This means that there is no upper limit on neither the sensitivity function nor the complementary sensitivity function. Then $M_{center} \to -1$ and $M_{radius} \to 0$, which means that the Nyquist curve may come arbitrarily close to the critical point -1, i.e. there is no robustness at all.*

2.3.3 PID Design

The PID controller can be described by equation (2.15).

$$C = K(1 + \frac{1}{T_i s} + \frac{T_d s}{\frac{T_d}{N} s + 1}) \tag{2.15}$$

The PID design is done by evaluating the performance of many controllers. This is done systematically by gridding the controller parameters T_i and T_d over a large region. Gridding over a more narrow area can optionally be done any number of times around the best found combination to find better values.

The controller gain, K, for each combination of T_i and T_d is determined by the Nyquist curve. K is chosen to be the largest gain which satisfies the robustness

condition and does not make the system unstable. To determine K with an algorithm, a maximum and a minimum gain have to be chosen, i.e. K_{max} and K_{min}.

One way to find K is to start by applying the maximum controller gain, K_{max} and then decrease it until the system is stable and satisfies the robustness condition. If there is no such controller with a gain larger than K_{min} the current combination of T_i and T_d is discarded.

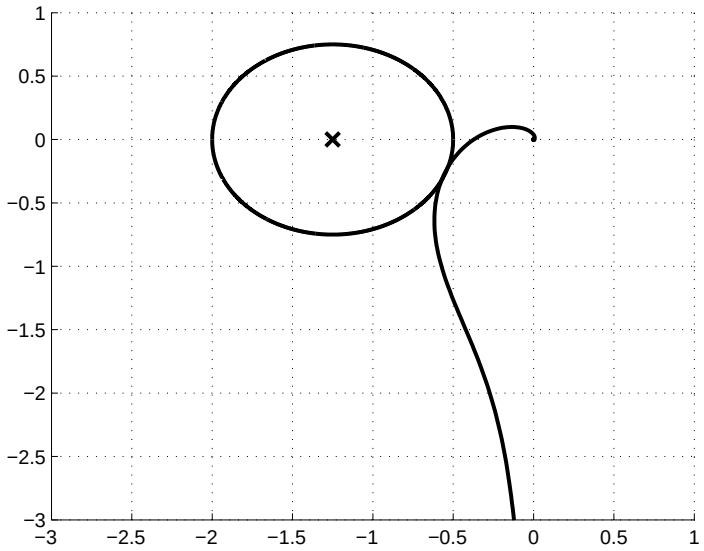

Figure 2.7: *Nyquist curve of stable system that satisfies the robustness condition.*

To minimize the number of gains that have to be tested one can take big steps, halving the gain every time. When a stable controller which satisfied the robustness condition is found, K is increased slowly until the M-circle is approached or the system becomes unstable. This is the algorithm used in [1].

A more elegant solution is to do this by means of binary search, halving the interval every time until the width of the remaining interval is sufficiently small, see appendix B.1. This reduces the complexity of the algorithm and makes it easier to split up over several scan since it does not have to remember if it is currently decreasing or increasing the gain. Also the K found by this algorithm is typically slightly larger which means that it is closer to the robustness limit, thus giving better performance.

Another great advantage is that the number of gains that have to be tested is greatly reduced, decreasing the execution time. Also to get a K closer to the M-circle is not computationally expensive compared to the algorithm used in [1], where a smaller increasing step is required, greatly increasing the number of gains to test. A comparison between the methods are illustrated in figure 2.8.

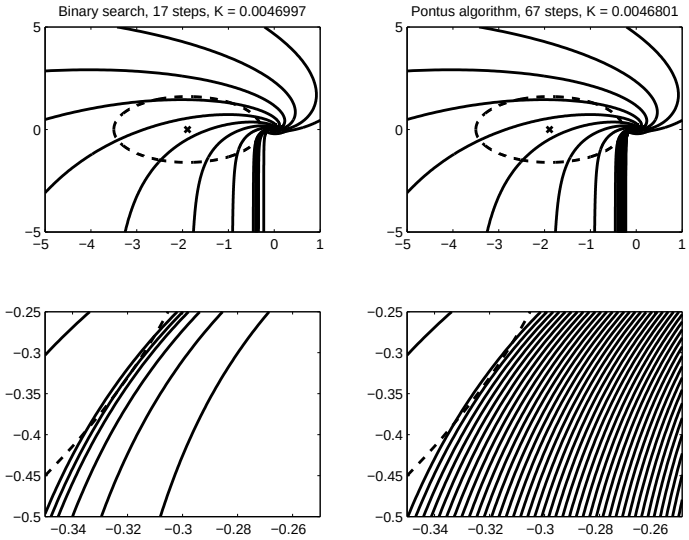

Figure 2.8: *Nyquist curve for tested gains with Binary search (left figures) and the algorithm used in [1] (right figures). The upper figures are overviews and the lower figures are zoomed in at the critical point. The number of steps is decreased with Binary search and also the obtained K is larger which means that the curve is closer to the M-circle and that the controller has better performance.*

Since the Nyquist curve is only evaluated at discrete frequencies it is important to select a proper frequency range and number of points to use. The current implementation evaluates 500 logarithmically spaced points in the interval $[0.001, 100]$, which might be a little too sparse but saves a lot of time. In [1] the same interval is used but the Nyquist curve is evaluated in 2000 points.

For large values of K this is more important since then the Nyquist plot is amplified much. Then one might miss that the robustness condition is not satisfied or even that the system in unstable, which in the worst case overflows the IAE variable. With $K_{max} = 100$ and 500 points there were a few cases where simulations were made on unstable systems. This might also be connected to the discretization, also mentioned in section 2.3.4. The cause in the specific cases has not been analyzed.

It is important to choose K_{max} and K_{min} properly since they will affect the final performance of the PIDs. IAE is normally decreased with higher allowed gain, K_{max}, but also makes the control signal more noisy. IAE surfaces for simulations on the same system with different K_{max} are seen in figure 2.9.

$$P = \begin{pmatrix} \frac{1}{10s+1} & \frac{0.5}{20s+1} \\ \frac{0.5}{20s+1} & \frac{1}{10s+1} \end{pmatrix}, D = \begin{pmatrix} \frac{1}{10s+1} & -\frac{0.5}{20s+1} \\ -\frac{0.5}{20s+1} & \frac{1}{10s+1} \end{pmatrix} \tag{2.16}$$

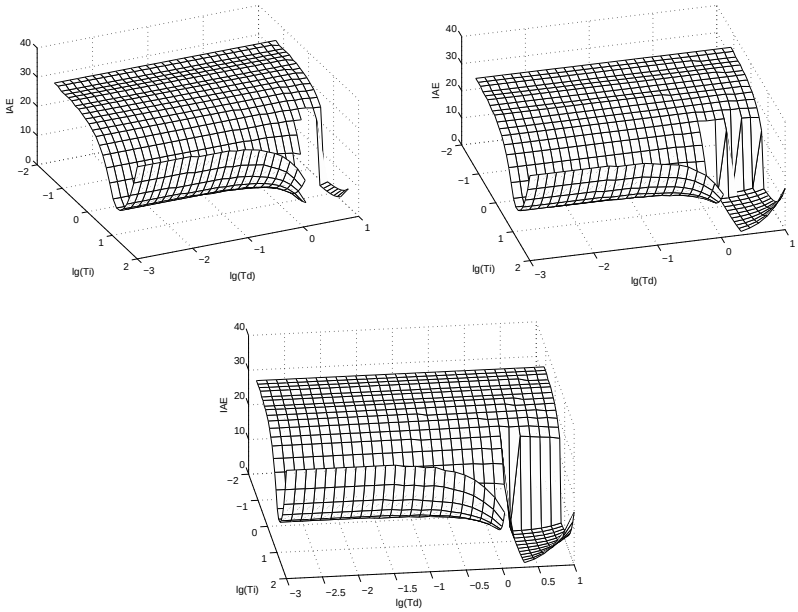

Figure 2.9: *Simulated IAE surface for the system in equation (2.16) but with different K_{max}, in the upper left, upper right and the lower plot K_{max} is 10, 100 and 1000 respectively. The higher gain allowed, the more of the surface is allowed and the smaller is the minimum IAE.*

2.3.4 Simulations

For every set of controller parameters, IAE simulations have to be done. A step load disturbance is applied at the time $t = 0$ and the simulation time is set to 10 times the system's time constant plus dead time.

Continuous IAE Simulations

Compared to running simulations in Control Builder, running them in MATLAB is rather easy. There are many powerful functions available and the array handling is very easy. Thus the simulation algorithm was first implemented in MATLAB using a continuous SimuLink model, see figure 2.10.

Discrete IAE Simulations

In Control Builder it is unrealistic to implement a variable time-step algorithm. Also, if the tuning algorithm uses the actual blocks that will be used when running the controller, a more accurate simulation will be obtained. However, only the controller and decouple filter are discrete and a completely discretized simulation would also mean discretization of the process.

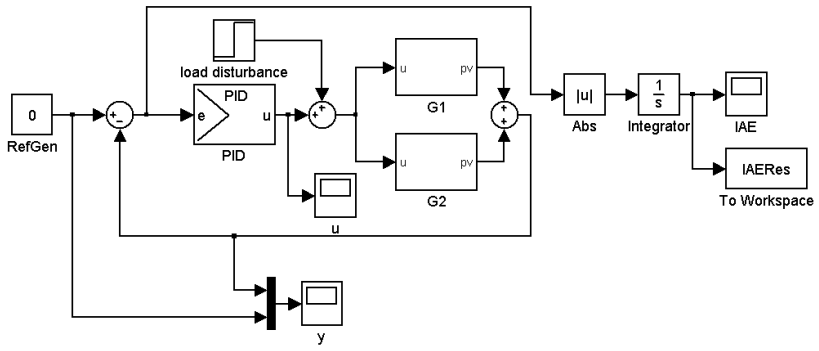

Figure 2.10: *SimuLink model for continuous IAE simulations run for one PID at a time. The blocks G_i are second order transfer functions with dead time, see equation (2.5).*

The ultimate simulation would be a discretized controller using fixed-step, connected to a continuous process using a variable-step algorithm. However, to decrease execution time the completely discretized simulation using fixed-step is used. Thus the SimuLink model was discretized to use the actual Control Builder PID and filter algorithms, see figure 2.11.

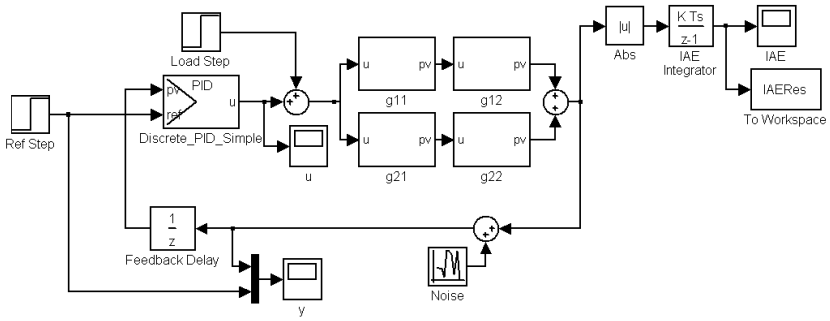

Figure 2.11: *SimuLink model for discrete IAE simulations run for one PID at a time. The blocks G_ij are first order transfer functions with dead time, see equation(2.5).*

A few of the simulated IAE controllers were unstable even though they satisfy the robustness condition. Possible cause were mentioned in section 2.3.3.

Choosing the Simulation Sample Time

A given candidate to use in the IAE simulations is the actual sample time. However, it is common that operators do not choose the sample time to fit the bandwidth of the system so the actual sample time might be a lot shorter

than appropriate, which means that more simulation steps are required. For excessively oversampled systems this will lead to an unreasonable amount of steps. Thus it is appropriate to set an upper limit for the number of simulation steps. If the upper limit is exceeded a more suitable sample time will be used in the simulations.

Controller Performance

IAE for a load disturbance is a commonly used optimization criterion in the process industry where load disturbances are common. For a coupled TITO system this is even more important since the two control signals themselves work as load disturbances on each other. However, only minimizing the IAE for a load disturbance may result in a controller which has other bad properties, such as noisy control signal or bad setpoint step response, see figure 2.12.

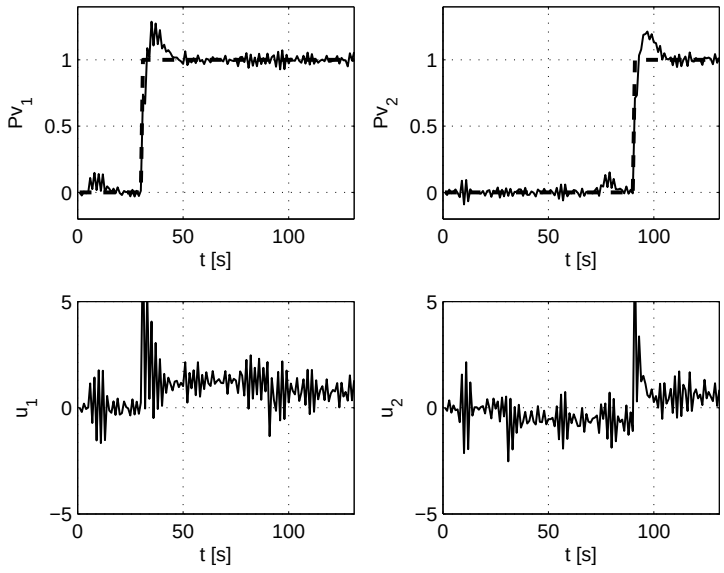

Figure 2.12: *A controller is obtained using the IAE minimization algorithm with $M = 1.4$. For Pv1 a load disturbance is applied at $t = 10$ and a setpoint step is made at $t = 30$. For Pv2 the times are $t = 60$ and $t = 90$. The load disturbance responses are really good and the step response are acceptable but the control signals are quite noisy.*

There are various ways to handle poor setpoint step responses. One is offered by the Control Module PidCC so exchanging the currently used Control Module PidSimpleCC for PidCC will solve that.

To avoid noisy control signals one could let it pass through a suitable low pass filter but this would introduce dynamics which may ruin the decoupling. Another way would be to set the limits on K_{max} and $T_{d_{max}}$ lower but that would

limit the performance for other controllers which could use higher values without making the control signal noisy, though this is easier and more preferable than adding a filter.

