University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/Specialization: Master of Information Technology Automation and Signal Processing	Spring semester, 2014 Confidential
Writer: Eivind Brate Midtun	Eivind Brate Midtun (Writer's Signature)
Faculty supervisor: Morten Tengesdal External supervisor(s): Eivind Bjørge Sandsmark, Statoil	
Thesis title: **"Study on a Solution for Condition monitoring of Process, Process Equipment and Control Loops, and efficient System identification for retuning"**	
Credits(ECTS): 30	
Key words: Control systems, instrumentation, modelling, system identification, state detection, performance analysis, intelligent control loop monitoring	Pages: 84 + enclosure: 44 Stavanger, 27/6-2014 Date/year

Preface

***When I first left** my friends and family in Bergen to start my higher education at the University of Stavanger, blissfully was I unaware of all the academic challenges that would be thrown at me during the years. Being a student is often associated with living on bread crumbs and water (thankfully not all the time), like an imagined prisoner, but the prison that surrounds you is actually the lack of the formal education you need in order to start the job career of your desire.*

***I have been lucky** and quickly selected the field of work appropriate for my gray cells. Although the course sounded very interesting when I first read the description, there was little information provided from people with the same interests as you, but on a later life phase, ahead of making the career choice. So the randomness of nature sent me to Stavanger, which I in retrospect do not regret.*

***Ever since I was a little boy** I have physically realized my imagination in different forms; back then I had woodworking from growing up on a farm, drawing from my mother's father, and of course the Lego that I always wanted for the birthdays and Christmases; and it became a lot over the years, and eventually I got my hands on a Lego Mindstorms set. I have more tools in my toolbox now, and the products of my labor are more beneficial for others. I take pride in my work and consider myself ambitious; sometimes ambitiousness throws you down paths you did not expect, but having learned to cope with challenges, for good and for worse, has made me aspire to be a strong engineer.*

***There are many people** to thank for being given this opportunity and it is too easy to take it for granted. But I know that without support from my family and friends, the journey would have been a cruder. I will allocate more time for you now.*

Thanks *to all my teachers during the years. Thanks to the University of Stavanger for providing such great lecturers and to aspire to be a great University. Thanks to my supervisor for giving me this opportunity, and to the rest of the crew at Statoil Statfjord Plant Integrity in Stavanger who has put up with me for 6 months straight. I hope that we can still meet at least once a week and play football together. Thanks to all my friends, that I too often prioritized second to my education, but I will make it up to you now.*

And especially *a big thanks to my family, whom I love more than anything.*

Sincerely,

Eivind Brate Midtun,

Master of Technology,

Faculty of Science and Technology,

University of Stavanger,

2014

Abstract

The presented thesis serves as part of a technology development program by Statoil ASA. The idea behind the technology seeds back several years, and is a product of hands- on- practical experience. Extensive experience has indicated feasibility of more work toward understanding the principles in action and approaching a practical solution for use on physical plants.

The idea is registered internally at Statoil with Statoil reference K4064, and the current intellectual property strategy requires all information relating this idea, including this master thesis, to be qualified as 'Confidential'.

Introduction to the technology plan [1] defines the prime motivation of the technology development:

There is a growing motivation for monitoring control loop performance and for a change in maintenance strategy towards more condition based maintenance. Monitoring control loop performance and equipment condition enables for early action when deterioration of a control function starts developing, i.e. before the developing fault(s) or changes in process characteristics has had a significant negative effect on business.

The proposed technology is a practical method for superficial system identification and technical/operational state detection, with the goal of optimizing process component maintenance, and control loop retuning. The vital components of the plan will be properly detailed in the thesis outline, **1.3**, with the associated deliverables and constraints. A description of the proposed test mechanism will be provided under the theory chapter, **2.3**.

Also as part of the same Statoil Technology Development program, a bachelor thesis [2] was conducted in 2010, by Espen Svandalsflona and Frode Tuen associated with the University of Stavanger, concluding among other topics that more study on the results criteria for making the maintenance decisions would have to be made and review changing several process parameters in combination. Additionally it would need to be proved that oscillations induced by the proposed test mechanism would not upset other parts of the process to such a degree that the desire for increase of product quality, effectiveness and the cost advantages, would be overshadowed by the potential performance loss associated during the testing. The latter topic will not be discussed in this thesis due to the time constraint.

Table of contents

Acronyms

This section contains clarification of commonly used terms in order to lighten the reading, as well as a complete list of figures and tables for convenience. If something is unclear or needs elaboration, the reader may have use of referring to this section.

Term/symbol	**Meaning/annotation**	**Comment/explanation**
Stiction		*Static friction. A combination of stickband and slipjump*
Slipjump		*Example: A steady increase of throttle results in an abrupt change of value*
Hysteresis		*Example: Input must exceed a certain value after reversing direction before reaching its previous configuration*
Stickband / Deadtime		*E.g. time before system reacts to an event on its input.*
Pv	*Controlled variable*	
Op	*Controller output*	
Sp/ref	*Set point*	
Mv	*Valve positioner*	
MTBF	*Mean Time Between Failures*	
RCM	*Reliability Centered Maintenance*	
SFC	*Statfjord C*	
S-function		*Used to extend the capabilities of Simulink. Matlab code is compiled to c-code on runtime.*
FFT	*Fast Fourier Transform*	
NaN/NA	*Not a Number/Not applicable*	
DNF	*Did not finish*	
RPD	*Reference-process-deviation*	
FOI	Frequency of interest	
$\omega_{180}/\omega_{-180°}$		Frequency of when Pv has a phase lag of -180° relative to the reference
ω_c		Frequency of zero dB gain of loop transfer function
ΔK		Gain margin, stability criterion and control performance indicator
φ_m		Phase margin, stability criterion and control performance indicator

Chapter 1
Introduction

The introduction chapter will start off with giving a small insight of the issues presented in this thesis. The author will discuss some of the objectives of having control loops governing processes, and some concerns associated with this. Section **1.2** will introduce some of the related work done within the field of control performance monitoring, as well as suggest some of the main differences between the mentioned solutions and the one that will be proposed in this thesis. In section **1.3** the work of this thesis will be further and more detailed described, and also the scope of the work will be clarified by listing constraints, initial simplifications, and the most important deliverables.

Note: During the report some abbreviations or standard annotations may be used, and their meaning or description is placed in the Acronyms table after the table of contents. Figures and illustrations are also listed in a table at the end of Appendix for convenience.

1.1 Model based control systems

Control systems are widely used in real life environments. The controllers are small, and built on a solid foundation of mathematical theory and practical experience. The controllers collect information of the process through the inputs which is then processed. The controllers further outputs reactions to physical actuators that can influence the process image, like valves and pumps, with the intention of keeping or bringing the process into a desired state. The use is growing and getting more extensive and our modern technology depend on them. Developing control systems is often about automating a task that is too costly, too rapid or too dangerous for humans to control. Good control systems require knowledge of the system, and in order to gain great performance, a lot of time is often sunk into the system- design and modeling, although the modeling is usually much simplified. The control performance can be measured in the forms of lower energy consumption, less environmental consequences, heightened safety, greater product quality, and so on according to the control system's operational area.

The dynamics of the sensors and devices included in the process are given by specifications provided by the equipment manufacturers, and although some operational drift often is informed by the manufacturers, it can be unreliable or impractical to compensate for in reality (and may often work against the intention). Since all physical sensors and devices are exposed to operational drift due to physical strain, the control system will change[1] over time resulting in lowered efficiency and less of the desired performance. If the state of the system has reached a state of undesired or critical amount of lowered performance, a group of engineers can be called to conduct a survey on the system and either tune the controller parameters if the state of the process equipment is adequate, or else replace/run maintenance on key components believed to be responsible for the deterioration. Retuning controllers to compensate for erroneous or bad behavior elsewhere in the process can often be considered a bad thing. Such actions may keep the faults partly hidden until their magnitude is so high that the consequences may be more severe, than dealing with them immediately; such as a plant shutdown. If changes have been made to the physical process, the control parameters should also be updated. E.g. replacing a transmitter (possibly even with another type) may directly change the process gain, which calls for an adjustment of the control parameters. Other elements affecting the operation of the controlled directly or indirectly process can also in most cases be regarded as dynamical and time variant. Such elements can be among change of flow into the process and change of pressure. This can be caused by change of set points. In the scene later described in this thesis, and which is the test site for this technology development, this can be a choice derived from the constant change of oil well properties. To summarize: There is a lot to keep in mind while designing and tuning process control loops, and the need of having a broader perspective covering all the typically thousands of active control loops, yet simultaneously individual and accurate health assessment for each component, is of high desirability.

1.2 Previous related work in the field

Assessment of control loop performance is hardly a new topic. Industry is and always will be results driven, and having optimally performing control loops is a key property in order to achieve this. Unfortunately, manually keeping a close eye on every control loop, and all its consisting equipment on a large scale is not feasible. An industrial plant may consist of thousands of control loops, and only a few process control engineers to ensure they are performing satisfactory. This

[1] Control performance can either be improved or degraded over time. The controller is not necessarily "perfectly" tuned upon commissioning.

suggests that automatic monitoring tools to flag and indicate degradation and source of problems as they arise would be of great assistance. For such monitoring systems to be helpful, they cannot be allowed to further complicate the process image, which hints to the use of relatively simple systems that require little attention before implementation and in addition decent flexibility and adaptability. Such monitoring systems already exist, and most associated technologies are often based on analysis of collected data during operation. Most of them are non-invasive and/or model free methods. Just to mention a few of such products that are commercially available, known to the author [3]:

- ABB: Loop optimizer suite
- Honeywell: Loop scout
- Matrikon: ProcessDoc

Many technologies are directed toward a data driven, analytic approach. Such technologies have a strong mathematical base and the methods care little about the particular process itself. That means the mechanism cares little about what the process consists of and what causes the dynamics, but is rather focused on the dynamic relation of input- output data during operation. Minimum variance controller is derived using such a “blind” approach. These methods are very adaptable and sturdy, and can give some insight of the control loop performance without too much adaptation of the mechanism. Although they provide a good understanding of the performance they lack the insight required to identify causes and suggest precise solutions. In many practical situations “bad” or “loose” control is even desired. What we want to achieve would then rather be high precision and individual follow-up. To list some of the experienced challenges of today’s typical solutions:

- **1:** Looks only at parts of control function
- **2:** High sensitivity to noise and process disturbances
- **3:** Further analysis usually required to perform diagnosis
- **4:** Significant uncertainty
- **5:** Difficult to evaluate a loops performance against specific controller objectives and constraints
- **6:** Tools available typically implement only a few of several strongly related business processes which should be integrated

- **7:** Lack of awareness of operational and technical states

The proposed test mechanism has been proved in collaboration with the project supervisor to respond superior on these challenges and the results can be summarized by:

- **1:** Evaluates the performance of the complete control function
- **2:** Low sensitivity to noise and process disturbances
- **3:** Results from test, preprocessing and analysis is directing towards the actual problems
- **4:** Low uncertainty
- **5:** Results are used for retuning with high precision
- **6:** Test mechanism implement all business processes
- **7:** Awareness of operation and technical states

The work presented in this thesis will be a more practical approach for state detection, while the comparable results used for providing the process "footprint" used for developing the solution (which will henceforth be referred to as the "test mechanism") are generated through simulations of the chosen process of interest.

1.3 Thesis outline

The scope of this thesis will be directed toward the field of condition monitoring, detection of fault modes and deterioration of control loops, where the objective is to present a robust indicator for lowered performance and early detection of equipment fault and/or drift. This will be used to better schedule maintenance and retuning of the control loops, which in return yield expectancy of increased production efficiency, increased product quality, and further reduced environmental impact. In addition, preemptive detection can cause fewer emergency shutdowns on the plant, as well as reduce the load on the operator.

The work is as mentioned in the abstract based on a Statoil technology development plan (TDP). The decided approach is a mechanism based on a modified relay test, which has been proven a good tool for system identification earlier in practical applications. The mechanism excites the system in a controlled manner by affecting actuators, such as valves and pumps, and the resulting process responses are logged for analytical purposes. Among the response properties to be studied are:

- *Pv-Sp* 180° phase lag frequency, $\omega_{-180°}$
- Loop transfer function, h_0, 0dB gain frequency, ω_c
- Gain margin, ΔK
- Phase margin, φ_m

These properties are considered indicators of the current process state. As the process changes, these indicators will change as well. Appropriate margins are commonly specified for a control system, and are therefore good indicators that also indicate the current stability limits of the control system. More indicators may be proposed and added to the routine later. The important principle is that the mechanism is able to recognize the current state of the system, thus identify whenever the process equipment are under influence of fault modes; such as stiction, hysteresis, dead time, bad transmitter filter settings and more, which is often caused by physical deterioration or faulty calibration.

Before any industrial implementation, the technology will be thoroughly tested and prepared in simulation environments, and the system of interest will be analyzed while at a state of satisfactory performance criteria, for later reference. Under operational conditions, the real process, as mentioned earlier, changes (i.e. drifts over time), altering the performance. During simulations common operational faults will be added to the process model and controlled by the simulation sequencer. The test mechanism iteratively excites the simulated process plant with a wide range of simulated fault parameters so that indications of changes can be found.

An extra test mechanism will excite the physical process during a given window of time, but not to the extent that it will be intolerably interfered. The test window will be selected so that the testing undergoes when the process plant is stable and there is no risk. Ideally the testing would be unnoticeable, but some oscillations will be induced in the control loop. The size of the oscillations can be easily controlled, but should be of such magnitude that the logged responses are accurate. Upon implementation at the physical plant, the monitoring process will have data acquired during the simulation study, providing an understanding of changes and consequences affecting the monitored process, and will be able to analyze data and make diagnostics of the process accordingly. As mentioned the test mechanism will be limited so that is does not analyze or affect the process at times of abnormal activity or risk. In addition to the simulation study and the practical test mechanism, a part of the solution will be a descriptor based human interface, presenting the relevant information in an understandable and descriptive manner. The interface will differentiate between different degrees of symptoms, and recommend counter measures accordingly.

This thesis will serve as a continuation of the preliminary study on the following topics, and will give an idea of the feasibility and profitability of the research, design and implementation of such a system. Since the design objective of this study is to have a kind of a general solution for use in an industrial setting, there are some constraints limiting its form according to the TDP [1]:

- The solution shall be based on a modified relay test. The relay approach has shown practical reliability and is well proven which opens up for more comprehensive work related to it.
- The solution must produce results with sufficiently low uncertainty that relate to the operational- and technical state, as well as be robust to process disturbances and noise.
- The solution should not negatively impact HSE (Health, Safety and Environment).
- The form of the solution should be general and adaptable, as well as documented, enough that it can be transferred to other similar processes without too much associated complex work. Solution algorithms should be fairly non-complex and structurally interchangeable.

Since the study is fairly comprehensive some simplifications will also have to be taken in order to ensure the quality of the work. Such simplifications are:

- The inlet separator tank of the Statfjord C production train has been selected as the process image of focus for the preliminary study.
- Study directed toward valves, which are important components of our process image. A valve should be a sufficient starting point and be able to demonstrate many of the symptoms that we want to recognize and classify.
- There will be chosen some “boundaries” for the process image. “Boundaries” refer to environments in such a steady state they can be assumed static. I.e. there are no dynamics associated to the “boundaries” and they will help limit the scope of the simulations.
- Process models will be implemented in Dymola, which is a Modelica based simulation and analysis tool. Its object oriented form will increase the reusability of the results, among other benefits which will be mentioned later.
- A simulation environment in Matlab will interface with the Dymola model. The author is familiar with Matlab through extensive academic use, which will increase the productivity on this part.

At completion a number of deliverables are expected, both as a whole or partly solution to the problems undertaken, but also as requirements formally requested by the contractor, Statoil. This thesis will contain the relevant information and documentation needed to understand the methods, grant insight in the solution process. Deliverables are:

- Dymola model for “Statfjord C- inlet separator” and sufficient surroundings[2].
- Governing program sequence for a “Monte Carlo”[3] based simulation setup written in Matlab.
- Evaluation mechanism of test results with associated descriptors. This demands some sort of HMI.
- Documentation of setup, tests and results reflecting the potential feasibility and profitability of the technology.

All delivered program code after the study shall be modular and generic enough for other engineers to be able to modify and update in a later time. At the completion of the thesis there will be an official handover process to assure that achieved project progression is maintained.

[2] In order to be able to determine the state of the system, a sufficient model that is able to reveal the symptoms produced by the dynamics of faulty or improperly calibrated equipment and sensors must be created. A considerable part of this thesis will be spent on producing such a sufficient preliminary model.

[3] In this setting Monte Carlo simulation is referred to as a broad simulation where a wide range of parameters are adjusted during simulations. The results are stored and provide a map of how different parameters affect the process.

Chapter 2
Theory

The theory chapter will detail some on the proposed test site for the technology development, and the main principles used for the operational and technical process plant state detection. Insight in the development of faults will be provided, and what these may consist of. Refer to the mentioned bachelor’s thesis [2] if some topics call for more attention, as things may be intentionally left out to prevent extensive overlapping of the collaborative work toward a solution. Such a topic is for example the details of the modified relay method, and the explanations of the equations used for estimating ω_{-180° , ω_c, gain- and phase margins in **3.3**. Topics that are considered of high relevance and importance are further detailed here by the author.

2.1 The production train

The task of the production train [4] of SFC (Statfjord C) is to process the well stream so that it reaches the desired specifications. The well stream contains a complicated mixture of hydrocarbons under high pressure. Along with the hydrocarbons, some pollution such as water, sand and other solid substances is also brought along. These components need to be separated, and also since they are naturally under high pressure they will need to be stabilized for storage and transportation.

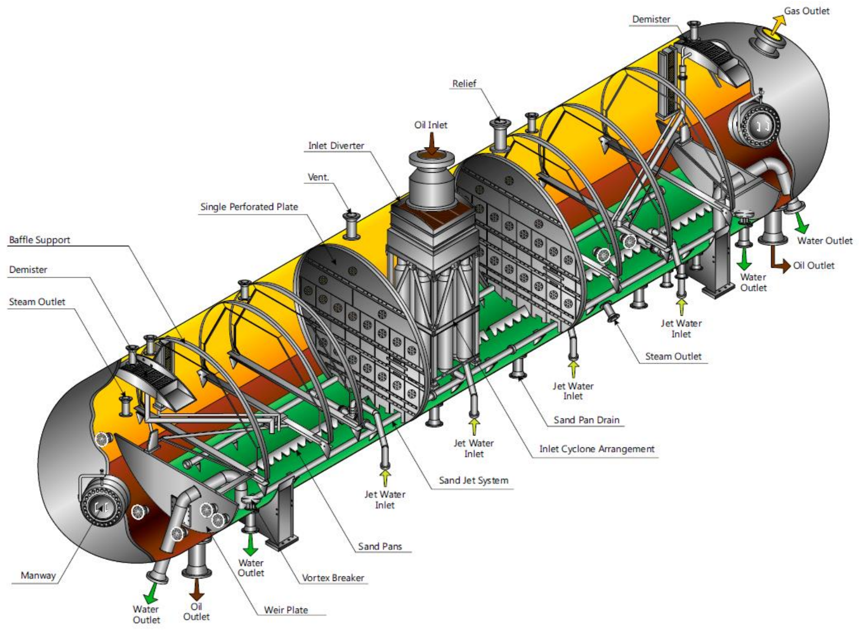


Figure 2-1: Inlet separator (CD2001) illustration [4].