2.3.5 Code Splitting

The calculation of the controller gain, K, and the IAE simulations both take a long time to execute and thus they need to be split up over several scans and a maximum CPU utilization needs to be selected. There is no permissible way to measure the execution time with enough accuracy inside an algorithm in Control Builder. The only way to measure the execution time at all is not cascadeable and is used to measure the execution time for entire Control Modules or systems. However it may be used during development but then the measurement will be hardware specific. Since there are no other options, a variable describing the Controller's speed will be used with the measured hardware specific value as default value.

2.3.6 A Special Case

A special case where the simulation time can be reduced is when the order of the process and the decouple filter is the same and the same amount of dead time is decreased in both columns, i.e. $\min(L_{21}, L_{22}) = \min(L_{11}, L_{12})$ in equation (2.20).

Then the complete transfer functions for both loops will always be the same, resulting in the same controller parameters. Thus simulations only have to be run for one controller, halving the required simulation time.

$$P = \begin{pmatrix} p_{11}e^{-L_{11}s} & p_{12}e^{-L_{12}s} \\ p_{21}e^{-L_{21}s} & p_{22}e^{-L_{22}s} \end{pmatrix} \tag{2.17}$$

$$\text{adj}(P) = \begin{pmatrix} p_{22}e^{-L_{22}s} & -p_{12}e^{-L_{12}s} \\ -p_{21}e^{-L_{21}s} & p_{11}e^{-L_{11}s} \end{pmatrix} \tag{2.18}$$

$$D = \begin{pmatrix} p_{22}e^{(\min(L_{21},L_{22})-L_{22})s} & -p_{12}e^{(\min(L_{11},L_{12})-L_{12})s} \\ -p_{21}e^{(\min(L_{21},L_{22})-L_{21})s} & p_{11}e^{(\min(L_{11},L_{12})-L_{11})s} \end{pmatrix} \tag{2.19}$$

$$PD = \begin{pmatrix} p_{11}p_{22}e^{(\min(L_{21},L_{22})-L_{11}-L_{22})s} & 0 \\ 0 & p_{11}p_{22}e^{(\min(L_{11},L_{12})-L_{11}-L_{22})s} \end{pmatrix} \tag{2.20}$$

2.4 The Range Problem

In a SISO system the range problem is solved by normalizing the span of the input range with the span of the output range. Then calculations in the controller are done independent on the ranges. For TITO control with a decouple filter there is a new pair of ranges added, the PID output ranges, which have to be carefully chosen, see figure 2.13.

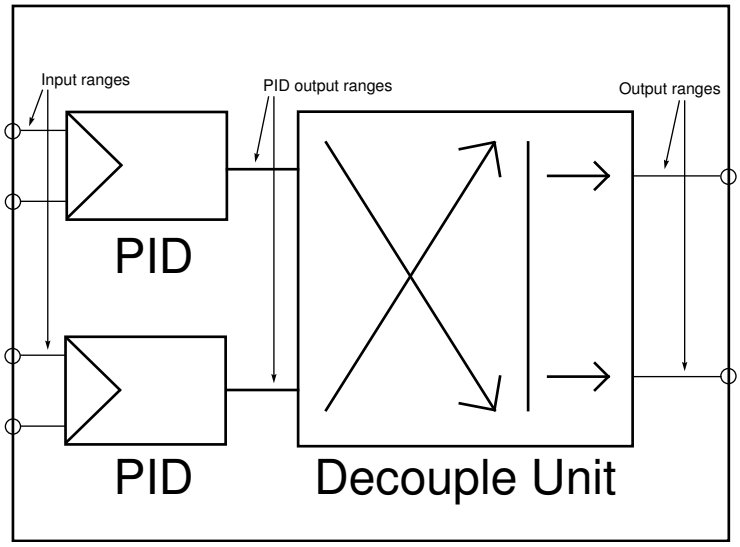

Figure 2.13: *Implementation model with ranges*

Let us first consider only the decouple filter with given input- and output ranges. The input ranges are weighted with the decouple filter's static gains which gives output ranges depending on input ranges and the gains. A naive approach would be to simply take all four combinations of minimum and maximum input values and calculate all possible output ranges using the gains and then normalize the extremes to fit the actual output ranges. This is equivalent to scaling the static gains in the decouple filter's rows and this ruins the decoupling, see example 2.4.1.

The only case when normalization is OK is when both ranges are amplified with the same factor, but then the ranges generally do not fit. An attempt to transform the decouple matrix to make the ranges fit without ruining the decoupling is presented in section 2.4.1. The developed method proves to only work in specific cases so another approach is developed, see section 2.4.2. Here the decouple filter's input ranges are dynamically calculated from the decouple filter's static gains and the output ranges.

Example 2.4.1. *Consider the process*

$$P = \begin{pmatrix} 3 & 1 \\ -1 & 1 \end{pmatrix}$$

Using the inverse of P as decouple filter we have

$$D = P^{-1} = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

The process is decoupled since $PD = I$.

If both the inranges and the outranges are $0 - 100$, the calculated output ranges using the decouple filter's gains are

Out1: 0 - 100
Out2: -25 - 25

To fit these to the actual output ranges, a gain of 2 has to be applied to Out2. A gain on the output is equivalent to applying the same gain to the second row in the decouple matrix and results in

$$PD = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} 3 & 1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{7}{4} & \frac{5}{4} \\ -\frac{5}{4} & \frac{1}{4} \end{pmatrix}$$

The process is no longer decoupled.

2.4.1 Fixed PID Output Range

Applying a gain on a row in the decouple filter is allowed if and only if the same gain is applied to the other row, i.e. multiplying all decouple matrix elements with the same constant. But this operation does not affect the relative range, just the ranges' sizes. A more useful operation which is allowed is to apply a gain to either input instead. This is equivalent to right hand multiplication with a diagonal matrix, which does not affect the decouple filter's performance. This is also equivalent to multiplying either column by a constant.

We now begin developing a transformation for an arbitrary decouple filter. For simplicity the input values are normalized to the range $0 - 100$ before any calculations. The reachable values after the decouple filter depend on the gains of the decouple matrix's elements. To visualize these values we start by representing the possible normalized input values as a square. Transforming this square with the decouple filter gains matrix we get a new quadrilateral which contains all possible values after the decouple filter, see figure 2.14. The maximum and minimum values for each output can be set to the smallest rectangle to envelope the transformed quadrilateral.

Here we have to be careful since different lengths of the rectangle's sides will cause the outputs to be normalized by different values, just like in example 2.4.1. It is preferable to find input gains which make the range spans equal, see theorem 2.4.1. For most real systems it is possible to find such a transformation

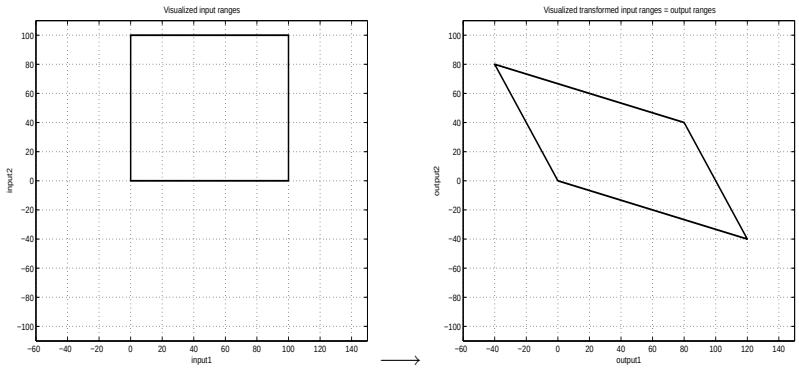

Figure 2.14: *Transformed input ranges*

for the decouple filter, but for the cases where it is not possible there are two options. Either we can pretend that the big range is as big as the small range, forcing a max/min reached state when it tries to go beyond that, or we could expand the small range with unreachable values. These operations are not unequivocal, in the first case one has to decide which parts to forbid and in the second case we need to choose what offset to add on to the original small range.

Theorem 2.4.1. *We want to find the input gains, x_1 and x_2, which make the output ranges' spans match. There are a few cases depending on if some elements in the decouple filter are zero.*

For a diagonal matrix it is always possible and the input gains are

$$\begin{cases} |x_1| &= \frac{1}{|d_{11}|} \\ |x_2| &= \frac{1}{|d_{22}|} \end{cases}$$

For an upper triangular matrix it is possible if $|d_{12}| < |d_{22}|$, then the input gains are

$$|x_1| = 1 - \left| \frac{d_{12}}{d_{11}d_{22}} \right|$$
$$|x_2| = \frac{1}{|d_{22}|}$$

For a lower triangular matrix it is possible if $|d_{21}| < |d_{11}|$, then the input gains are

$$|x_1| = \frac{1}{|d_{11}|}$$
$$|x_2| = 1 - \left| \frac{d_{21}}{d_{11}d_{22}} \right|$$

For a full matrix it is possible if exactly one of the inequalities is valid in both $|d_{11}| > |d_{21}| \bigvee |d_{12}| > |d_{22}|$ *and* $|d_{11}| < |d_{21}| \bigvee |d_{12}| < |d_{22}|$. *Then the relation between the input gains is*

$$|x_1| = \frac{\beta |x_2|}{\alpha}$$

where $\alpha = |d_{11} - d_{21}|, \beta = |d_{22} - d_{12}|$.

Proof. Start by applying input gains to the original decouple filter

$$D\Lambda = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} \begin{pmatrix} x_1 & 0 \\ 0 & x_2 \end{pmatrix} = \begin{pmatrix} d_{11}x_1 & d_{12}x_2 \\ d_{21}x_1 & d_{22}x_2 \end{pmatrix}$$

Without loss of generality we say that we want both ranges to have span 1

$$\begin{cases} |x_1 d_{11}| + |x_2 d_{12}| &= 1 \\ |x_1 d_{12}| + |x_2 d_{22}| &= 1 \end{cases}$$

We now have 3 cases

1. Diagonal matrix
2. Triangular matrix
3. Full matrix

For a diagonal matrix one element on each row is zero, we have

$$\begin{cases} |x_1 d_{11}| &= 1 \\ |x_2 d_{22}| &= 1 \end{cases}$$

$$\begin{cases} |x_1| &= \frac{1}{|d_{11}|} \\ |x_2| &= \frac{1}{|d_{22}|} \end{cases}$$

For a triangular matrix one element is zero, we have

$$\begin{cases} |x_1 d_{11}| + |x_2 d_{12}| &= 1 \\ |x_2 d_{22}| &= 1 \end{cases}$$

$$|x_2| = \frac{1}{|d_{22}|}$$
$$|x_1 d_{11}| < 1$$
$$|x_2 d_{12}| = \left|\frac{d_{12}}{d_{22}}\right| < 1 \Rightarrow |d_{12}| < |d_{22}|$$

If the condition is satisfied then

$$|x_1| = 1 - \left|\frac{d_{12}}{d_{11} d_{22}}\right|$$

If d_{12} would have been the element which is zero instead of d_{21} then symmetry gives

$$|x_1| = \frac{1}{|d_{11}|}$$
$$|x_2| = 1 - \left|\frac{d_{21}}{d_{11} d_{22}}\right|$$

Finally we have the case of a full matrix where none of the elements are zero.

$$\begin{cases} |x_1 d_{11}| + |x_2 d_{12}| &= 1 \\ |x_1 d_{12}| + |x_2 d_{22}| &= 1 \end{cases}$$

$$|x_1 d_{11}| < 1$$
$$|x_2 d_{12}| < 1$$
$$|x_1 d_{21}| < 1$$
$$|x_2 d_{22}| < 1$$

$$|x_1| < \frac{1}{\max(|d_{11}|, |d_{21}|)}$$
$$|x_2| < \frac{1}{\max(|d_{12}|, |d_{22}|)}$$

$$|x_1 d_{11}| + |x_2 d_{12}| = |x_1||d_{11}| + |x_2||d_{12}| < \frac{1}{\max(|d_{11}|,|d_{21}|)}|d_{11}| + \frac{1}{\max(|d_{12}|,|d_{22}|)}|d_{12}|$$

The last expression has to be at least one. This is guaranteed if $|d_{11}| > |d_{21}| \bigvee |d_{12}| > |d_{22}|$. In the same way we get $|d_{11}| < |d_{21}| \bigvee |d_{12}| < |d_{22}|$ for the second equation. To satisfy both conditions exactly one inequality is valid in each case and we can rewrite the original matrix as

$$\begin{pmatrix} (|d_1| \pm |\alpha|)|x_1| & |d_2||x_2| \\ |d_1||x_1| & (|d_2| \pm |\beta|)|x_2| \end{pmatrix}$$

where $\alpha > 0, \beta > 0$ and the $\pm$-signs are equal.

With equal ranges we get

$$\begin{aligned}
(|d_1| \pm \alpha)|x_1| + |d_2||x_2| &= |d_1||x_1| + (|d_2| \pm \beta)|x_2| \\
(|d_1| \pm \alpha)|x_1| + |d_2||x_2| - |d_1||x_1| - (|d_2| \pm \beta)|x_2| &= 0 \\
|d_1||x_1| \pm \alpha|x_1| + |d_2||x_2| - |d_1||x_1| - |d_2||x_2| \mp \beta|x_2| &= 0 \\
\pm\alpha|x_1| \mp \beta|x_2| &= 0 \\
\pm\alpha|x_1| &= \pm\beta|x_2| \\
\alpha|x_1| &= \beta|x_2| \\
|x_1| &= \tfrac{\beta|x_2|}{\alpha}
\end{aligned}$$

Thus it is always possible to find the values x_1 and x_2 when the condition is satisfied. If it is not, there are symmetric cases, consider $|d_{11}| \geq |d_{12}| \bigwedge |d_{21}| \geq |d_{22}|$. In the case of equalities the ranges are the same to start with so in that specific case it is still possible. In any other case there is no solution since the first row will always have a greater value in each column regardless of x_i. □

There also are a few disadvantages with input gains. The remaining coupling will get amplified with the same gain, see example 2.4.2. Also the inverse gain indirectly is applied to the controller, making it more noise sensitive. The example shows that even when it is possible to match the ranges' spans it is not always reasonable to do so, i.e. never use the transformation with big gains.

Example 2.4.2. *Start with*

$$P = \begin{pmatrix} 11 & 1 \\ 1 & 1 \end{pmatrix}$$

and suppose that after decouple filter design we get

$$D = \begin{pmatrix} 0.101 & -0.102 \\ -0.098 & 1.101 \end{pmatrix}$$

$$PD = \begin{pmatrix} 1.013 & -0.021 \\ 0.003 & 0.999 \end{pmatrix}$$

To match the ranges' spans the gain 333 is applied to column one in D

$$D = \begin{pmatrix} 33.633 & -0.102 \\ -32.634 & 1.101 \end{pmatrix}$$

$$PD = \begin{pmatrix} 337.329 & -0.021 \\ 0.999 & 0.999 \end{pmatrix}$$

Due to the range manipulation the second coupling is as big as it was without any decouple filter.