The raw well stream is fed into the inlet separator tank, CD2001, which is the first stage in the production train. Oil, water and gas are separated to a satisfactory extent for this stage, and also the accompanied removal of solids. As hinted, the components of the wells stream (although they would ideally) are not perfectly separated into each category, and pollution of the other substances still occurs in the separator outlets. Most of the produced water is removed by the inlet separator, but both separator 1 (inlet) and 2 are three- phase separators, separating oil, gas and water by exploiting the differences in weight. Water, which is the heavier liquids of the three main components of the well stream, will sink to the bottom of the separator. Oil will separate from the water and float to the top of the liquids. Physical plates are installed in the separator to separate the oil from the water. The plates are called weirs. The oil level is kept above the weir plates so that sufficiently separated oil enters on the other side. The side of the oil is considered the light side of the weir and the inlet side, where the water is kept and drained, is considered the heavy side.

A multiple of other equipment are connected to the output pipes of the inlet separator tank. Equipment considered to impose main influence in the near connectivity on the inlet system are:

- ***Hydrocyclones*** are connected to the separator's water outlet pipe with purpose of further excrete oil from the water.
- Water from the hydrocyclones continues to the ***degasser*** before it is returned as sea water.
- Gas flowing through the gas outlet of the inlet separator enters ***heat- exchangers***. The gas temperature is lowered from 88°C to 35°C.
- ***Scrubbers*** further process gas leaving the gas outlets of the separator. The input gas is dehydrated; Water particles and the heavier hydrocarbons are condensed and collected on separate outlets. Condensed hydrocarbons are returned to an appropriate stage of the production train.
- The oil outlet leads to the next separator tank (***flashdrum 1***), CD2002, for further flashing[4].

These are considered to have some indirect effect on the control of the inlet separator. Loose or bad regulatory behavior in these elements can ripple to the inlet separator. The temperature of

[4] The liquids entering the separator are partially "flashed" into a vapor and liquid due to the different operational conditions of each separator stage; mainly lowered pressure.

the separator is 88°C, while the pressure is controlled with a set point of 19 BarG. In addition the separating line between oil and water also needs to be actively controlled to uphold the required settling time for the fluids for optimal separation.

2.2 Performance of control loops

In order to make decisions concerning the performance of our control loops, we need to have sets of quantitative indicators, and an understanding of the role of each control loop. This means that the roles must be bounded and quantified. The same principles as utilized while developing the control loops can be used to investigate the performance. Note that there is a degree of subjective freedom involved since the control loops can be tuned in ways that are irrational to explain with control theory, but can have practical gain in a specific industrial setting. Such freedoms involve the possibility of letting the controller control the process loosely in order to reduce the amount of oscillations further down the production train.

Each control loop gets their requirements from the controlled process. Such requirements include that of both static and dynamical properties. Such specifications can include to a selection of the following control properties [5], where some of them can be related to some degree:

- Time response
- Process output- reference overshoot constraint
- Controlled variable's follow reference property
- Margins for change of process gain with respect to stability
- Phase lag threshold at specific frequencies with respect to stability
- Noise compensation and dampening of oscillations property
- Bandwidth

Usually the importance of each of these properties is different for each control loop, and the strictness of each property may be weighted differently. Note that there is always a tradeoff in engineering; if you for example desire very fast response to change, you can expect more overshoot as well. Some properties will have more situational criticality than others; if the flow into the inlet separator (Figure 2-1) has a rapidly varying composition, e.g. a typical slug flow, the fluid levels in the separator will oscillate to such a degree that it causes poor separation, which can be seen as more oil in water exiting the water outlet, and more water in the oil travelling the oil outlet to the next separator. Oscillations can also ripple to other parts of the train, also affecting performance elsewhere.

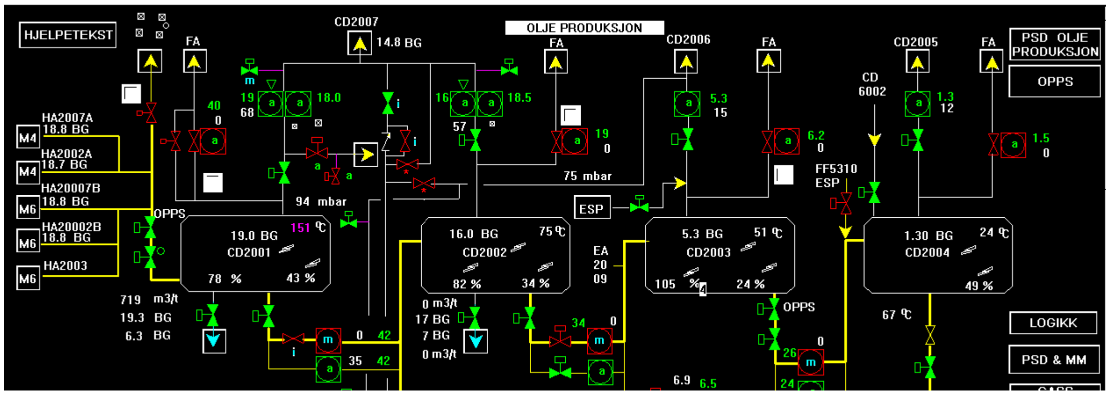


Figure 2-2: Production train overview from the Asset simulator, which is used to simulate real conditions for mimicking the Statfjord fields. CD2001 is the inlet separator (to the left), which is a three phase separator separating the crude production flow to oil, gas and water.

The control performance can be evaluated using different methods of analysis. Important features are the control loop's ability to follow the reference signal, and to compensate for process changes caused by disturbances. The control loop's tendencies toward these features can be investigated by analyzing the frequency responses, which can be expressed as how sine- and cosine- signals on the input are manipulated throughout the system. Frequency components can be phase- shifted, biased and amplified/dampened, independently. Finding Bode plots can be done in many ways, most of which are applicable in different circumstances. Examples are:

- If we have accurate transfer- functions[5] for the process in an operating point, we can easily express the frequency responses in a bode plot using direct mathematical analysis.
- If the transfer functions are unknown (e.g. in a case where the process has so many varying parameters that the original differential equations no longer accurately reflect the process) we can attempt to fit a model to the system response, using system identification theory or by trial and error (qualified guessing).

[5] Transfer functions can only be applied if the process is or can be considered linear around an operating point. When process parameters are shifted, the process transfer functions may need to be corrected.

- A different method is applying signals with known amplitudes and frequencies on the input (e.g. sine sweep), read the resulting output values and plot the relations in a bode plot. The frequency responses can be used to express control systems' properties with respect to stability and performance. The latter will be conducted in the simulation study in order to produce a footprint of the whole process, which will be compared with results from relay feedback testing.

2.2.1 Phase- and gain margins

The phase- and gain margins express how much change a system's frequency response can change before an asymptotic stable system becomes marginally stable [5].

- Asymptotic stable systems are characterized as having all poles in the left half plane of the unit cycle for the continuous plane (s-plane).
- Marginally stable systems have one or more poles on the imaginary axis.
- Unstable systems have one or more poles in the right half plane.

During operational conditions the poles and zeros of the process' transfer function[6] wander, and although we might have an asymptotic stable system at commission it can become less and less dampened until it reaches a marginally stable system. The effect of each stability property is:

- Asymptotic stable systems: The stationary impulse response is 0.
- Marginally stable systems: The stationary impulse response is different from 0, but limited.
- Unstable systems: The stationary impulse response is unlimited.

We can in other words not keep an unstable system under control and if the controller and process is left completely alone, over time it will likely "collapse" on itself. We need to maintain an asymptotic stable system, which is why we express requirements for the phase- and gain margins.

The amplitude crossover frequency, ω_c,is the frequency where the loop transfer function, h_0, gain is equal to 1. This means that for an open loop system, the amplitude output-input relation is 1:1.

$$h_0 = R(s)P(s)M(s)^7 \qquad (1)$$

[6] The transfer function is complex and unknown, but still existent.

$$|h_0(j\omega_c)| = 1 \qquad (2)$$

The phase crossover frequency, ω_{180}, is the frequency that causes the loop transfer function to have a phase lag of $-180°$.

$$\angle h_0(j\omega_{180}) = -180° \qquad (3)$$

The gain margin, ΔK, is the multiplicative increase that h_0 can take at ω_{180} before the loop transfer function passes the critical point[8] and becomes marginally stable.

$$\Delta K = \frac{1}{|h_0(j\omega_{180})|} \qquad (4)$$

In practice this can be interpreted analogous to a scenario where a level transmitter outputting 4-20mA for levels of 0-4m is replaced with a level transmitter with the same output range but for levels between 0-2m. This change effectively doubles the gain of h_0 and in order to still have an asymptotic stable system according to the gain margin, ΔK needs to be > 2. Normally it is the physical elements of the process itself that changes $|P(s)|$ and thus shifts the poles of the transfer function closer or farther away from the critical point. This can be exemplified as a valve developing stiction[9], causing more aggressive behavior to changes.

The phase margin is defined as the amount of added phase lag h_0 can tolerate before reaching the critical point.

$$\varphi_m = 180° + \angle h_0(j\omega_c) \quad (5)$$

These observations lead us to the Bode-Nyquist stability critera:

$$\Delta K > 0dB \; or \; \varphi > 0° \qquad (6)$$

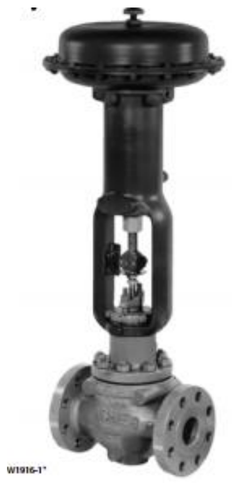
If any of these conditions are satisfied we are sure that our system is asymptotic stable, or in other words; controllable. Phase- and gain margins requirements should be specified according to the worst theoretical scenario. Note that controllable does not mean satisfactory controllable and that these measures do not reveal anything of our other control performance properties, such as the *Pv-reference* follow property.

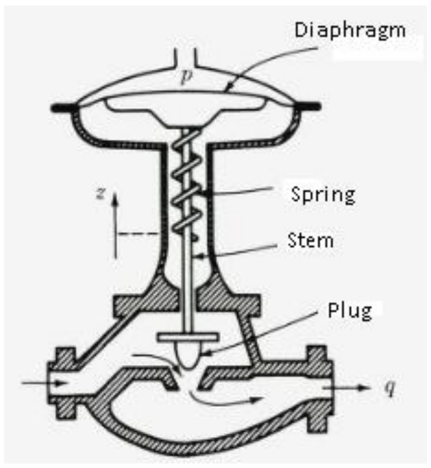
[7] R(s), P(s), M(s) are the control, process and measuring transfer functions consecutively.
[8] Where the poles intersect the imaginary axis and the process becomes marginally stable.
[9] Stiction is defined in 2.3.

2.3 Valve properties

Control valves are the most common actuators in control loops. They are mechanical and are used to limit, or restrict, the amount of flow of process medium through a pipe. This is used to control process parameters such as temperature, pressure and fluid levels. The Fisher 667-70 sliding stem valve (Figure 2-3) is one of the valves connected to the inlet separator, controlling the oil liquid flow exiting the separator. As descripted in [6] a valve consists of three basic components:

- Actuator; The positioner for the valve plug
- Valve body subassembly; Valve casing, valve seats and valve plug
- Accessories; Position sensors, I/P transducers etc.





- **Figure 2-3: Leftmost: Fisher 667 sliding stem control valve [7]. The 667 valves are reverse acting, which means that pressure of applied on the bottom of the diaphragm creating a force opposing the spring force. This setup gives a fail closed position. The rightmost figure is taken from Finn Haugen's book "*Regulering av dynamiske systemer, 1994*" [5], and describes the same working principle and general internal structure as used in the Fisher 667.**

Since a valve is a mechanical component, they develop faults over time due to wear and tear. Consider as example the extreme case of a valve controlling a well stream, consisting of a mixture of different substances such as crude petroleum, chemicals, water and sand. Coarse working conditions can cause corrosion, which over time alter the characteristics of the valve, and may eventually result in leakage; either internal (can result in fluid passing through a closed valve) or external (such as stem leakage). In addition, the valve depends on mechanical movement in order to perform its task, and friction on the actuator- stem can increase, resulting in a slower working speed. Non optimum clean fluids can contribute to deposits on stem.

The ideal valve as described by Choudhury [6]:

An ideal valve should have a constant gain throughout the valve travel span, i.e. a linear installed flow characteristic, no dead time with properly adjusted packing and a small time constant.

Choudhury further mentions details of problematic tied to practical valves. They can be summarized shortly as follows:

- Incorrectly sized valves or incorrect flow characteristic valve for intended operation.
- Corrosion on valve seat, casing and plug.
- Actuator faults, such as faulty diaphragm.
- Partly or fully blocked air vents.
- Packing leakage or too tight packing on stem.
- Crystallization and scale on stem, plug and seat.

Any of these problems, or a combination of them, can results in an associated development of:

- Static friction, “stiction” (slip jump and dead band behavior).
- Saturation of valve travel range.
- Backlash (slack or reverse motion).
- Increased response time due to change of friction on stem or by weakened actuator (valve travel time).
- Change of flow characteristics (wear and tear).
- Change of gain, Kv [8].

Many of these problems will add additional linear and nonlinear properties to the process, and if their combined effect gets too big, the control loop will no longer be able to perform its task at a satisfactory level, and in the worst case cause a shutdown. It is desired to proactively search to eliminate the problems before they become too dominant, either by running maintenance on, or by replacing the affected valves. Valve service plans are usually created based on experience or some form of statistical expectations. The manufacturer provides the expected mean time between failure (MTBF [9]) according to the equipment’s specifications and operating area. The MTBF is predicted to fulfil certain reliability. In order to achieve the reliability factor, the manufacturer will advise to run maintenance more often than the MTBF. Statoil mostly perform reliability centered maintenance (RCM, [10]), which is an experienced based approach to scheduling and performing maintenance to uphold the process integrity.

Depending on the working conditions, and the natural randomness of things, valve problems are allowed to develop for some time. Unsatisfactory and faulty valves are usually depending on being manually detected. If problems are comprehensive and in e.g. a critical section of the crude oil separation, this can result in downtime or a reduced production rate. It is noteworthy that even small deviations from the ideal valve characteristics will affect the behavior of the control loop, and reduced performance may propagate in some degree to other parts of the process plant.

2.3.1 Stiction

Static friction, also termed "stiction", is the most common problem in spring-diaphragm-type valves. In the lack of a formal definition of stiction, Choudhury investigated earlier self-proclaimed and adapted definitions of stiction for common properties. Choudhury then proposed a formal definition, able to define the phenomenon of stiction in valves, as follows:

The presence of stiction impairs proper valve movement, i.e. the valve stem may not move in response to the output signal from the controller or the valve positioner. The smooth movement of the valve in response to a varying input from the controller or the valve positioner is preceded by a stickband and an abrupt jump termed as slip-jump. Its origin in a mechanical system is static friction, which exceeds the dynamic friction during smooth movement of the valve.

Stiction is a byproduct of a valve-packing-tradeoff. Too tight packing around the stem prevents leakage, but also increases the friction on the stem movement. Corrosion and flaws can also add to the unevenness of the valve stem, and increase the friction. See Figure 3-23 for typical effects caused by stiction.

2.3.2 Valve saturation and diaphragm faults

Mohamed A. Sharif and Roger I. Grosvenor [11] did experimental tests to highlight limitations by today's diagnostics tools on control valve. Their experiments illustrate how common control valve problems affect the valve properties, and pose degradation and limitations to the valves' objective. Among the discussed problems are:

- Entrapment of air in the upper diaphragm casing as result of partly of fully blockage of air vent, which works as an air spring opposing the pressure force applied to the other side of the diaphragm. Additionally the blockage will likely reduce the valve's "backward" travel capabilities, so that the valve's travel time in each direction is different.
- Diaphragm rupture, which results in a loss of applied pressure to the valve stem, reducing the sensitivity to pressure changes. Even though there is a throttling element which provides pressure its maximum allowed pressure output is set.
- Internal and external leakage. This is normally due to bad stem packing, or corrosion at the valve seat or plug. Can also come from an unbalanced valve plug.

Any of the mentioned faults can cause saturation of the valve's operation and either reduce the possible amount of fluid flow through the valve, or inhibit the valve from preventing liquid flow.

2.3.3 Valve leakage

Valve leakage can be closely related to seals and gaskets. According to an article by Sanders, D. [12], this is one of the two leading causes of user concern regarding control valve performance, along with oversized valves. Tight seals are vital to ensure product quality, but also in respect to safety and the environment.

- Too loose gaskets on the valve stem will make valve travel rapid and fluent, but will also contribute to allowing fluids to exit the valve along the stem. This again can impair the valve's movement and cause an increase of wear and tear on places that are not designed to be in direct contact with the process fluids. If the process fluids (or other substances like gas) exit the pipes in unintended places this can cause hazardous situations, or make an increase of potential risk.
- The valve plug-to-seat interface is the largest contributor when it comes to seat leakage, which is an internal leakage problem. Internal leakage will cause a form of saturation as mentioned in the previous section.

- In some scenarios the process medium can corrode new passages for it to flow. The process medium can be turbulent and have a corrosive composition. It is important that the valve material choice is appropriate for its task. If this is a problem and the valve is left too long without maintenance, both internal and external leakages can develop.

2.4 Measuring equipment properties

Among measuring equipment, level transmitters will be of focus, but most of the theory is applicable to other types of transmitters as well. The level transmitters at SFC are Rosemount 3051 (see Figure 2-4) differential pressure transmitters, measuring the differential pressure between their two connectors. Common errors and mistakes are:

- The transmitters work as transducers meaning that the physical displacement of a thin diaphragm separating the two connected inputs to electrical energy of 4-20mA read by the process control and data acquisition system. Transmitters need to be scaled properly both in the field and in the control system, which can be a potential cause of erroneous readings. For example a transmitter is set to provide its full range of electrical output for a level reading between 0 to 3 meters. This means that it will reach saturation if the liquid levels go beyond these limits.

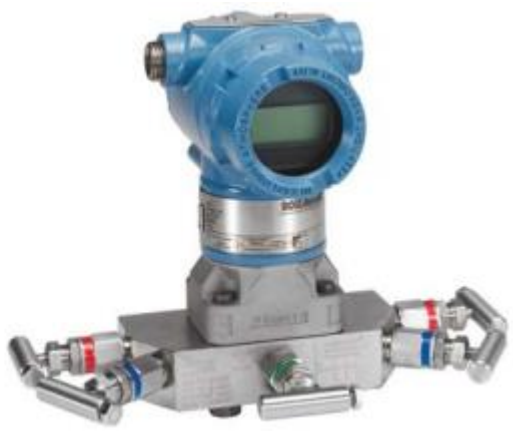


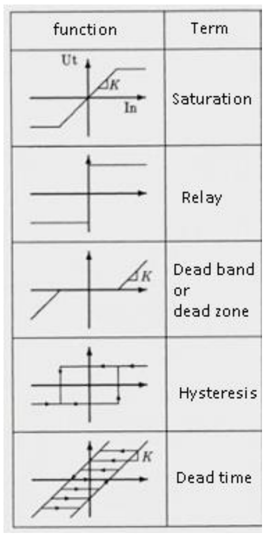
Figure 2-4: Rosemount 3051 pressure transmitter [13].

- It is important that the bandwidth of the transmitter is not exceeded, as the frequency response of a transmitter is normally designed to be approximately linear around its intended operating area, while below or above it may have nonlinear characteristics.