Conclusion

It has been shown that there is no good way to manipulate the decouple matrix statically to avoid the range problem. The only possible transformation does not work on all matrices and even when it works there are many cases where it is not reasonable to apply it. Some other way of handling the ranges is required. If one would decide to use this method anyway it should also be mentioned that the calculated decouple filter is only valid if the ranges remain unchanged. If the ranges were to change, the decouple filter would have to be altered to rematch the output ranges' spans. This is rather easy since a modification in an input range is compensated by the same change in the corresponding column, a modification in an output range by modifying the corresponding row.

2.4.2 Dynamic PID Output Range

The approach in section 2.4.1 is based on the assumption that the ranges before and after the decouple filter are fixed. That approach was discarded meaning that a way to calculate dynamic PID output ranges is needed, see figure 2.13. In this section a way to choose these ranges is derived to completely handle the range problem.

From linear algebra it is well known that a transformation, T, is bijective iff $\det(T) \neq 0$. This means that there exists an inverse transformation, T^{-1}. Here the transformation matrix is the decouple filter's static gain matrix, D, and we can represent the PID output ranges' extremes with four points which are unambiguously mapped to the output ranges' extremes. Taking the extreme values of the transformed points gives the output ranges.

We would like to do this the other way around, i.e. starting with the output ranges calculating the input ranges. This is not possible since both the points and their transformations are unknown, only the extreme values for each output range are known and this is not sufficient.

Hence the given output ranges' extremes are inverse transformed instead. Taking the minimum and maximum value in each direction gives the PID output ranges. This normally means including PID output combinations which are outside of the allowed output ranges, see figure 2.15.

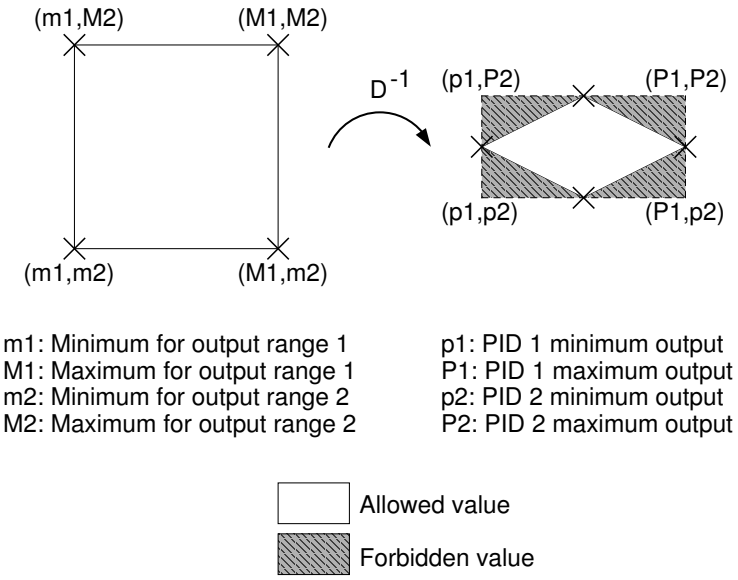

Figure 2.15: *Inverse transformed output ranges and valid PID outputs.*

Example 2.4.3. *Consider the decouple gain matrix*

$$D = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

$$D^{-1} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$$

The output ranges are -50 - 100 and 20 - 150. The four extreme points are gathered in a 2x4-matrix,

$$E = \begin{pmatrix} -50 & -50 & 100 & 100 \\ 20 & 150 & 20 & 150 \end{pmatrix}$$

Then E is inverse transformed

$$P = D^{-1}E = \begin{pmatrix} -70 & -200 & 80 & -50 \\ 90 & 350 & -60 & 200 \end{pmatrix}$$

Taking maximum and minimum values for each row gives the PID output ranges, $[p_1, P_1]$ *and* $[p_2, P_2]$

$$\begin{pmatrix} p_1 \\ P_1 \\ p_2 \\ P_2 \end{pmatrix} = \begin{pmatrix} -200 \\ 80 \\ -60 \\ 350 \end{pmatrix}$$

The only valid output value combinations are the ones inside the quadrilateral with vertices defined by the columns in P, see figure 2.16

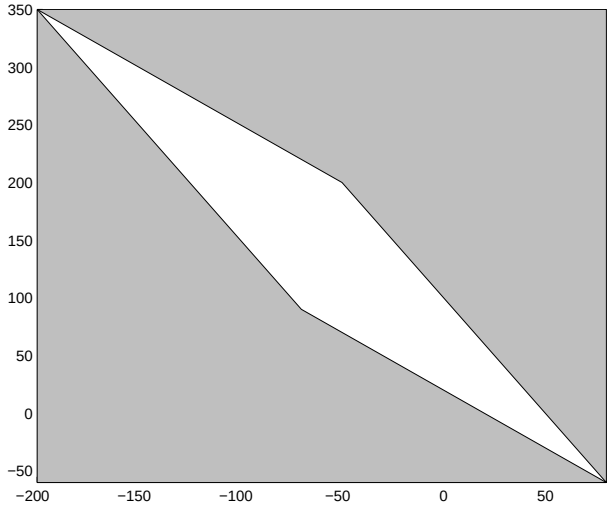

Figure 2.16: *Inverse transformed output ranges.*

One way to handle a forbidden PID output combination is to look at the output values. If any of them has reached its limit, one uses the signs in the decouple filter gain matrix to determine if this corresponds to a minimum or a maximum for each PID. Then one can send this information to the PIDs to inhibit them from increasing/decreasing further. If an output reached a maximum and is related to the PID through a positive gain then the PID has also reached a maximum. If the gain would have been negative the PID had reached a minimum and so on, see example 2.4.4. If the gain is zero, the PID state is unaffected.

Example 2.4.4. *Consider the decouple gain matrix,*

$$D = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$$

It has an inverse transformed range which can been seen in figure 2.17.

a) The out values have not reached any limits which corresponds to the PID outputs being inside the allowed region. They are both free to either increase or decrease their control signals.

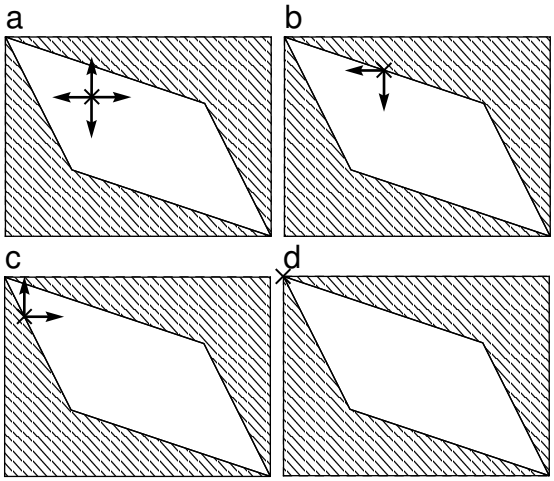

Figure 2.17: *The current value (cross) and the allowed directions for the PIDs (arrows). When the border is reached the value is inhibited to go outside of the allowed region. In (d) the control signals get stuck since there is no allowed direction for either PID.*

b) and c) One of the out values have reached a limit which corresponds to the PID outputs being on the border of the allowed region. The control signals are then inhibited to go out of the region.

d) Two output limits are reached at the same time which corresponds to the PID outputs being in one of the inverse transformed corners. In this case a corner which inhibits both increment and decrement of both control signals, the PIDs are stuck forever in this corner unless this is handled in some way.

In example 2.4.4 we see that limiting the PIDs in this way may lead to getting stuck in a corner. Simply by checking if both PIDs are inhibited to both increase and decrease tells if the signals are stuck. This however can only occur if two output limits are reached simultaneously, which means that the output signal cannot go any further out of the PID output region. The only way that the PIDs can move the output signals is then towards the allowed area so all inhibits may be removed. This has the great advantage that we do not have to check which corner has been transformed to which corner in order to keep track on how to inhibit the PIDs to go further out of the region.

Example 2.4.5. *The current value is located in the point D, see figure 2.18, which corresponds to the dangerous corner D'.*

Here a fundamental property of linear transformations is used, i.e. straight lines are transformed at straight lines, intersections are transformed at intersections and inner points are transformed at inner points.

By removing all inhibits it is possible for the PIDs to move into either of the regions denoted a', b' or c'. In the right figure it is seen that these regions are

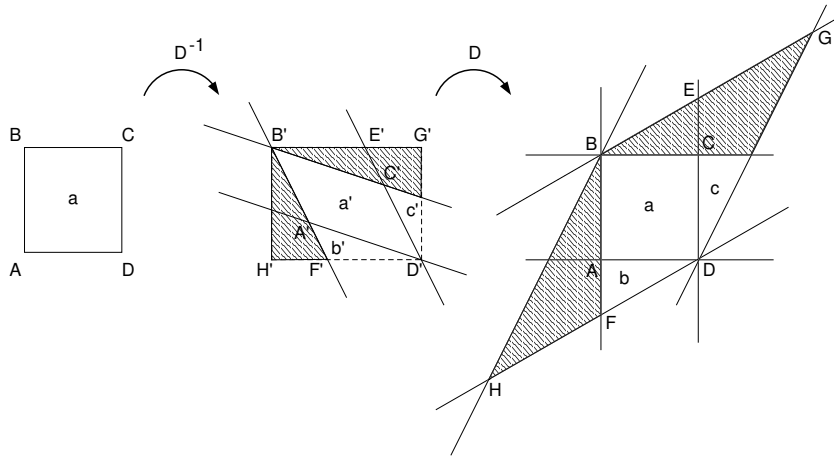

Figure 2.18: *The current value has ended up in D, which corresponds to a dangerous corner. The given output ranges (left) are inverse transformed (middle) and then transformed back (right) to show which areas are possible to go to if all inhibits are removed.*

transformed at regions along the sides of the original range which always is the case for regions with dangerous corners. Going into either of these regions would mean leaving the dangerous point. What happens the next sample depends on which region the control signals move into. If they move into region a, there are no inhibits on the PIDs. If they move to either region b or c the PIDs will be inhibited as usual by that single border.

Tests have shown that the dangerous corners not only exists theoretically but are possible to reach for real systems. Mostly you end up in a dangerous corner when setting unreasonable combinations of setpoints. With the addition to release the inhibits it is always possible to leave a dangerous corner.

Conclusions

There exists an easy way to determine the dynamic output ranges for the PIDs by calculating the inverse transformation of the decouple filter's output ranges' extremes. Also inhibiting the PIDs' output values from going outside the transformed region is solved by simply looking at the decouple filter's gain matrix with an additional fix for dangerous corners.

This method is not restricted to the TITO case and is applicable for any number of inputs and outputs as long as the determinant of the decouple filter's gain matrix is not equal to 0.

Note that simply releasing all the inhibits may cause wind-up problems in the dangerous corners. This can only happen if the backward range sent by the decouple filter is not accepted by the previous Control Modules, i.e. the PIDs.

Then the assumption that the previous Control Module cannot go any further out of the allowed region might not be valid. In this implementation the PIDs are internally set to always accept the backward range automatically so for this implementation it is a simple yet good fix.

2.5 Noise Estimate

A good initial estimate of the noise is important since a poor estimate may stall the identification. Additive white noise is assumed. A real-time algorithm is preferable to avoid peak utilization. The algorithm must be reliable and robust so that it works in any possible case.

In the available relay identification algorithm a noise estimate is done to be able to decide the hysteresis for the relay identification. The implementation takes the minimum and maximum process value during some scan and uses the span as noise estimate, see appendix B.2.

The method has been analyzed with the conclusion that the confidence interval for the estimate increases with longer measurement, see figure 2.19. Also, the estimate varies with the number of scan used to measure it, which is unsuitable.

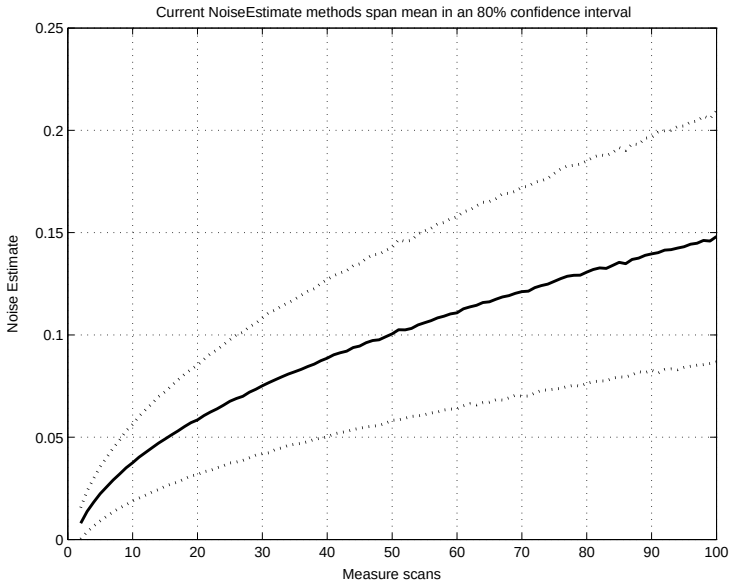

Figure 2.19: *Noise estimate method in the relay identification algorithm as a function of measurement scans with an 80% confidence interval.*

A method with overall smaller confidence intervals and specifically smaller confidence interval with longer measurements is required. An ordinary variance estimate, see equation (2.21), has all these properties, see figure 2.20. However it cannot be implemented as a real-time algorithm since the mean value is

unknown during the measurement.

$$\sigma_{est}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{2.21}$$

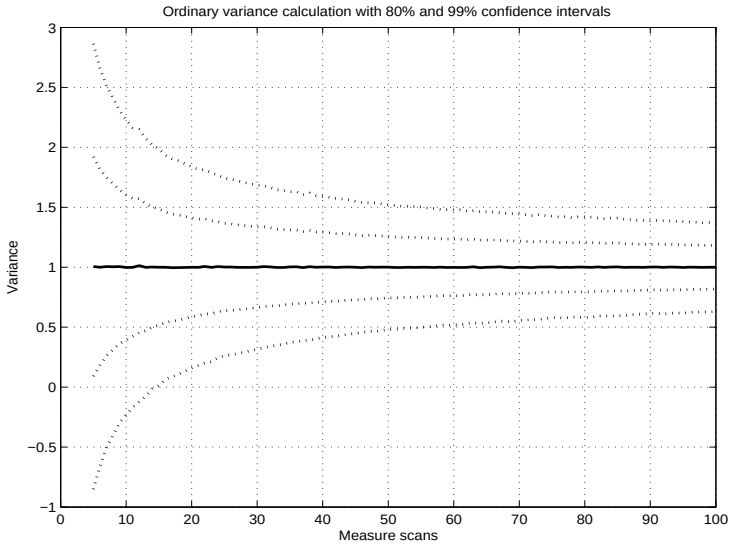

Figure 2.20: *Common estimation of standard deviation as a function of measurement scans with an 80% and an 99% confidence interval.*

Since the noise estimation is the first phase of the entire identification the decouple filter has just been set to through mode. Without a proper way to make sure that the process values do not vary transiently because of this, or since the operator may start the identification when the process values are changing, the process values cannot be assumed to be steady.