- The physical connectors can be partially or fully blocked, preventing the transmitter from making the differential pressure readings correctly. The diaphragm can also get worn out or in some way become stuck. This can cause signal freezes or total malfunction.
- The transmitters' readings are based on the differential pressure calculations of liquid height. To get the calculations correctly the composition of the liquids on both sides must be known. This is not always trivial, although constant measurements on the well stream composition are done. Incorrect specific weight values errors on readings, such as added (and unknown) skew.
- Small dents and unevenness will contribute to altering the readings from the transmitter. This is vulnerability since it measures the differential pressure of two potentially corrosive substances on either side. The substances can also potentially scratch and deposit sediments on the diaphragm. As result; bias or "*zero- drift*" can be added to readings.
- The transmitters come with an option called "*damping*", which in reality is a low- pass filter option. In practice the setting is used to prevent rapid fluctuations in readings caused by noise or disturbances that are not of interest. In many cases a common mistake is to set the damping factor too high, so that the transmitters become blunt and too slow at responding to changes so their output no longer adequately represent the physical levels.

2.5 Generic nonlinear effects

Finn Haugen points out in his book "*Anvendt reguleringsteknikk,1990*" [14] (English: Applied control theory) that in reality there are elements that add nonlinear dynamics to a given process. These elements can for example be associated with valve travel dynamics. Such elements are contained by the process but may be hidden and less obvious to the control engineers. Their roots can be related to the physicality of the process and the process equipment, and their magnitude may change during operation. Their effects will only be apparent in the process response, but it is important to have an understanding of them nevertheless.


Ut
ΔK
In
ΔK
ΔK

function	Term
	Saturation
	Relay
	Dead band or dead zone
	Hysteresis
	Dead time

Figure 2-5: Figure is an excerpt of an illustration adopted from Finn Haugen's "*Anvendt reguleringsteknikk, 1990*" [14], with some translations.

The illustrated functions are considered to contribute with generic nonlinear effects to the controlled process, and their effects are shown in Chapter 4.2. The meaning of the term *generic* in this context is that the functions (and their associated features) are considered underlying causes of most of the prominent and prevalent nonlinear features observed at the process output. Each function is later simulated in series with the process, individually, and also in combinations. We will also see how the suggested indicators (according to the TDP [1]) change with the different configurations.

2.6 Test mechanic hypothesis

The theory the research is built on is derived from practical experience and conviction that a relatively simple approach to controller tuning has more potential than what has been exploited previously. The suggested method is based on a modified relay method where a relay is put in series with/or replaces the PID controller. For robustness the relay is placed in series for this purpose.

Åström and Hägglund [15] proposed the standard relay test in a paper of 1984; replace the regulator with a relay in series with the process to automatically detect the critical gain and critical frequency, with the objective of automating tuning of simple controllers as well as initialization of more complicated adaptive controllers. The method has been widely used and modified during the years.

Implications of the theory used as base for this thesis is an extended use of the modified relay method to not only use it for support and ease controller tuning by knowing the amplitude and phase margins, but to reveal process control loop and control equipment characteristics, thus possibly indicate the source of changes in control performance (Figure 2-6).

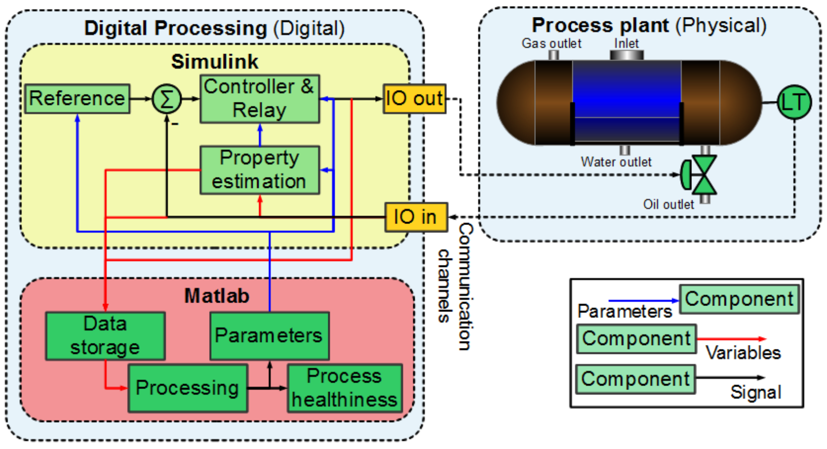


Figure 2-6: Descriptive drawing of the test mechanism implemented on the physical plant.

The method is a variation of a modified relay test and is directed toward building an understanding of how different faults affect the frequency response of the process in series with the known controller. The operators know when the process is performing satisfactory and the process state at that point can be used as a reference. By studying how the selected indicators change with different parameter settings of modelled faults during the simulation study, the knowledge base shall give insight and enable setting limits to determine whenever the performance is lowered, and what likely causes are. The indicators will indicate the process healthiness.

2.7 Standard and modified relay feedback method

The standard relay feedback method can be seen in Figure 2-8. Once the testing is initiated the PID controller is disconnected from the control loop. The relay (with adjustable hysteresis) has a built in memory and outputs its value according to the rules:

- High signal,$+M$, if $e < -\varepsilon$
- Low signal,$-M$, if $\varepsilon < e$
- Else $y = u$

Where ε is the modifiable hysteresis on the input of the relay that is controlled by the test mechanism and e is the $Sp - Pv$ deviation. An expression that is central in the derived equations for the relay tests and the test mechanism is $\frac{4M}{\pi}$, which corresponds to the amplitude of the first harmonic frequency of the relay, where $\pm M$ is the magnitude of the relay output.

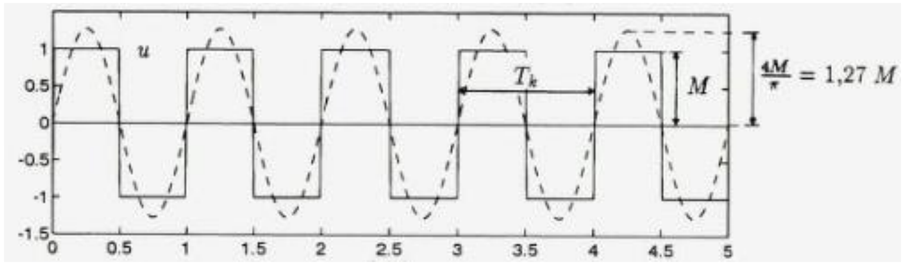


Figure 2-7: Illustration of the relay output and its corresponding first harmonic frequency. Illustration is adopted from a textbook written by Finn Haugen [5].

The relay method gives the open loop response. The feedback is "cancelled" by the relay, and the relay induces oscillatory behavior in the process output, called *limit cycles*. The frequency of the output is dependent on the chosen hysteresis, ε. If $\varepsilon = 0$ the produced limit cycles have frequency of $\omega_{-180°}$which corresponds to the frequency of the critical point. $\omega_{-180°}$ means, as mentioned earlier, the frequency of where the open loop response has a phase lag of $-180°$ from the reference. On the unit cycle this equals -1, and with the negative feedback this corresponds to -(-) which gives +, thus a positive feedback and instability unless we have an appropriate amount of gain dampening. ω_c can be found by adjusting the hysteresis, and then the phase and gain margin can be easily calculated as well. The method used for this is found in [16], and is also summarized in 3.3.8.

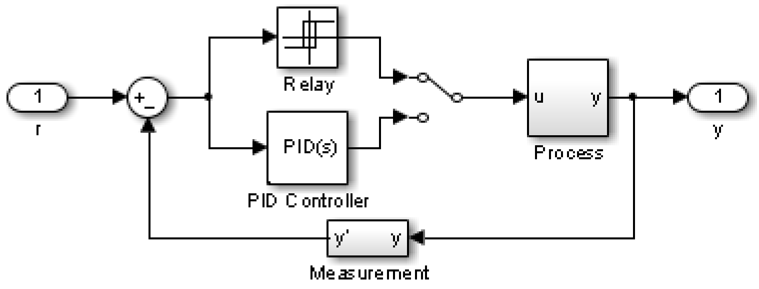


Figure 2-8: The standard relay feedback method. The relay and controller are run in parallel.

Running the test mechanism as a standard relay method was desired since the controller's transfer function is not included in the transfer function for the revealed indicators. The transfer function seen by the relay method is $P(s)$ for the standard relay test. $R(s)$ is the transfer function for the controller, $P(s)$ is the process transfer function and $M(s)$ is the measurement transfer function. There is one undesirable effect in particular that points toward a modified relay structure instead.

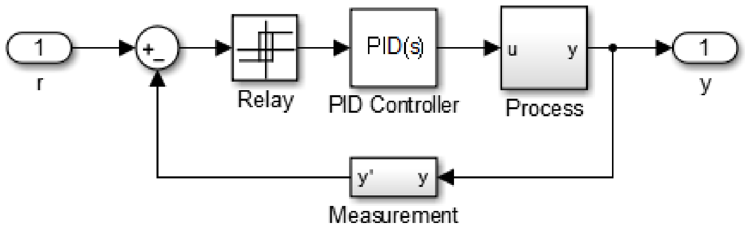


Figure 2-9: The modified relay feedback method. The relay is now in series with the controller.