A typical measurement with a process value movement dominating the noise can be seen in figure 2.21. To obtain good noise estimates a moving process value must not affect the estimate. To completely distinguish the process movement from the noise is impossible but some algorithms do it better than others.

One way to limit the influence is to check the difference between the initial and the end value, or mean values for these. If it is large compared to the estimated noise level, the estimate is probably poor. With a real-time method this can be tested at each scan, which means that the algorithm can be restarted as soon as the condition fails, decreasing unnecessary measurement time.

For non real-time approaches all measurement points are saved and the variance calculation is done when the measurement is completed. Since more information is available this can potentially give better estimates. A disadvantage is that the algorithm will have a peak utilization when the measurement completes.

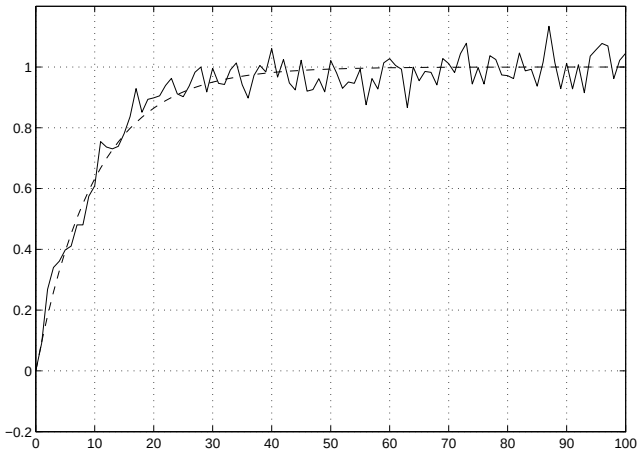

Figure 2.21: *Typical process value variations.*

For systems with very small noise levels, or during simulation without any noise, the process values will have to be practically absolutely steady. For small noise levels this takes a very long time so there should be a specified minimum noise level.

2.5.1 Real-time

For all real-time algorithms the estimation is restarted as long as at least one process value is not steady enough. Equation (2.22) describes the steady condition. The closer to equality, the worse the estimate is.

$$|\bar{x}_{final} - \bar{x}_0| \leq k\sigma_{est} \tag{2.22}$$

Simulations have been done in MATLAB to determine how much the estimate is affected because of moving process values for each algorithm. A mean movement will increase the estimate which in turn will allow larger mean movements and so on. Thus one has to be careful when deciding the constant k not to get an algorithm which accepts just any estimate.

2.5.2 Real-time - Static Mean

The static mean algorithm uses the initial value as the mean value during the entire estimate, in figure 2.21 that would be 0.

Suppose that the noise can be neglected due to a large process value movement. Then σ_{est} will be about the same size as the mean movement. Choosing k larger than 1 will mean that this estimate will be accepted even if it has nothing to

do with the noise level. With k less than 1 however, a moving mean alone will not be enough to accept the estimate.

Now suppose that the process value is completely steady and that only pure noise is measured. Then choosing k too small will cause problems since a valid estimate may be rejected. For example, having a good estimate of the first point, only using the last point as end mean and having $k = 0.5$ will only accept 38% of the estimates. With $k = 1$ only 68% of the valid measurements will be accepted. How small k may be depends on how many points are used to estimate the means but if the end mean is calculated during the end of the measurement, many points should not be used since that will give an old mean which can also cause problems.

Simulations show that $k = 0.5$, using the 4 last points of the measurement is an appropriate combination. With these parameters only 16% of the valid estimates are rejected and 4% of the estimates are accepted when the mean is moving twice the noise level, the worst accepted σ_{est} was 2.5σ.

2.5.3 Real-time - Variable Mean

The real-time method in section 2.5.2 uses a constant mean during the estimation. Another approach is to use a variable mean. Two such methods have been analyzed, namely moving average and forgetting factor.

2.5.4 Real-time - Forgetting Factor

The forgetting factor method weights the measured values so that older values affect the current mean less than newer values, see equation (2.23).

$$\bar{x}_i = \lambda \bar{x}_{i-1} + (1 - \lambda) x_i \tag{2.23}$$

How good the estimate is with this method depends heavily on the forgetting factor, the larger the less noisy $\bar{x}_i$ is but the longer it takes for new values to dominate over old ones, see figure 2.22. This is similar to the moving average method when having n big, see section 2.5.5. The estimate error for a moving mean depends on the forgetting factor as seen in figure 2.23.

When calculating the variance it is important to use $(x_i - \bar{x}_{i-1})$ and not $(x_i - \bar{x}_i)$ since x_i and $\bar{x}_i$ are correlated. This method was not analyzed further since the moving average method was found to work very well.

2.5.5 Real-time - Moving Average

The moving average method uses the arithmetic mean of a fixed number, n, of the last measured values as the current mean. The variance estimate is then

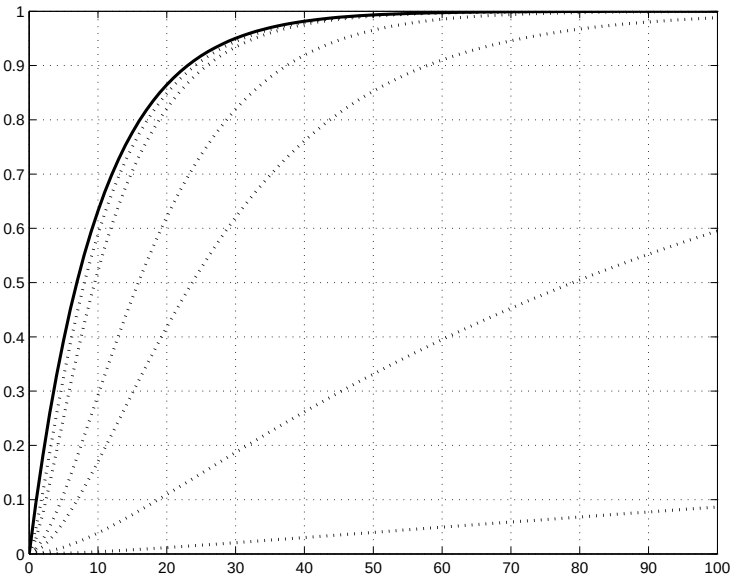

Figure 2.22: *Actual mean (solid line) and various moving means using forgetting factor (dashed lines). Starting closest to the solid line the forgetting factors are 0.5, 0.7, 0.9 0.95 0.99 and 0.999.*

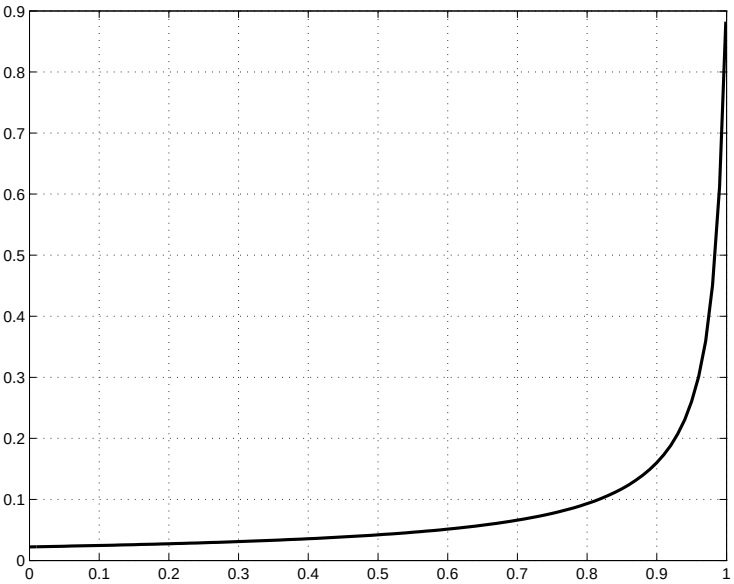

Figure 2.23: *Erroneously added variance as a function of the forgetting factor, with a process value moving as in figure 2.22.*

$$\sigma_{est}^2 = \frac{1}{n-1}\sum_{k=1}^{n}(x(k) - \bar{x}(k))^2 \tag{2.24}$$

where

$$\bar{x}(k) = \begin{cases} \frac{x(k-1)+x(k-2)+\cdots+x(k-i)}{i}, & k = i < n \\ \frac{x(k-1)+x(k-2)+\cdots+x(k-n)}{n}, & k \geq n \end{cases}$$

The moving average method also requires a normalization since many measurement points are added, with that their variance. Due to this the estimate will also have to be normalized, see equation (2.25).

$$\sigma_{est} = E((e_i - \frac{1}{n}\sum_{k=1}^{n} e_{i-k})^2) =$$

$$E(e_i^2 - \frac{2}{n}\sum_{k=1}^{n}(e_{i-k})e_i + \frac{1}{n^2}\sum_{k=1}^{n}(e_{i-k})^2) =$$

$$E(e_i^2) - \frac{2}{n}E(\sum_{k=1}^{n} e_{i-k}e_i) + \frac{1}{n^2}E(\sum_{k=1}^{n}(e_{i-k})^2) =$$

$$\sigma^2 + \frac{\sigma^2}{n} =$$

$$\frac{n+1}{n}\sigma^2$$

$$\sigma_{norm} = \sigma_{est}\sqrt{\frac{n}{n+1}} \tag{2.25}$$

Having n small will make the moving average noisy, which is not necessarily bad, while having n larger means that the actual mean is followed slower, decreasing the quality of the estimate, see figure 2.24

A disadvantage with this method is that it requires buffering of the measurement points and an average calculation for each scan. The larger n is, the more utilization is required and the worse the estimate is since the mean is followed slowly. From this we draw the conclusion that there are actually no advantages with having n larger, so let us choose it as small as possible, i.e. $n = 1$. Then the normalization factor is $\frac{1}{\sqrt{2}}$ and the mean value used each scan is simply the last measured value. This is the method used in the actual implementation.

The specific case $n = 1$ has been analyzed further to see how much the estimate is affected by moving process values. Comparing this to the static mean in section 2.5.2, it is the previous value which is used as current mean instead of the beginning value. It seems reasonable that the process value would be

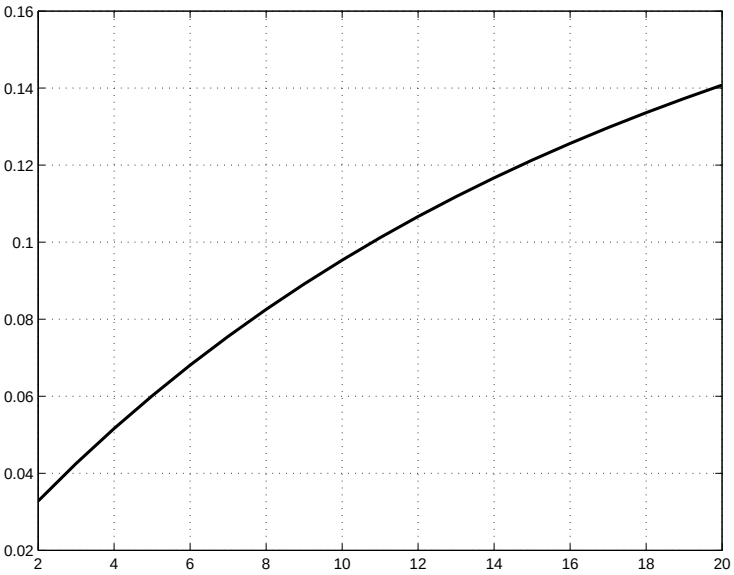

Figure 2.24: *Erroneously added variance as a function of n.*

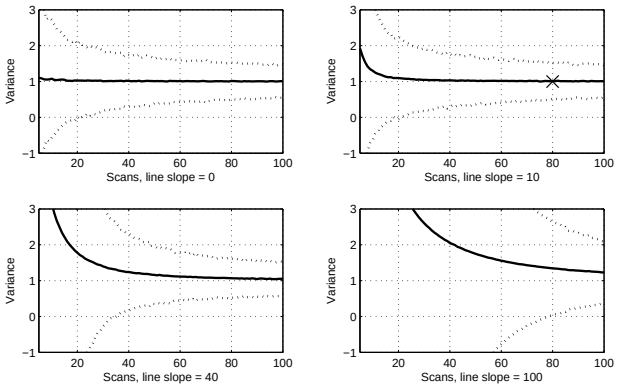

Figure 2.25: *Noise estimates with 99% confidence intervals for various measurement scans and line slopes. X marks the worst case for the current implementation.*

allowed to move up to the maximum for the static mean method, but once each measurement scan instead, i.e. m times $k = 1 \Rightarrow k = m$.

With $k < m$ a moving mean alone will not be accepted. How much error there is in the estimate for various slopes has been simulated, see figure 2.25. A decent estimate is obtained even when having m close to k, but keeping it smaller will give better estimates. I have chosen to use 80 measurement scans and $k = 10$ in the implementation. This would correspond to $k = 0.125$ in the static mean case which means a risk of approximately 4% that a poor estimate is accepted in the case when the process value movement is of the same order as the noise level. Here the mean of the accepted estimates was $\bar{\sigma}_{est} = 1.35\sigma$ and the worst accepted in 100 000 simulations was $\sigma_{est} = 1.7\sigma$ and. For process value movements of more than twice the noise level the risk can be neglected, not a single estimate was accepted in 100 000 simulations. Also the acceptable span is increased 10 times which practically nullifies the risk of rejecting good estimates.

2.5.6 Non Real-Time

With a non real-time approach it is possible to fit curves to the measurement data using regression methods to get better estimates. In the typical case a process value moves toward a steady value exponentially, see figure 2.26.