For the modified relay test the observed[10] transfer function is $G(s) = R(s)P(s)M(s)$. Although $P(s)$ and $M(s)$ are unknown, the form of $R(s)$ is known exact since we design it by setting its parameters. If we measure $G(s)$ we can therefore derive the transfer function contributions

[10] Observed as in the meaning that we see how the known input is changed throughout the system.

originating from$P(s)M(s)$ in series, and if we could know $M(s)$ exact we could isolate the contribution from $P(s)$, but in reality these two are inseparable. The transfer function for a PID controller is given by the equation (this is the standard and generally known equation):

$$R(s) = K_p \frac{T_i T_d s^2 + T_i s + 1}{T i s} \quad (7)$$

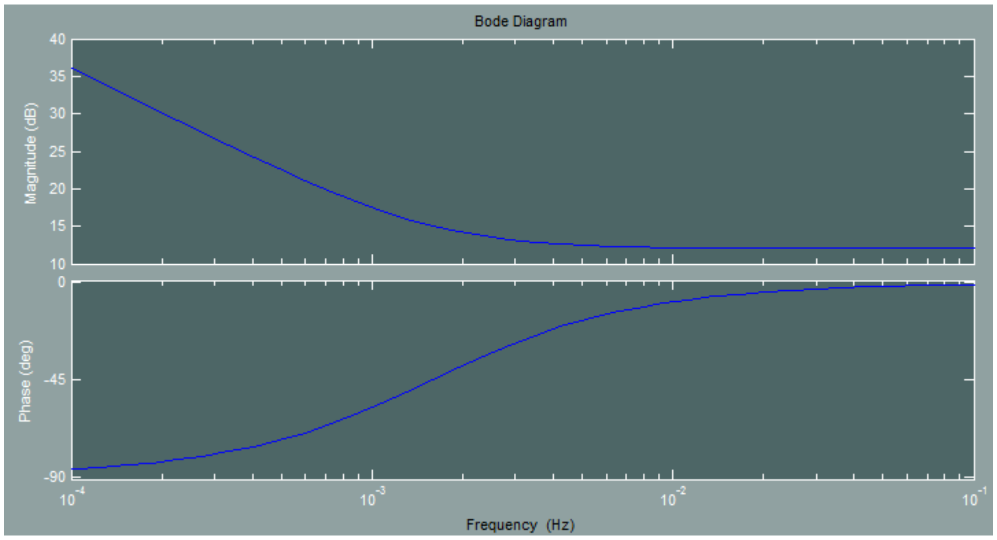


Figure 2-10: Frequency response of a PI(D) controller with Kp=4, Ti=100 and Td=0. This is the controller setup used in the Monte Carlo simulation in Chapter 4.

The response seen from Figure 2-10 is an example of the added controller contribution to the loop transfer function. The advantage of keeping the controller in the loop for the relay feedback testing is to keep the process around the operating point, which is important for different factors. If substantial amount of disturbances and forces act on the process output, they can break the relay mechanism by preventing its input value from reaching the required value for the relay to switch. $\omega_{-180°}$ for $G(s)$ will be different than that of $P(s)$ and $M(s)$ in series, but since we are interested in the control performance this is a topic of no concern. We are interested in process parameters, as well as control performance.

2.8 Mapping process "footprint"

As mentioned in **2.7**, zero hysteresis makes the relay produce oscillations with frequency corresponding to $\omega_{-180°}$, and corresponding gain can be read accordingly. ω_c is also found according to the mentioned method, and together these two frequencies mark the edge points of interesting frequencies to study. Finding "footprints" for our process model is vital to provide an understanding of how our process state indicators travel as the process is under operation. When we know how the indicators travel, we can later look on the indicators in order to see how the operational and technical states have changed, and we can make plans for controller retuning and process equipment maintenance accordingly. In addition to being reliable tools for obtaining the frequency responses for the "footprints", the proposed methods can also be used to obtain additional indicators for the test mechanism; e.g. obtain gain dampening per decade or phase sensitivity at specific frequencies.

Several methods for mapping the process footprint was developed, but most were considered too unreliable that they could be used. We need something that is little subjected to random variations induced by the simulation time steps and discretization. The methods are not required in practice as they would never be run on a physical plant due to their effect on the control loops. Two approaches were proven to be outstanding and are included in sub sections **2.8.1** and **2.8.2**. The first approach is a post simulation procedure and finding the phase and gain frequency responses, while the other is benefiting from simple components to do it in real time.

2.8.1 Approach 1: Sine sweep and fast Fourier transform

To find the exact bode plots for the simulated process we use a sine sweep approach by applying sine waves of known amplitude, frequency,ω , and phase,θ, on the input to $G(s)$, and read the resulting amplitude and phase response on the output signal. The difference between the input-output amplitude and phase then corresponds to $G(jw)$. The observed system, as opposed to in the modified relay test, is the closed loop of $G(s)$. This means that $G(s)$ is part of a normal control feedback structure described by:

$$h(s) = \frac{y(s)}{r(s)} = \frac{G(s)}{1+G(s)} = \frac{R(s)P(s)M(s)}{1+R(s)P(s)M(s)} \qquad (8)$$

The contribution of $P(s)M(s)$ can be found by solving equation (8) by reorganizing for $P(s)M(s)$, but the feedback structure applies a resonance peak on the amplitude response and which cannot be removed by reorganizing the equation when calculating "backwards"[11]. Instead using the fast Fourier transform (*FFT*) is suggested and it has proven itself reliable and accurate in most situations[12]. This transform brings the time variant signals to the frequency plane, where we can easily isolate the frequency of the limit cycles and obtain phase and gain change from *Op* to *Pv*.

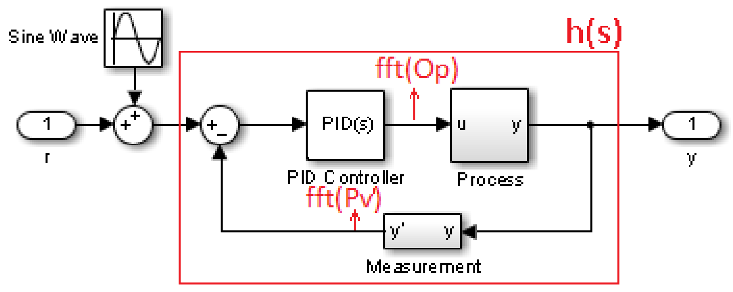


Figure 2-11: Applying sine waves on the reference while the process is quiescent at the operating point.

The transfer function of $P(s)M(s)$ is unknown, and we have to estimate its frequency response by studying how its output behaves with different frequencies on its input side, which is connected to the controller output. Sine waves ranging from ω_c to $\omega_{-180°}$ are applied as time varying set point for the control loop and the frequency responses are easily obtained by logging the controller output signal , *Op*, and the controlled process value, *Pv*. Thereby the response is found by applying the *FFT* on the data to obtain $P(s)M(s)$' gain and phase lag [17] during the post simulation procedure.

2.8.2 Approach 2: Relay sweep adjusting hysteresis and transport delay

This approach is more practical and exploits how the modified relay forces limit cycles and how these cycles can be modified by adding nonlinear elements on the digital processing side to pro-

[11] This was attempted, but the resonance peak completely overshadowed some of the interesting frequency response, as well as manipulating some of the values around.

[12] FFT does not cope as good in situations with high signal deterioration, such as when frequencies are close the sample frequency or if there are nonlinear elements in the process/equipment so that some energy from the first harmonic frequency of the input is shifted to other frequencies (and therefore no longer noticed by the FFT gain/phase algorithm).

duce additional information. The method has further shown that the sine sweep approach is obsolete, since *FFT*s can be obtained for the digital signals produced in the relay setup as well, thus removing the need for a sine wave source, thus further generalizing the method. The only drawback is that the relay produces a range of frequencies rather than just one, but we can dismiss all but the first harmonic in the resulting *FFT*s.

Adjusting hysteresis relay sweep:

As mentioned in **2.7** we can lower the frequency of the limit cycles by applying hysteresis to the relay. This has been proved to provide reliable results for the gain response of $G(s)$, and thus for $|P(s)M(s)|$.

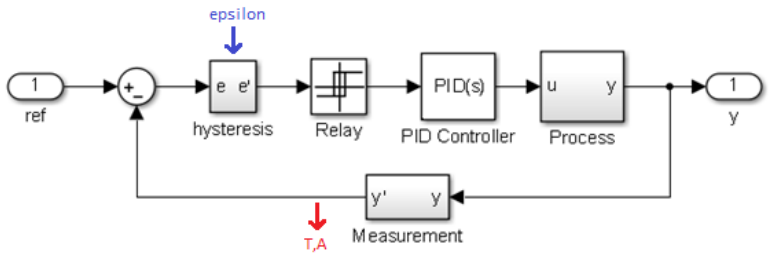


Figure 2-12: Illustration of the adjusted hysteresis relay sweep setup (in Simulink the hysteresis is in reality part of the relay block, hence its block drawing).

The relay hysteresis, ε, is adjusted between the value that produced ω_c, and 0 (which produced $\omega_{-180°}$).

$$\varepsilon = \langle\, \varepsilon_{w_c}, 0\, \rangle$$ [13]

The measured process output is passed through a period- and an amplitude estimator (detailed in 3.3.6 and 3.3.7). Consequently,

$$|G(j\omega)| = \left|\frac{y(j\omega)}{u(j\omega)}\right| = \frac{y(j\omega)}{\frac{4M}{\pi}} \quad (9)$$

[13] Note that $\varepsilon = 0$ and $\varepsilon = \varepsilon_{\omega_c}$ are not included in the sweep since their amplitudes are already known from the test mechanism.

Where ω_p is the estimated frequency, $\frac{1}{T_p}$, in hertz. If the estimators are correctly parameterized this provides an accurate and reliable reading for $|G(j\omega)|$. Further; $P(j\omega)M(j\omega)$ can be isolated by removing the contribution from the known $R(j\omega)$.

$$|P(j\omega)M(j\omega)| = \frac{|G(j\omega)|}{|R(j\omega)|},\ or$$

$$|P(j\omega)M(j\omega)|_{dB} = 20\,log_{10}|G(j\omega)| - 20\,log_{10}|R(j\omega)|$$

However, it is not straight forward to derive the effect that adjusting the hysteresis has on the phase of $G(j\omega)$ so a second test is proposed for finding the phase response; adding a transport delay on the relay output.