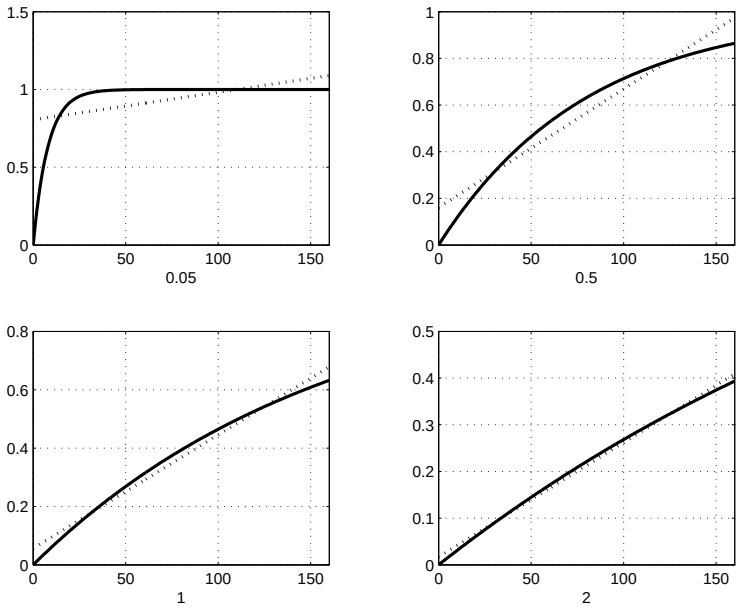

Figure 2.26: *Linear fits for typical measurements.*

How good the fits are depends on the quotient between the filter time and the sample time. It also depends on the number of measurement points, how much

a process value is moving and the order of the fitted polynomial. With linear regression, see figure 2.27, it is concluded that at least 40 measurement points should be used, preferably 80. Using more than 80 points will not improve the estimate considerably.

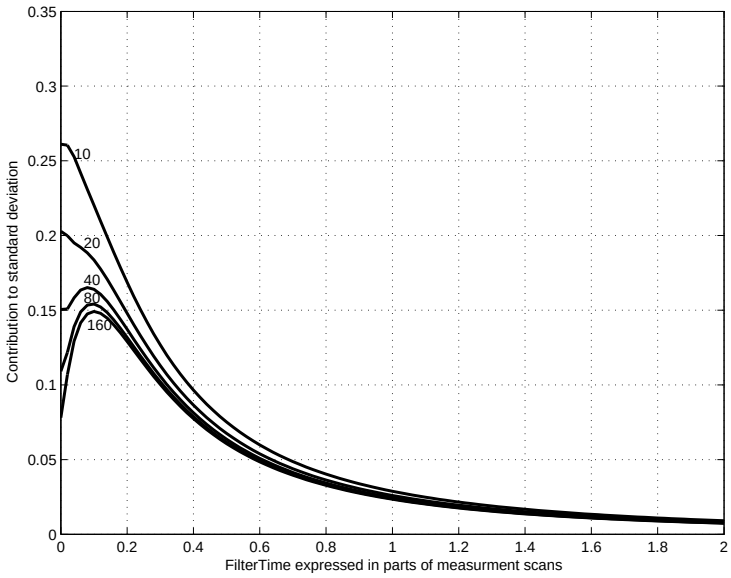

Figure 2.27: *Extra standard deviations added as a function of measurement scans and of the quotient between filter time and sample time.*

How much error is introduced due to moving process values increases linearly with how much a process value moves. Using 80 measurement points the worst case is about 15% of how much the process value moves. For 40 and 80 scan the peak is at 3.2 and 8 scan respectively. With a well chosen sample frequency, i.e. $2 - 20$ times the process's speed, both are probable. Better estimates can be obtained by limiting the process value movement, like for the real-time methods.

A way to further decrease the estimation error is to fit a polynomial of higher order using multiple linear regression. It would be better to fit an exponential curve, or even better an exponential curve with an initial dead time. Then all cases for first-order systems with dead times are handled well. Since all these cases would increase the computation time considerably, this kind of fitting has not been analyzed. See table 2.1 and figure 2.28 for polynomial fits and their respective errors added depending on the polynomial order.

Order	Error
0	0.1910
1	0.1377
2	0.0814
3	0.0403
4	0.0170
5	0.0063
6	0.0020
7	0.0006

Table 2.1: *Maximum estimation error per unit process value movement due to fit error for various polynomial orders. The time constant for the movement is 10 and the number of measurement scans = 100.*

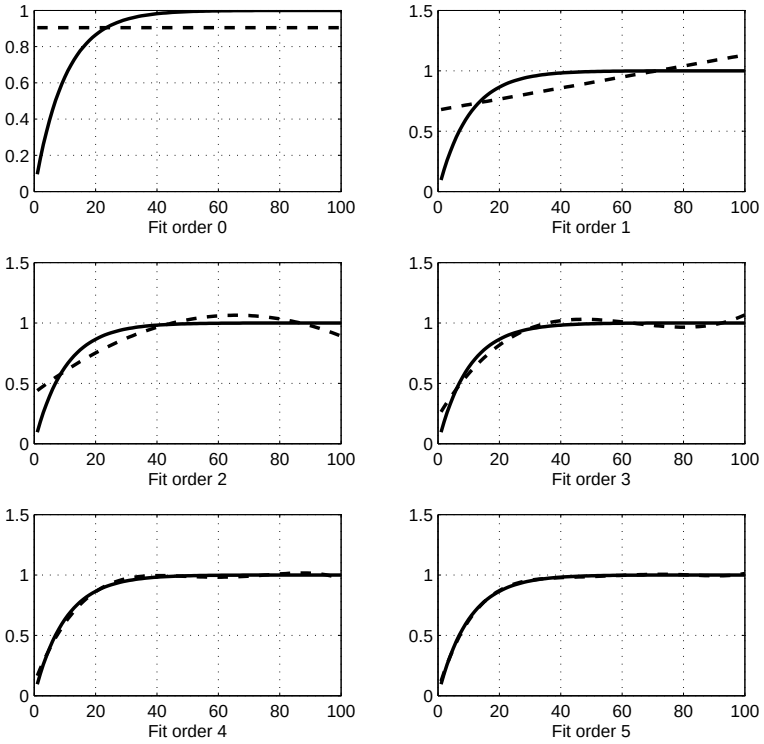

Figure 2.28: *Fit plots for various polynomial orders. Filter time = 10% of measurement scans = 100.*

Chapter 3

Implementation

3.1 General Strategy

In the process industry processes are often modelled as FOWDT transfer function. Since the process is not known a priori a suitable method of system identification has to be used. During the identification the decouple filter and the controllers must be passive, i.e. the decouple filter is the identity matrix and the controllers are tracking the identification outputs.

With a model of the system, a decouple filter can be designed. The final part is designing the controllers, which could prove difficult since the current tuning algorithms assume simple processes. The decoupled systems are usually very complex so the extensive search approach in [1] will be used.

1. System Identification
2. Design Decouple Filter
3. Controller Design

During development the aim will be to reuse as much of the existing Control Modules and Function Blocks as possible. Reusing existing code has many advantages. Less time will be required during development, the code will be easier to maintain and also the reused parts are more well tested than new ones. However, one has to be careful only to reuse code which actually does what you want it to, and nothing else.

A bottom-up approach for software development will be used, designing smaller parts, putting them together to build up more complex systems. This way the separate parts are easier to test and it is easy to change one part without affecting others.

Note that in this chapter input refers to the decouple filter's inputs, i.e. the process values, and output refers to the decouple filter's outputs, i.e. the control signals.

3.2 Control Builder

3.2.1 Introduction

The Decouple Filter will be developed in the engineering tool called Control Builder, or CB for short. It is developed by ABB and follows the IEC 61131-3 standard. There are five programming languages defined by 61131-3 but only Structured Text will be used since it is the most powerful of the languages, and also the only one to resemble ordinary programming languages.

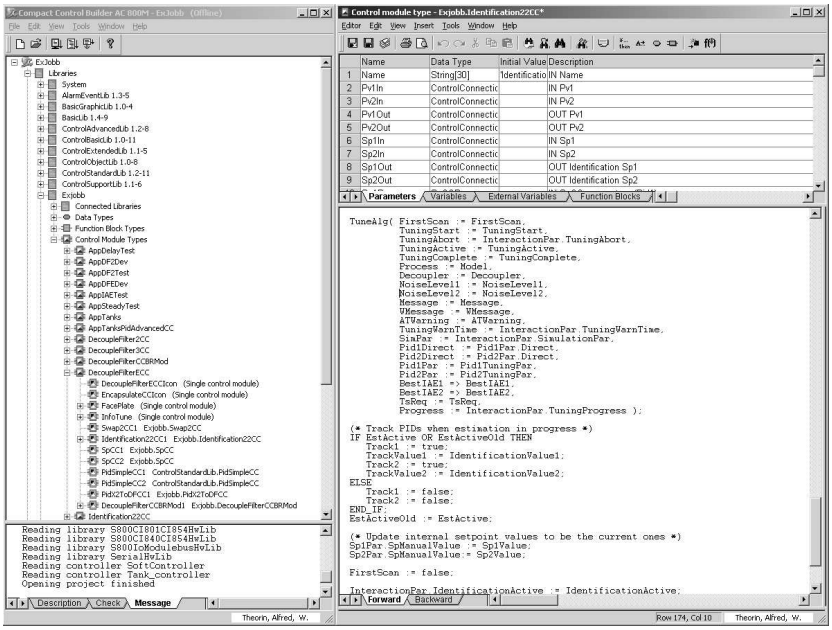

Figure 3.1: *Screenshot of a Control Builder working space.*

Targets for the CB applications are controllers, either hardware or software. For development the software controller is the most suitable since it is easier to set up and does not require more than a common desktop computer. Also the clock frequency and the amount of available memory is higher on a desktop than on a hardware controller, which means better performance. For all critical systems and most other real systems a hardware controller is preferable since it is much more reliable. Other advantages are that it has lower power consumption, it is easily replaced if it breaks, it takes less space and it can be placed almost anywhere, reducing the wiring costs.

Developing applications in the CB environment is somewhat different from ordinary software development. A great advantage is that the application can be altered while it is running, keeping all the current variable values.

Another advantage is that the execution order is determined automatically by sorting the code topologically, see figure 3.2. There are certain common cases where some variables cannot be topologically sorted and then one can tell the compiler not to sort these variables. This means that one has to consider the execution order when programming the parts containing these variables.

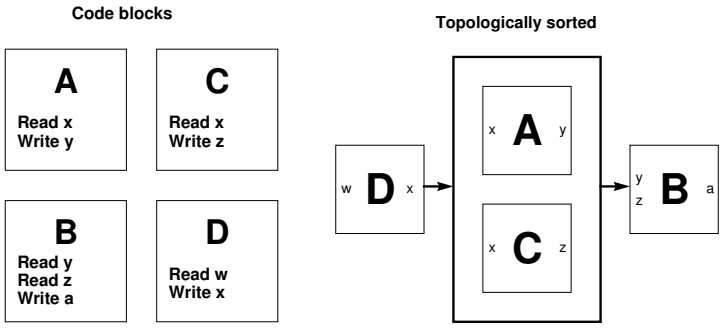

Figure 3.2: *A and C require the output from D, thus D will be executed first. B requires output from both A and C, thus B will be executed last. The internal execution order between A and C does not matter so any order is possible.*

A disadvantage is that ordinary functions have to be pre-defined in the controller's firmware so one cannot simply write small, simple functions to obtain clean and tidy code without writing altering the controller's firmware.

3.2.2 Structured Data Types

For variables with data that belong together, Structured Data Types can be defined. They simply are collections of variables of various Data Types and/or Structured Data Types. For instance the Data Type `Complex` would suitably contain the variables `Re` and `Im`, both of the Data Type `real`.

3.2.3 Function Blocks

Function Blocks are much like ordinary functions except that they do not return any values, which means they cannot be used in expressions. Results from a Function Block are received only through the Function Block's parameters. What makes them more powerful than ordinary functions is that they can have static variables which are separate for each instance of the Function Block. This makes them very useful when splitting up the code. One can place anything from smaller functionalities to complex sequential algorithms in a Function Block.

3.2.4 Control Modules

Control Modules may contain several code blocks, Function Blocks and even other Control Modules and they have their own graphical interface, see figure 3.3. They may be interconnected with other Control Modules, preferably graphically with the Structured Data Type `Control Connection` to form the complete control system. To denote that a Control Module uses Control Connections, by convention CC is appended to its name.

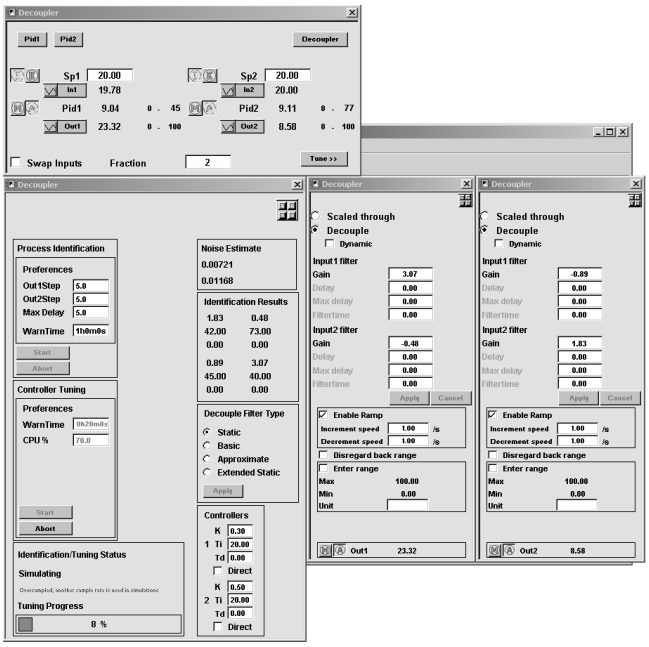

Figure 3.3: *Screenshot of the developed Decouple Filter's GUI.*

A Control Connection contains information like values, ranges and signal status. The Control Connection is split up into two separate parts, namely Forward and Backward. The Forward part contains information that is sent to the following Control Module and the Backward part contains information sent to the previous Control Module.

A Control Module which is connected to other Control Modules with Control Connections should also be split up into at least two code blocks, usually named Forward and Backward. Here the Forward block handles the Forward part of the Control Connection and the Backward block handles the Backward part of the Control Connection.

If properly written, all Forward blocks will be sorted to execute before all Backward blocks, making it possible to send information from any Control Module to any other Control Module within the same scan, assuming of course that they are connected and that the specific information is included in the Control Connection, see figure 3.4.

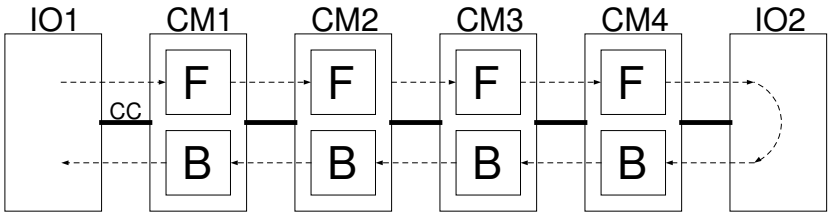

Figure 3.4: *Control Module execution order.*

This means that it is possible to know what happens any number of Control Modules ahead in the same scan as they happen, compared to the common case where information is only propagated one module backward each sample. For instance it is possible to avoid windup in the same scan as the IO reaches its limit, and it is possible to start controlling again in the same scan as the IO leaves the limit. Also this makes programming easier as one does not have to worry about how long time it will take for the information to reach a specific module, and one can write more independent code of lower complexity.

Of course not all information that is needed for any desired interaction with a Control Module is included in the Control Connection. Usually a Control Module also has its own specific interaction parameters, usually bundled in a Structured Data Type, which can be accessed by other Control Modules.

3.2.5 Logging

When running a Control Builder application on a controller there are a few ways to debug the application. In the GUI one can either view the current values or a history graph. One can also use the Online Editor, see figure 3.5.

The Online Editor shows the running code with all the *final* variable values for the current scan shown together with the variables. Most times this is sufficient but the refresh rate in the GUI and in the Online Editor is about once a second, so faster events are impossible to debug without adding latch variables to the code. Also, if deeper analysis or a higher quality plot is required, one needs to extract the raw data for each scan. Unfortunately there is no built-in log functionality so one has to resort to other means of extracting the data.

To read the current values from the controller one can connect to it with an `OPC server`. The server works as a bridge between `OPC client` applications and the controller. With the right client one can both read, view and log any specific data from the controller at pretty high refresh rates. It is not uncommon though that some value are missed either by the server or by the client so it is appropriate to store each variable set together with a time stamp. In this master's thesis the application `OPC Data Logger` from `www.opclogger.com` has been used.

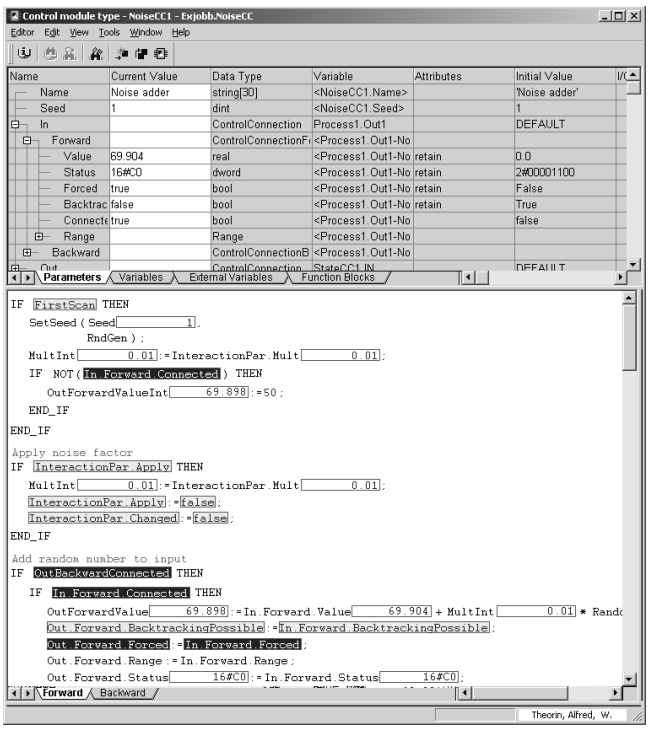

Figure 3.5: *Control Builder Online Editor*

3.3 System Identification - First Approach

The process will be modelled by a 2x2-matrix with FOWDT transfer functions as elements. A FOWDT is often sufficient in the process industry and the identification algorithm for SISO systems of this kind is both available and well-tried. However, implementing an identification algorithm for a TITO system is not as simple as running one SISO identification for each element.

The standard PID uses a relay identification which is sufficient to tune a SISO controller, but the information obtained from the relay identification is not sufficient to design the decouple filter. The available step response identification gives a FOWDT model and requires the results from a relay identification. Using the available algorithms means that for all four elements, both a relay and a step identification have to be run.

The relative gain between a coupling element and its corresponding diagonal element has to be known before starting the identification of the coupling element. It needs to be used to determine the maximum amplitudes for both the relay and the step identification algorithm, otherwise the diagonal process value will typically go far beyond its specified limits. No coupling at all can also cause problems, since then there is nothing to identify. The SISO identification also assumes that the process value is steady when it starts, but does not leave it steady as it completes. This means trouble when we run several consecutive identifications. Extra supervision will be needed to be able to handle all this.

3.3.1 Identification Parts

To integrate the available algorithms the system identification is split up into steps. The first three steps are supervision preparations and the last four steps each identify one transfer function. As mentioned in the introduction, the relative gain between the diagonals their corresponding couplings must be known before running an identification on the coupling element. With the watcher approach, see section 3.3.6, this means that the diagonal elements must be identified before their corresponding coupling element.

The identification steps are

1. Estimate noise
2. Estimate static gains (optional)
3. Decide which output shall control which input (optional)
4. Identify first diagonal element
5. Identify first coupling element (if possible)
6. Identify second diagonal element
7. Identify second coupling element (if possible)

The identification steps are in turn split up into phases

1. Stabilize inputs
2. Relay identification
3. Design temporary controller from relay identification
4. Step identification with temporary controller

3.3.2 Identification Supervision

The supervision algorithm is put into a separate Function Block. It tells the other algorithms what to do, and collects and manages their results. The algorithm also decides which values to consider during an identification, such as input, output and maximum amplitude. This reduces the complexity of the algorithm and also the number of required Function Block instances since only one set of identification algorithms is needed. Otherwise one set for each transfer function would be required.

3.3.3 The Steady Test

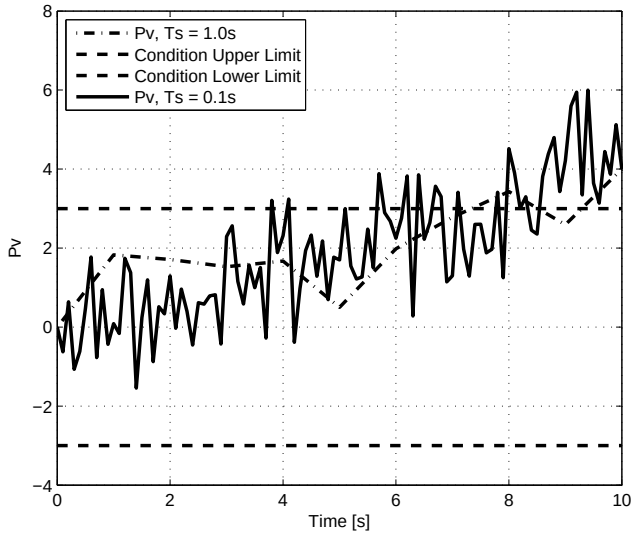

Figure 3.6: *With a proper sample rate the condition fails at the 8th point. With a 10 times higher sample rate the condition is successful and wouldn't have failed until at the 38th point.*

In many parts of the identification algorithm one has to await steady process values, for example in the stabilization phase before a relay identification, see

section 3.3.7, and in the initial estimate of static gains, see section 3.3.5. To determine whether the process values are steady the noise estimates are used. The steady condition is that a process value lies within 3 standard deviations 10 scan in a row.

For a steady process value there is a 99.7% chance that it lies within the 3 standard deviations. For 10 consecutive scans the chance is $0.997^{10} \approx 97.0\%$ and since there are two process values the total chance is $(0.997^{10})^2 \approx 94.2\%$. The risk that this happens by chance when the process values are not steady is negligible. Of course the process values will not have to be absolutely steady but the steady condition will only be satisfied when they are both moving very slowly compared to the noise level.

Here the condition has a weakness. If the sample frequency is chosen poorly and set too high by the operator, the process will still be moving at the same speed in the time domain but a lot slower in the sample domain, causing a larger movement to be accepted by the condition, see figure 3.6. Having a sample frequency of more than 100 times the process's bandwidth will cause problems in the application that uses the steady test algorithm.

Within 3 standard deviations could use some clarification. What is meant is the deviation from the initial point, not from the previous point, see figure 3.7. By using the previous point one would allow larger movement of the process value. For the test to fail, a process value would then have to move more than one to two standard deviations each scan, compared to one or two standard deviations in 10 scans.

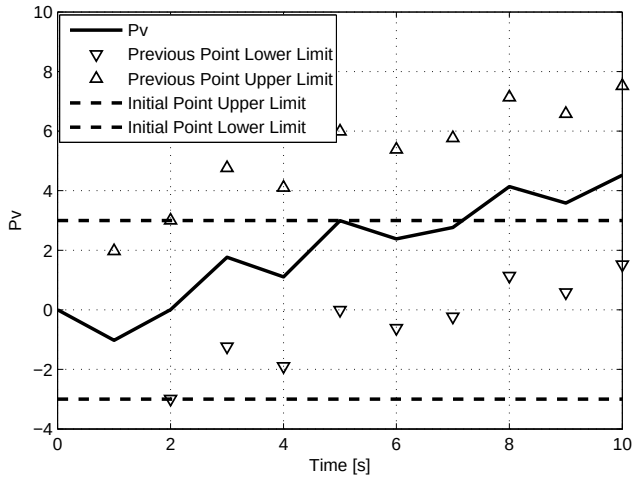

Figure 3.7: *With the correct interval used (dashed) the test would fail at the 8th point. With the previous point used (triangles) the test would incorrectly be successful.*

For most uses the Steady Test is a required but often not a sufficient condition. This will be discussed in detail where it is used.

3.3.4 Quantization

For process values with quantized levels, i.e. for all real processes, the steady condition has a flaw. If the process is steady when estimating the noise level the value will jump one quantization level up or down every now and then. The noise level will then completely be determined by how many jumps it does during the estimation. If the estimated noise level is less than a third of the distance between the quantization levels, every jump between two quantization levels will consider the process value non steady which may stall the steady test.

To handle this, the last initial process value can be saved, and as long as the process value jumps between the current and the previous initial process value it is still considered steady. In figure 3.8 the steady test has been run on a real process. The estimated noise level from the noise estimate algorithm was 0.010 and the step between two quantization levels was 0.032. To fit all values in the same figure the process value has been amplified with a factor of 6 and the counter with a factor of 0.1. Some consecutive values are missing in the log before $t = 50$, otherwise no significant measurement point was missed.

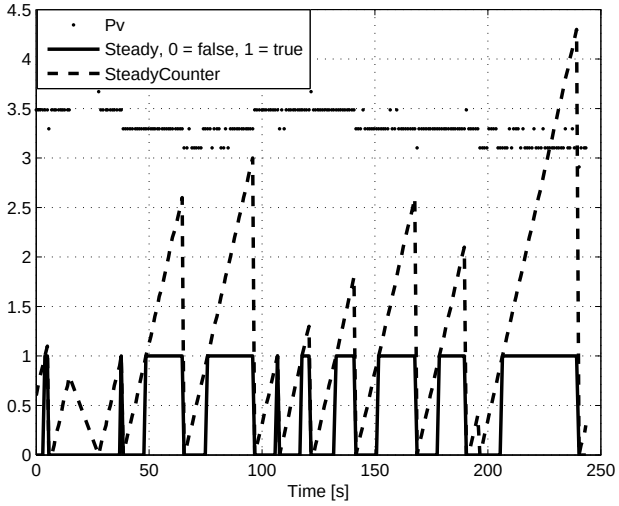

Figure 3.8: *The Steady Condition with added quantization level support run on a real process. The process value moves between 4 quantization levels and as long as it varies between the two latest initial values the SteadyCounter keeps increasing, otherwise it is reset to 0. The Steady Conditions is satisfied when the SteadyCounter is* ≥ 10.

3.3.5 Initial Estimate of Static Gains

As mentioned in the introduction the relative gains between the main and the coupling elements are required. They can be obtained by running two consec-

utive control signal steps, one on each output. After each control signal step, steady process values are awaited.

Here the steady condition is a necessary but not a sufficient condition. With dead times in the system, the process values may be steady simply because they have not started moving yet. To make sure that this is not the case another condition is added to test if any of the process values has moved from the initial value. A way to do this is to check if it has remained within a certain amount of standard deviations. Taking in account that the process values may still be moving slowly after the noise estimate step, 5 standard deviations should be enough. 4.26 standard deviations include all but 1 out of 10^5 values, meaning that if the dead time is 100 scans it still would trigger incorrectly only once in 1000 times. Checking the condition two scans in a row, the risk of incorrect triggering is negligible.

With the static gain estimates it is possible to handle incorrectly connected systems, i.e. systems which do not have larger gains in the diagonal. Calculating the RGA, Relative Gain Array [4], for the estimate will tell if the system is properly connected. If not, the input channels can be swapped internally to compensate for this. It is also possible to detect and compensate for negative gains.

Since the steady test condition is sensitive to oversampling, the initial estimate of static gains will only be an optional feature.

3.3.6 The Watcher

Since the initial estimate of static gains, see section 3.3.5, is not yet fully reliable a supervisional watcher for the other input is used instead. The watcher is active when identifying a diagonal element and estimates the gain by checking how much the coupling process value and the output signal have changed respectively during the setpoint step. If the coupling is relatively small or if there is no coupling at all the coupling identification is skipped and the element's gain is set to zero.

3.3.7 Stabilization

The stabilization phase is needed since the relay identification algorithm requires the process values to be steady when it is started. For all except the first identification a previous identification has just been run and the process values are typically not steady without having a phase between that stabilizes them. The stabilization algorithm uses the steady test, see section 3.3.3, which here is a necessary but not a sufficient condition due to possible dead times and sensitivity to oversampling. An additional condition is used, requiring the process values to be sufficiently close to their respective setpoints.

Thus it can be determined if the process values are steady at the setpoints

or if they are incorrectly considered steady somewhere else. Since it is more important for the relay identification that the process values are steady than that they are really close to the setpoints one could allow a pretty wide range around the setpoints to decrease the stabilization time.

The stabilization phase uses the temporary controllers designed by using the results of the relay identifications. When identifying the first diagonal element there are no results available so the controllers entered by the operator are used. This also applies when identifying the first coupling element and the second diagonal element but then only for the second controller.

With a too high sample frequency the process will be considered steady as soon as the setpoint is approached, which may be a problem if the system is close to instability due to poor choice of controller parameters by the operator. In a robustness perspective this is not acceptable, thus it is important to improve the steady test, to avoid using the controllers defined by the operator.