Adjusting transport delay relay sweep:

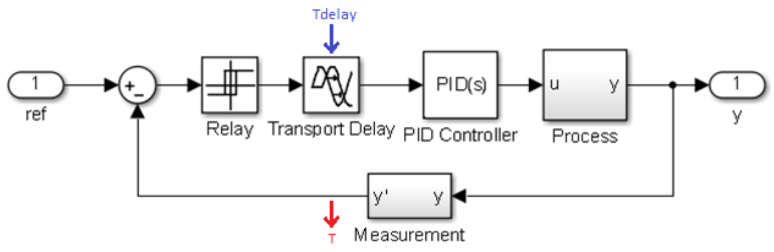


Figure 2-13: Illustration of the adjusted transport delay relay sweep setup.

The transport delay's ability is to "steal" bandwidth from the rest of the system during the modified relay test with zero hysteresis. Since the relay (with zero hysteresis) will always induce the critical limit cycles ($\theta = -180°$), the resulting frequency of the output oscillations is given by

$$\omega_p = \omega_{G\left(j\omega_{-180°-\vartheta(T_{delay},\omega_p)°}\right)}$$

Where

$$\vartheta\left(T_{delay},\omega_p\right) = -\frac{T_{delay}}{T_p} * 360° = -\omega_p * T_{delay} * 360°$$

Since ω_p is unknown until we have ran the test with a given transport delay, T_{delay}, the resulting frequency shift, $\Delta\omega_p$, can be tiny or large depending if we are positioned at a steep slope or on flat ground on the phase response of $G(j\omega)$.

The value of T_{delay} has to be trialed and adjusted incrementally since we cannot predict which frequency we will end up on in advance. T_{delay} should therefore be incremented by small steps at start, and one could consider letting ΔT_{delay} increase inversely proportional to $\Delta\omega_p$, if the resulting $\Delta\omega_p$s are small.

$$T_{delay}(n) = T_{delay}(n-1) + \frac{K_{delay}}{\Delta\omega_p}$$

Where K_{delay} is a suggested sensitivity factor of the T_{delay} adjustment between simulation number n and $n-1$. For high frequency response resolution K_{delay} should be kept small, but for simulations each increment of T_{delay} must be higher than the simulation step time. Simulating increments of T_{delay} is done until $\omega_p < \omega_c$.

$$T_{delay} = [\, {T_{delay}}_{\omega_p < w_c}, 0 \,\rangle$$ [14]

$\angle P(j\omega)$ can now be found for all produced limit cycles, ω_p, according to:

$$\angle P(j\omega_p)M(j\omega_p) = -180^\circ - \angle R(j\omega_p)^\circ - \vartheta(T_{delay}, \omega_p)^\circ$$

We can now finally represent the sampled frequency response of $P(j\omega)M(j\omega)$, which is our process "footprint".

[14] The phase of $T_{delay} = 0$ is already known from the relay test with zero hysteresis (-180°).

Chapter 3
Experimental

The experimental chapter will start off by listing some of the requirements (**3.1**) for the models produced (**3.2**). Most of them are related to the modelling power of faults mentioned in **Chapter 2**. The chapter will then progress on by illustrating and describing the produced models (**3.3**), and associated sub elements, as well as fairly in- depth descriptions of their structure and behaviors. Component testing and associated results can be found in section **3.4.1**.

3.1 Model requirements

Model in this context is a reference to all Simulink and Dymola models produced. Among such models can be the top level Simulink model, or the model of a control valve. The models used for the simulation study need to be able to reflect realistic technical and operational changes that the inlet separator of SFC is subjected to. The system can be split into three main components; transmitters, valves and physical separator, which are each modelled independently and detailed in separate subsections of this chapter.

The models produced are to be used in conjunction with both the Simulink and the Dymola process models, meaning that the Simulink control valve and transmitter models are also acting as elements in Dymola, and need the proper interfacing. Their objective is to create a realistic image of real equipment, with parametric faults, for the simulation study. They shall reflect behavior seen in real equipment, where faults have root in physical causes that can be modelled, and which can be governed by Matlab. The focus will be directed at some or most of the fault modes described in 2.3 and 2.4.

The models should ideally have realistic inputs to them, but to keep in line with the main topic of the thesis which is finding the signatures of different linear and nonlinear characteristics induced by change of technical states; this would in contrary bury some of the interesting results. This is still optional and can easily be included on a later stage for hardening the proposed test mechanism itself against disturbances. Relevant data can be obtained by logging the liquid flows through each level control valve and create a looping vector of these values and insert into the simulated environment.

3.2 Models produced according to requirements

All parameters in the Simulink model are stored in a structure called *MODEL*, which means they have the prefix *MODEL.* when they are initialized. The model separates the digital processing signal side from the physical part of the process.

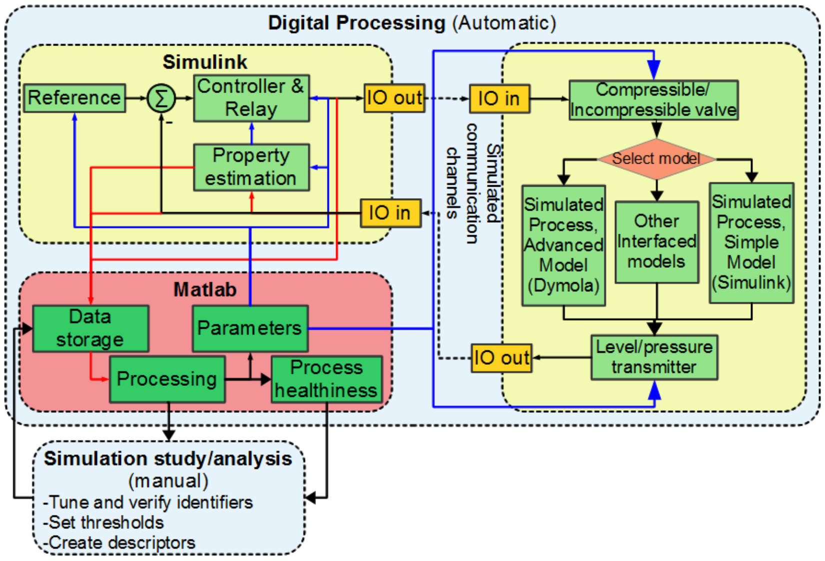

Digital Processing (Automatic)
Simulink
Reference
Σ
-
Controller & Relay
Property estimation
IO out
IO in
Simulated communication channels
IO in
Compressible/ Incompressible valve
Select model
Simulated Process, Advanced Model (Dymola)
Other Interfaced models
Simulated Process, Simple Model (Simulink)
Level/pressure transmitter
IO out
Matlab
Data storage
Parameters
Processing
Process healthiness
Simulation study/analysis (manual)
-Tune and verify identifiers
-Set thresholds
-Create descriptors

Figure 3-1: Principle drawing of the Simulink model containing the actuator, process and equipment models. Data flow with Matlab is also included.

An accurate process model has been created in Dymola and based on existing Statoil libraries. Re-use here reduced the workload by a large factor and allows more concentration to be spent on the actual topic of this thesis. The Dymola model's task is to maintain a realistic and true process image during simulations, benefiting of the powerful physical modelling capabilities of Dymola, while the control oriented approach of Simulink and the analytical capabilities of Matlab is used to dictate the simulation sequencing, and interpret the produced data.

The time constraint of this work prevented the model from being operational and implemented in Simulink. It lacks parameter setting, and interfacing with Simulink. The Dymola model was going to be exported to the Simulink environment as a *.MEX* file. The created Dymola model (Figure 3-5) shows the choice of boundaries (where the production train meets stable counter forces) and the elements included.

3.2.1 Simulink simplified process model

Only the light side of the weirs in the separator is considered in the simplified model. Light side refers to the side where the oil is drained. The separator is a hollow cylinder shape, with half spheres at each end point. Weir plates are located on each side of the separator. The crude three phase fluid from the production manifold enters the separator in the middle. Sand and other minerals and chemicals may also be included in the fluid mixture, but are not important for modelling purposes. The substances of interest are gas, oil and water. Water, which is the heavier substance, will by sink to the bottom of the separator by gravitational principle. The water level is held at a desired level by water control valve, below the weir plates so that water does not flow over the weir plates, sinks, and gets drained along with the oil continuing further down the production train. The water is not considered pure and continues through hydro cyclones and to the degasser to achieve the required maximum of oil in water before being disposed as sea water. When the three phase fluid enters the separator, lowered pressure causes the lighter hydrocarbons to flash. A combination of the separator volume occupied by the incompressible fluids, oil and water, and the amount of gas gives the total pressure exerted on the substances leaving the separator. Pressure in the separator is controlled to ensure optimal separation and uphold safety. Emergency valves are installed in case of dangerous situations. The pressure set point is maintained by draining gas through the gas control valve at the top of separator. As mentioned the oil level is kept above the height of the weirs. Both light sides of the weirs are connected so that the levels at both sides will be the same. Oil exits the inlet separator through the oil control valve.

Control loops involved in keeping optimal conditions for separation:

- Oil level control loop
- Water level control loop
- Pressure control loop (gas amount)

The simplified model was developed in Simulink for the purpose of providing the signatures[15] of (simulated) prominent process changes. Dimensions and terms are listed in Table 1 page 36. The model consists of a simplified three phase separator which only "sees" the light side of the weir (see Figure 3-2). Oil flows over the weir as a disturbance, and is drained through the oil control valve. The level of oil is kept under the weir height so that only the light side volumes are considered in volume calculations. This violates normal operational circumstances as the level of oil in the drum is usually set above the weir to prevent water from flowing over the weir and through the oil outlet. This should not have any impact on the verification of the test mechanism, since it will mostly only impact the residence time of the fluid in the drum, and the nonlinearity caused by the volume calculation at different set points.