3.3.8 Maximum Amplitudes for Coupling Identification

The relative gain between the diagonal and the coupling element is needed before starting the coupling identification to be able to set a proper maximum amplitude during both the relay and the step identification of the coupling. A small setpoint step on the couple input will mean a larger step on diagonal input.

Example 3.3.1. *Say that the relative gain is* $\frac{1}{10}$ *and that the operator has limited the diagonal element's step amplitude to* 10. *Then the setpoint step during the coupling identification must not be larger than* $10 * \frac{1}{10} = 1$.

If the calculated step amplitude is not considerably larger than the noise level then the coupling element is set to have a zero gain and the identification of the coupling is skipped.

3.3.9 Relay Identification

In the available PID Control Modules the first and usually only identification is a relay identification. The relay identification requires a noise estimate and in the current implementation it cannot be entered through the parameter list and has to be run internally. As always a noise estimation is sensitive to moving process values and this is not handled properly by the current implementation. This can be defended since the implementation is developed for SISO systems and then this is no big deal. If the process value was not steady and the identification stalls, the identification can simply be aborted and restarted again when the process value has stabilized.

When implementing an automatic TITO identification algorithm this causes big problems since another part of the identification just has been completed

and the process values are most likely not steady. This is solved by having a stabilization phase, see section 3.3.7, before the relay identification phase.

When the relay identification is completed, a controller to be used during the rest of the identification is designed based on the results from the identification. This is done because the results from the step identification, see section 3.3.10, depend on the controller performance. Also the stabilization time between the steps is reduced by using a more suitable controller.

3.3.10 Step Identification

The available step identification algorithm starts by choosing a setpoint and then it waits for the controller to stabilize the process value to that setpoint. Then a step is made on the setpoint and the controller controls the signal to the new setpoint. Using the information from these two steps the algorithm makes a control signal step without any controller active and it is during this step that a FOWDT model is estimated. The step identification algorithm requires an already working controller which is designed with the results from a relay identification, see section 3.3.9.

3.4 System Identification - Second Approach

The supervision algorithm in section 3.3.2 become pretty complex, just to be able to use the existing identification algorithms. Also the identification took a very long time. An attempt to decrease the complexity and identification time was made by extending the initial gain estimate algorithm in section 3.3.5.

It still has the same weakness as the initial gain estimate, i.e. sensitivity to oversampling, with no means to determine whether or not the system actually is oversampled. It can however be detected after a completed identification by comparing the sample frequency and the estimated time constant, but this requires that the system is not too oversampled to get a reasonable estimate. Two identifications run on a real process with different grades of oversampling can be seen in figure 3.9 and 3.10.

In figure 3.9 the system is oversampled by at least a factor of 2 and in that case the estimate is still OK. In figure 3.10 the system is oversampled by at least a factor of 8 and it can be seen in the figure that the estimate has lost accuracy. In both cases it is possible to detect that the system is oversampled.

The algorithm does two steps on each one of the out values. First, steady process values are awaited using the steady test. Then the first step occurs and the gain and dead time is estimated. The step is considered complete when both process values are considered steady, by the steady test.

To know for how long to wait at most a maximum dead time has to be specified. If the process value does not move during this time, a gain of zero is assumed.

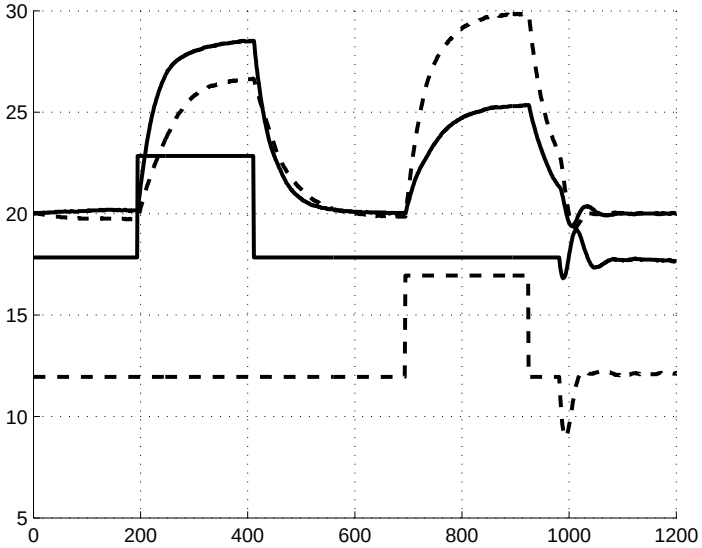

Figure 3.9: *The identification algorithm run with a little too high sample frequency. The first step is considered complete a little too early but does not affect the estimates considerably.*

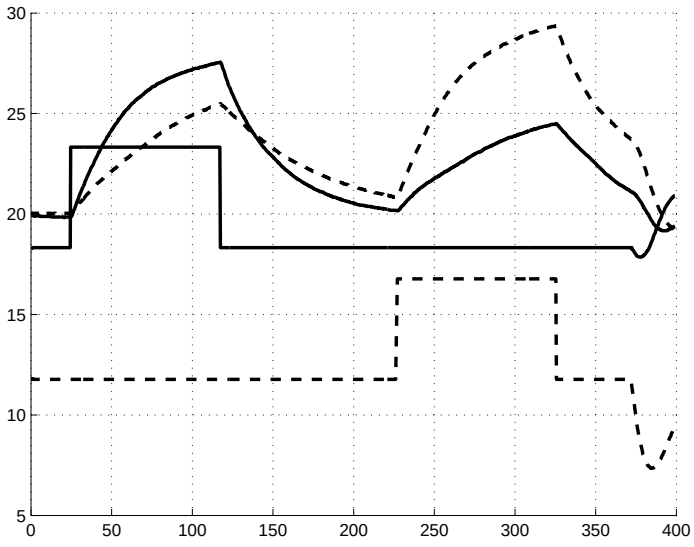

Figure 3.10: *The identification algorithm run with far too high sample frequency. The first step is considered complete way too early, leading to bad estimates.*

Then the second step is applied and the time constant for the system is estimated by measuring how long it takes between passing two given levels, the first at 5% and the second one at 65% of the estimated gain. The step is considered complete when both process values have passed the second limit.

The results from identification of simulated FOWDT processes have all been very good. The results from identification of the quadruple tank were close to the values estimated values by PidAdvancedCC, so aside the sensitivity to oversampling and perhaps worse accuracy for approximating processes of higher orders, the identification algorithm has no obvious flaws.

For systems with properties that have time constants of different orders, an identification algorithm sensitive to oversampling will never be able to get a good estimate of the slower system. The sampling rate has to be higher than the fast system which will be too fast for the slow system.

3.5 A First Approach

An initial attempt was to use two instances of the existing PidAdvancedCC Controller, which already has all the required identification algorithms implemented. The existing decouple filter would be extended to remotely manage the system identification in the PidAdvancedCCs and extract the identification results. Based on the results the decouple filter and the PIDs would be tuned.

Unfortunately the possibilities of remote interaction with the PIDs was too limited, there was no way to access the identification results or even see if the identification was completed. Thus this approach had to be discarded.

3.6 A Second Approach

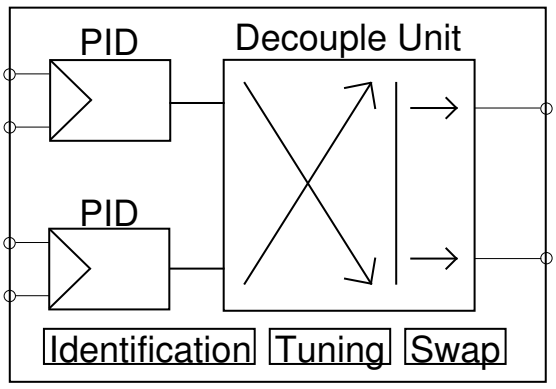

Figure 3.11: *Overview of the second implementation approach.*

Instead of trying to interact with existing Control Modules, this approach embeds the Function Blocks containing the PID, identification and tuning algorithms. All information and results are then available to all the other parts. The existing decouple filter was used as a base, see figure 3.11.

Note that with this approach the parts are just split up in thought as they all coexist in the same code block. The same applies to the data flow.

This approach seemed good at first but when the Forward part was pretty much completed and the Backward part was about to be written it was obvious that with all the Function Block calls and with all variables available in the entire Forward code it was very hard to keep track on in which order all variables were allowed to be manipulated. Especially it was pretty much impossible to keep track on all the ranges and where in the code to use which. The module had become too complex to be able to write the Backward block correctly. Even if one would succeed, one would need to duplicate a lot of code from the existing PID modules. All in all the code would be completely unmaintainable. Some way to split up the module into smaller, more overviewable and maintainable blocks had to be considered and this approach was discarded.

3.7 The Final Approach

Even though the big and complex Control Module from the second approach, see section 3.6, was completely discarded, the main idea was preserved and many parts from it were reused. However, instead of trying to integrate everything in the code blocks of one Control Module, this approach aims to encapsulate smaller, less complex Control Modules within another Control Module, internally interconnecting them with Control Connections, see figure 3.12. This has a great maintainability advantage since each block can easily be overviewed, debugged, altered or replaced. Also, if there are sufficient interaction parameters included, Control Modules such as the existing PIDs and the Decouple Filter can be used.

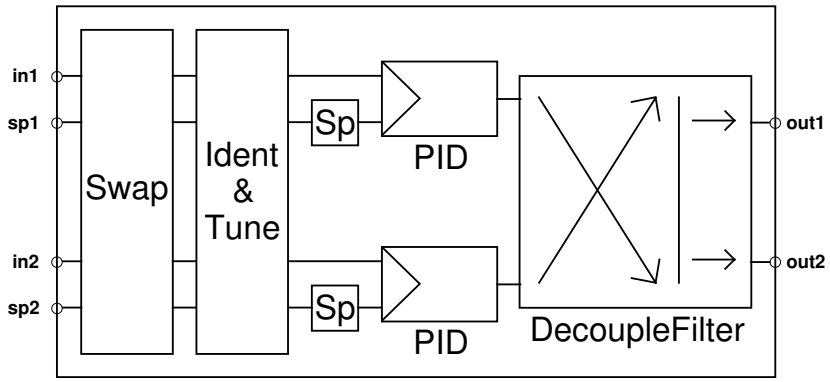

Figure 3.12: *Final approach Control Module overview*

The complex execution order from the second approach which was determined by the code order in the code block and only split up into parts in thought was now split up for real. This means that one easily can make sure that the parts are executed in the correct order and that they do not accidentally alter or use each others' ranges, values etc.

Another advantage is that the existing GUIs for the separate Control Modules may be imported if they do not allow modification of values which must remain unchanged or otherwise coordinated in order to work together as a complete module.

The only disadvantages with this approach is that the data is harder to access, and one depends on that the encapsulated Control Modules have sufficient interaction parameters to be able to coordinate them. Each Control Module's interaction parameters are connected to the encapsulating Control Module and then connected to the other parts that need access to them.

The requirements for the PID Control Modules due to the remote identification are remote setpoint setting and remotely requested forward value tracking. Also it has to accept the backward range automatically, i.e. the recommended range sent backward from the following Control Module. In chapter 2.4 it is explained why a dynamic range is required and why the range has to be accepted. Already the simplest of the PID Control Modules, PidSimpleCC, satisfies all these requirements and thus it has been chosen for the implementation.

The decouple filter is required to set limits on the control signals remotely, inhibiting them from increasing/decreasing further if a limit is reached after the decouple filter. The remote tuning also requires that the decouple filter transfer function can be set remotely. Also it must be able to set a proper backward range. The current decouple filter calculates an overly pessimistic backward range which only guarantees that all possible output values can be reached, resulting in an unnecessarily big backward range, containing lots of forbidden value combinations. Also it lacks proper backward limitations which will cause wind up problems. Since this is the only decouple filter Control Module available it had to be extended.

The other Control Modules in figure 3.12 will have to be implemented, but they are either very simple or have most of the code available from the second approach.

3.7.1 SwapCC

The SwapCC Control Module adds the feature to handle incorrectly connected systems. SwapCC makes it possible to swap the input channels if the coupling elements are larger than the diagonal elements and is essential for the identification algorithm that was used in the second approach. Since the connection status suitably is determined by the identification algorithm, the swap state must be in the module's interaction parameters.

SwapCC is required for the first identification approach but not for the second, which is used in the final implementation approach. However, without it, a less logical fix is needed instead as to make it easier to go back to the first identification approach it seemed quite rational to keep this part.

3.7.2 Identification22CC

The Identification22CC Control Module supervises the identification and needs to interact with both the PIDs and the decouple filter. The identification approach in section 3.4 was used.

3.7.3 PidX2ToDFCC

A Control Module, PidX2ToDFCC, had to be added between the PIDs and the Decouple Filter to get the control signals and the output ranges to the GUI. This is the sole purpose for this Control Module.

Chapter 4

The Tank Process

4.1 The Process

This master's thesis wouldn't feel complete without testing the implemented module on a real coupled process. An ABB hardware controller, PM864, was connected to the tank process from the department of automatic control in Lund, see figure 4.1.

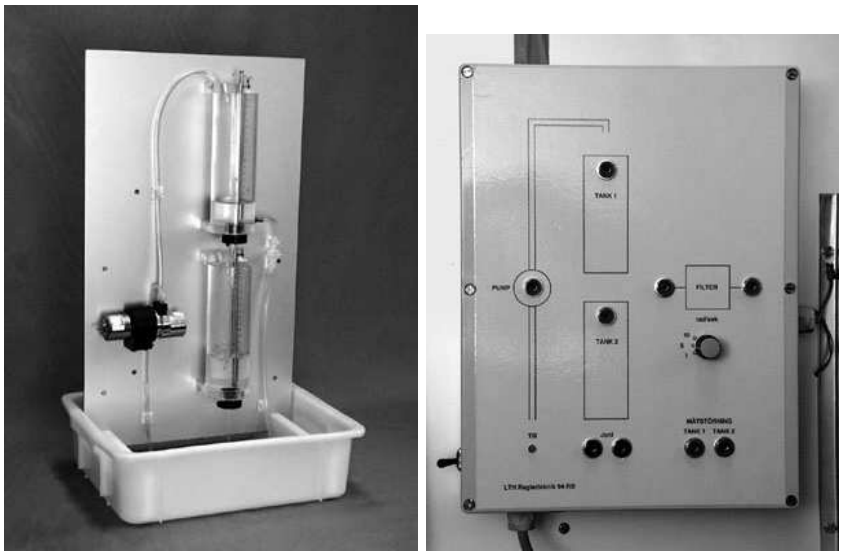

Figure 4.1: *The tank process consisting of two tanks with a small hole in the bottom. The water is pumped into the upper tank and then flows down into the lower tank, then out of the process.*

One tank process consists of two water tanks and a pump. The water tanks have a small hole in the bottom where the water flows out and are placed so

that the water which is pumped into the upper tank then flows down into the lower tank. The water levels can be measured for both tanks and it is the level in either tank that is controlled.

To get a coupled TITO process, two tank processes were used, splitting the water from both pumps into both of the upper tanks, about 70% into the primary tank and about 30% into the secondary tank.

4.2 The Experiment

Since the process is non linear it is important that the water level in the tanks are the same in all experiments and here 40% have been chosen. For the experiments it is the water levels in the upper tanks that are considered for control. For the upper tank there is a valve which can be opened manually to lead water directly out of the process. This is used to simulate load disturbances. Much care have been taken to open the valve the same amount and in the same manner in all experiments.

After the identification and tuning was complete, a setpoint step was applied. When the process values had stabilized a load disturbance was applied.

4.3 SISO Control

For reference SISO Control will be evaluated, testing both the default parameters and the tuned parameters in PidAdvancedCC.

4.3.1 Default Parameters

The tanks were connected to one PID controller each, using the default values. See figure 4.2.

4.3.2 PidAdvancedCC

The tanks were connected to one PidAdvancedCC each, using the included identification and tuning algorithms to tune the PIDs one at a time, see table 4.1.

The proposed controller was applied and the experiments were performed, see figure 4.3.

Compared to the default settings the controller is a little more aggressive but has almost the same control performance. That the difference is not bigger is because the default parameters happen to fit this particular process pretty well.

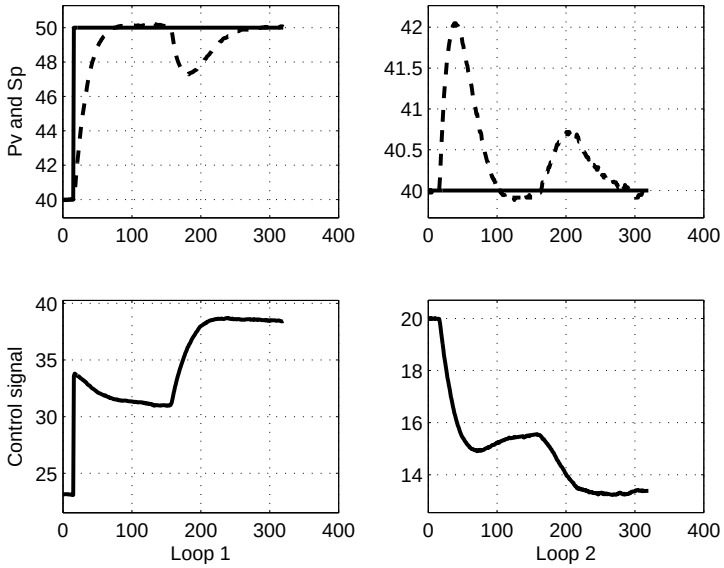

Figure 4.2: *The experiment run on the tank process with SISO control using the default controller parameters.*

	Tank 1	Tank 2
Gain	1.65	3.23
Filter time	41.98	69.30
Dead time	0	0
K	1.33	1.10
Ti	12.87	14.80
Td	5.41	6.25

Table 4.1: *Identification and tuning results from PidAdvancedCC.*

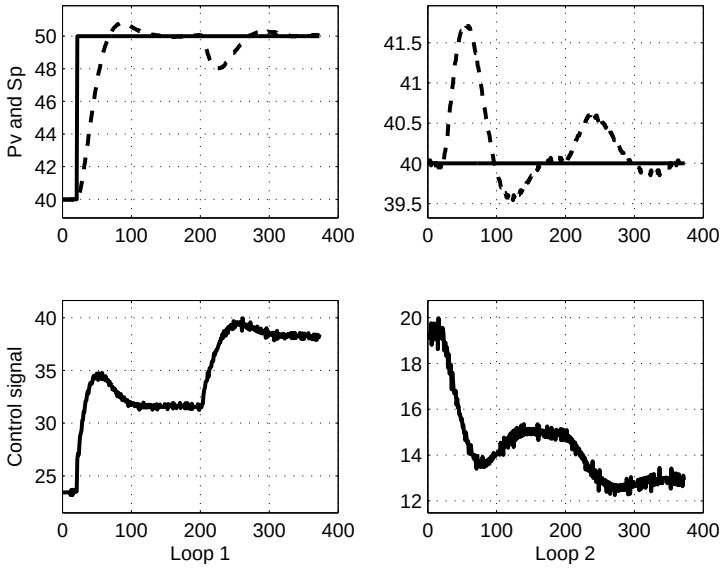

Figure 4.3: *The experiment run on the tank process with SISO control using the controller parameters from the tuning in PidAdvancedCC.*

4.4 TITO Control

Using the implemented decouple filter the system was identified and then tuned for two decouple filter types, i.e. static and FOWDT.

4.4.1 TITO Identification

The identification was run with a slight oversampling, see figure 3.9. The identification results are shown in table 4.2.

		x1	x2
	Gain	1.91	1.12
1x	Filter time	44.00	63.00
	Dead time	0	0
	Gain	1.7	2.86
2x	Filter time	76.00	63.00
	Dead time	0	0

Table 4.2: *Identification results from the TITO identification.*

	Tank 1	Tank 2
K	5.66	4.32
Ti	29.03	46.71
Td	0.02	0.02

Table 4.3: *Tuning results using a static decouple filter.*

4.4.2 Static Decouple Filter

The TITO identification results were used to design the static decouple filter and tune the PIDs. The tuning results are shown in table 4.3 and the experiment results in figure 4.4, note that in this specific experiment there are two load disturbances.

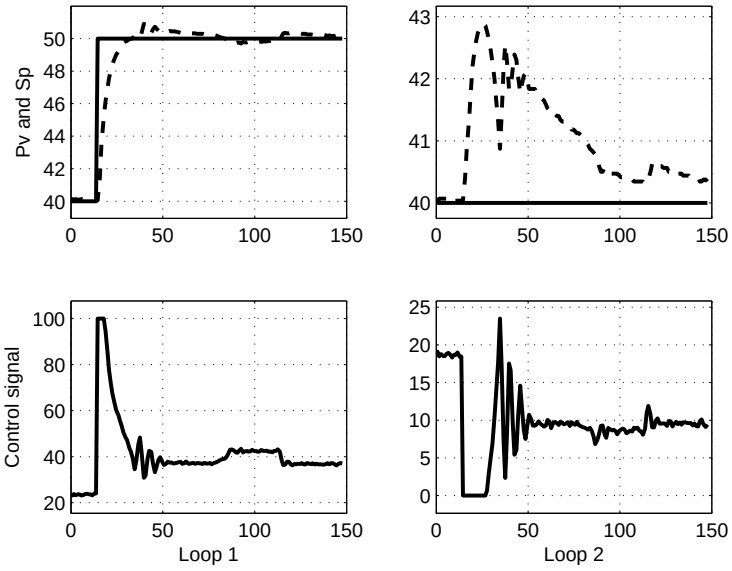

Figure 4.4: *The experiment run on the tank process with a static decouple filter and tuned PIDs.*

The control performance here is good, especially the setpoint step speed and the load disturbance attenuation which is barely noticed. On the other hand, the control signal is really noisy and there are big overshoots at the setpoint step which is really bad and is a sign of poor robustness.

4.4.3 FOWDT Decouple Filter

The TITO identification results were used to design the FOWDT decouple filter and tune the PIDs. The tuning results are shown in table 4.4 and the experiment

	Tank 1	Tank 2
K	37.15	37.15
Ti	90.92	90.92
Td	6.33	6.33

Table 4.4: *Tuning results using a static decouple filter.*

results in figure 4.5.

Note that the controller parameters are the same due to the process model satisfying the conditions for the special case in section 2.3.6.

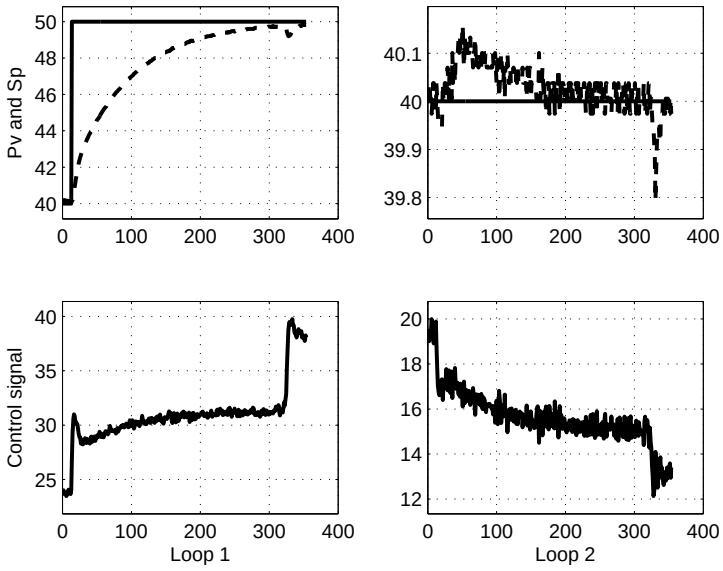

Figure 4.5: *The experiment run on the tank process with a FOWDT decouple filter and tuned PIDs.*

The control performance increases considerably with the FOWDT decouple filter. The setpoint step response is slower but the load disturbance attenuation is very good. The control signal on the other hand is really noisy compared to the other controllers.

Chapter 5

Summary and Future Work

A Decoupling TITO Controller module with all essential functionality has been implemented. It has been run on a real process and it was confirmed that it really works. Most parts of the module work really well but there are a few things which need further improvements to increase the module's robustness and performance.

5.1 TITO Identification

The new identification algorithm that was introduced in section 3.4 is faster and less complex than even the supervision part in section 3.3.2. Also it is easier to extend to MIMO systems with more inputs and outputs. However its sensitivity to oversampling makes it less suitable since the sample time often is chosen without considering the speed of the system.

A way to make the identification algorithm more robust would be to initially run relay identifications which would give the speed of the system. Knowing this, the identification algorithm can be developed to handle oversampled systems and systems with time constants of different order by discarding samples at a suitable rate in each steady test algorithm.

5.2 Decouple Filter Design

Decouple Filter Tuning

In chapter 5 in [1] a tuning of the decouple filter after it has been applied is proposed but has not received any attention at all in this master's thesis. It requires some additional identifications of the system with the decouple filter applied but it will results in a better process model, resulting in better decoupling.

It should be analyzed whether or not it is worth implementing.

Decouple Filter Normalization

The decouple filter design methods use the adjunct instead of the inverse. A problem with this is that the gain of the system will vary much between different systems. A process with small gains will result in a decouple filter with small gains. The total gain that the controller will see will approximately be the process gain squared. This means that the controller gain needs to be very big if the process gain is small and very small if the process gain is big. Thus constant limits in the controller design, K_{max} and K_{min}, should also vary with the system's gain.

An easier and more suitable way to solve this is to use the inverse instead of the adjunct when designing the decouple filter. Analysis of whether or not this has any disadvantages, or if there are more suitable ways to normalize the decouple filter is needed.

Need of Decouple Filter

There are certain types of systems which do not need a decouple filter and adding one anyway only introduces unnecessary dynamics. If the time constants differ much between the two properties, there is typically no need to reduce the coupling from the slower to the faster property since it will be compensated well by the other, faster controller anyway. In such cases a triangular decouple filter is preferred.

5.3 Controller Tuning

Tuning Time

There is need for improvements and optimization of the controller tuning algorithm. Currently it takes about one hour to execute the tuning algorithm. With other means than exhaustive search, the time it takes to design the controllers can be reduced.

Controller Robustness

The controller is designed to meet a specific robustness condition but testing the controller's robustness has not been done. How well the decoupled system handles process variations or higher order should be analyzed.

As mentioned in section 2.3.4, some controllers make the discretized system

unstable even though they satisfy the robustness condition for the continuous system. Some possible causes are the stability test algorithm, too few Nyquist curve evaluation points or the actual discretization which might be solved by using the discrete Nyquist curve instead. This should be analyzed further.

5.4 Bumpless Parameter Changes

When the decouple filter's parameters are altered there are really nasty transients. There is functionality to ramp when changing parameters but it should be possible to use backtracking to avoid the transients without ramping.

Appendix A

Glossary

CB	Control Builder
FOWDT	First order with dead time
IAE	Integrated absolute error
scan	A controller sample, during which all associated applications are executed once
SISO	Single input single output
TITO	Two inputs two outputs

Appendix B

Source Code Samples

B.1 Controller Gain - Binary Search

```
(* A Binary search algorithm *)

(* Initialization *)
LastBadK := KMax; (* Last failed gain test *)
LastGoodK := 0; (* Last successful gain test *)
CurrK := KMax; (* First gain to test *)

(* The quotient LastGoodK / LastBadK will start at 0 and
   approach 1 as the remaining interval becomes smaller *)
WHILE (LastGoodK / LastBadK < 0.99) AND (CurrK > KMin) DO

   ...
   (* Test if the controller with gain CurrK is stable and
      satisfies the robustness condition *)

   IF ControllerOk THEN
      LastGoodK := CurrK;
   ELSE
      LastBadK := CurrK;
   END_IF;

   (* Next K to test is in the middle between LastGoodK and LastBadK *)
   CurrK := (LastBadK + LastGoodK) / 2.0;
END_WHILE;

(* LastGoodK is used as controller gain in simulation *)
(* If LastGoodK = 0 then no stable controller was found with K in [KMin, KMax] *)
```

B.2 Span Noise Estimate

```
IF ScanCounter = 0 THEN
   (* Initialization *)
   PvHigh := PvMin; (* Lowest possible Pv *)
   PvLow := PvMax; (* Highest possible Pv *)
END_IF;

IF ScanCounter <= EstimateScan THEN
   (* Store extremes *)
   PvHigh := max(PvHigh,Pv);
   PvLow := min(PvLow,Pv);
ELSE
   (* Calculate estimate *)
   NoiseLevel := 0.5 * (PvHigh - PvLow);
END_IF;

ScanCounter := ScanCounter + 1;
```

Bibliography

[1] Pontus Nordfeldt, *PID Control of TITO Systems*

[2] Karl J. Åström & Tore Hägglund, *Advanced PID Control*

[3] Gunnar Blom & Björn Holmquist, *Statistikteori med tillämpningar*

[4] Karl J. Åström & Björn Wittenmark, *Computer-Controlled Systems*