Starting point of the oil height at light side is the mass balance equation:

$$\frac{dM(t)}{dt} = w_i(t) - w_u(t) \qquad (10)$$

Which further leads to:

$$\frac{dh(t)}{dt} = \frac{1}{A(h(t))}(q_i(t) - q_u(t)) \qquad (11)$$

$q_i(t)$ is considered as an unknown disturbance of oil flowing across the weir to the light side, while the outgoing flow, $q_u(t)$, is controlled by a level valve following the equation

$$\frac{Kv}{3600} f(x)\sqrt{\Delta P(t)} \qquad (12)$$

Note that *3600* is a conversion from $\frac{m^3}{h}\, to \frac{m^3}{s}$, and $f(x)$ is the valve flow characteristics. The differential pressure across the valve is calculated by:

$$\Delta P(t) = \left(\frac{\rho g h(t)}{100'000} + P_{CD2001}(t)\right) - P_{CD2002}(t) \qquad (13)$$

Where ρ is the density of oil and g is the gravitational constant. Division by 100 000 is a conversion from the physical unit Pascal to barG, which is standard in the SFC systems. $P_{CD2001}(t)$ and $P_{CD2002}(t)$ are internal separator pressures of the inlet separator and flash drum 1 respectively.

[15] The models are also used for validating the test mechanism during development iterations. It is important to keep in mind that the TDP is the basis, and that the equipment fault (and other process properties) signatures are of most importance.

These are realistically variable and controlled by separate control loops, but this dynamic is ignored and they are set to their respective set point values.

The valve equation is combined with additional elements in order to be able to simulate faults. The valve dynamics is located in the “*complete valve(1417_0304)*” subsystem (see Figure 3-4).

$A(h(t))$ can be calculated with basis in the figure below.

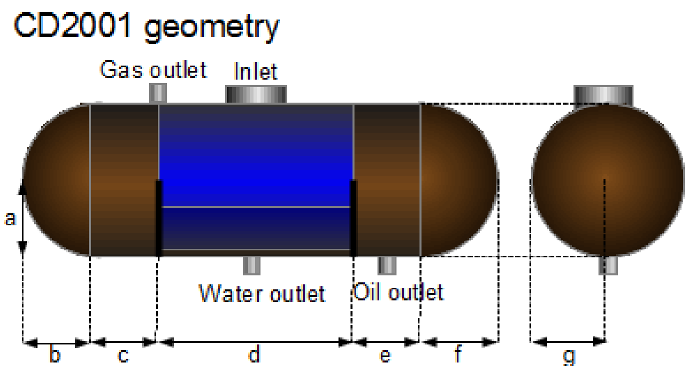


Figure 3-2: Geometry of the inlet separator.

Only the volumes represented in brown are considered as part of the simplification. Dimensions are obtained from [18]. Both ends can be considered as perfect spheres ($a = b = f = g$), which makes it easy to calculate their contribution to the cross section area as function of the oil level, h.

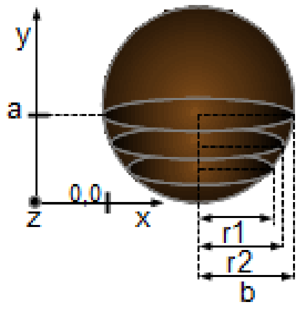


Figure 3-3: Both edge spheres have been combined to a whole sphere for calculations. Radius of this sphere at oil height h(t) is the same as the width/2 of the cylinder area (with length c+e).

The entire cross section area occupied with oil is

$$A_{total}(h(t)) = A_{cylinder}(h(t)) + A_{sphere}(h(t)) \quad (14)$$

$$A_{cylinder}(h(t)) = (c+e)r(h(t)) \qquad (15)$$

$$A_{sphere}(h(t)) = \pi r(h(t))^2 \qquad (16)$$

where

$r(h(t)) = \sqrt{\frac{a^2}{2} - (h(t)-a)^2}$, derived from the standard circle equation; $x^2 + y^2 = r^2$.

The resulting model is implemented in continuous form as a block diagram (see Figure 3-4).

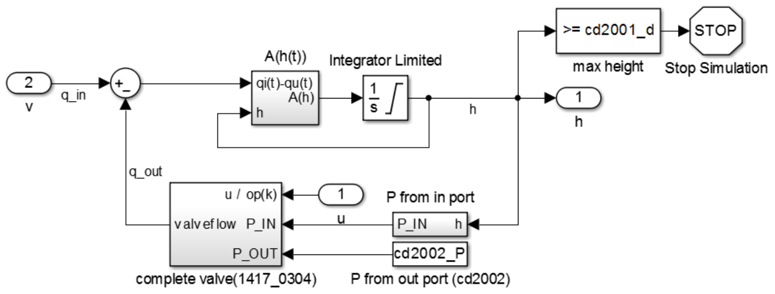


Figure 3-4: The simplified Simulink model produced according to the equations describing the input-output- and valve dynamics.

Symbol	**Description**	**Unit/value**
$w_i(t), w_u(t)$	*In/outgoing mass flow respectively*	$\frac{kg}{s}$
$q_i(t), q_u(t)$	*In/outgoing volume flow respectively*	$\frac{m^3}{s}$
$A(h(t))$	*Cross section area of light side as function of oil height*	m^2
Kv	*Valve coefficient in metric units. Flow through valve at 1 bar pressure drop*	$\frac{m^3}{h * barG}$
$\Delta P(t)$	*Pressure drop across the valve*	$barG$
g	*Gravitational constant*	$9.81\frac{m}{s^2}$
ρ_{oil}	*Density of oil, this is an estimation and may vary*	$836\frac{kg}{m^3}$
a, b, f, g	*Radius of edge spheres*	$1.9\ m$
c	*Length of left light side portion of cylinder*	$1.2\ m$

d	*This value is not used in the simple model*	$11090\ m$
e	*Length of right light side portion of cylinder*	$1.222\ m$

Table 1: Symbol table for the Simplified Simulink model.

3.2.2 Dymola more comprehensive process model

The purpose of all models is, as mentioned, to provide signatures of commonly prominent faults or change of operational or other technical states. The Dymola system's role is to represent the real inlet separator as realistically as possible, with more dynamics introduced to the inlet separator caused by surrounding elements and other control loops. The Dymola model takes both the separator and its close surroundings into consideration. Fluctuations caused by control loops in other elements connected with the production train can also ripple back to the inlet separator, acting as disturbances in its control loops. Development of the Dymola model had to be down-prioritized due to a change of approach to finding the signatures. Rather than diving straight into the complex system, more generic properties were considered of higher importance until a later stage of the technology development.

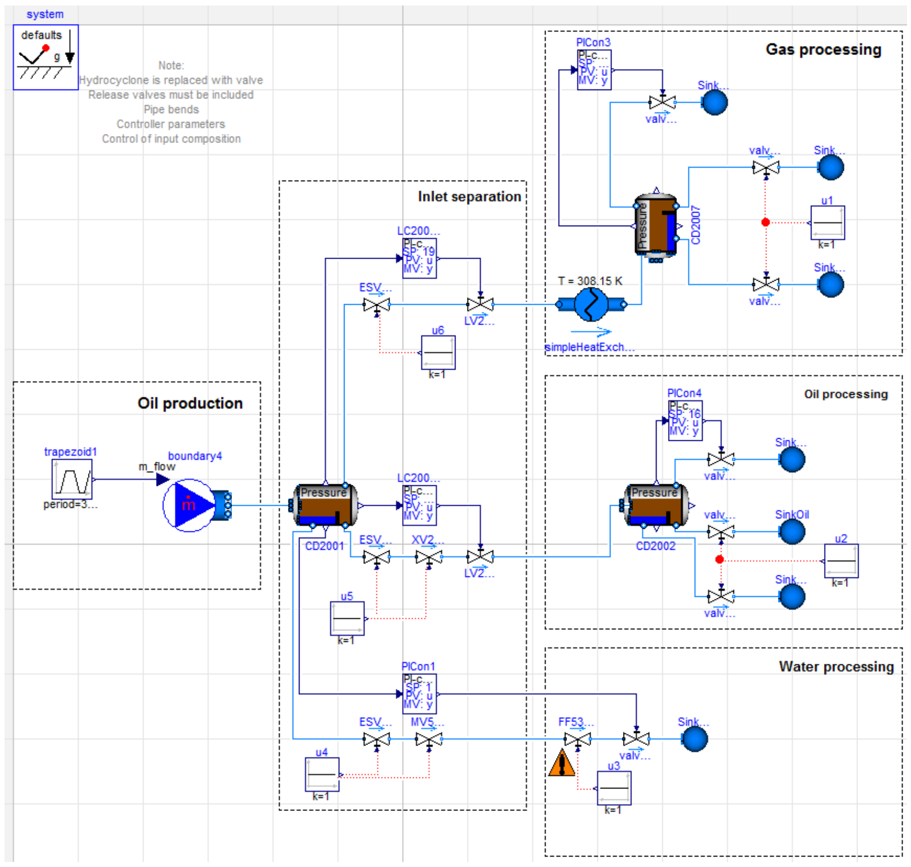


Figure 3-5: Dymola model produced. This was not implemented in Simulink due to the time constraint of the thesis.

3.2.3 Control valve model

The practical valve model is implemented according to specifications in 3.1 and is tested to verify that it behaves as expected (see Figure 3-6). The valve model is constructed in a way that is logical for users to interpret and thus simplifies the verification. The model is created so that signal faults are considered first, then actuation and mechanical features, followed by implemented valve characteristics and flow calculation. The inducing valve fault elements[16] are considered to have root in physical features exposed to operational drift and their rationales follow:

[16] The order of the elements is important.

- Different valves have different valve characteristics and different characteristics should be considered depending on its role in the process, for example a scenario where rapid valve closing is more important that valve opening.
- The actuator diaphragm can gradually rupture creating a loss of power transferred to the actuator stem. The valve's ability to move will be reduced accordingly. Diaphragm leak is considered to be a variable value as % subtracted from the desired pressure applied to the valve stem.
- The valve is considered to be a first order system, with mechanical movement described by its transfer function with response time *valve_T*. Friction on valve stem, weight, actuation method among others affect the time constant.
- Valve saturation can be changed by erosion, corrosion and replacing of the actual valve present in the process. Valves are often implemented with fixed valve saturation for e.g. making 85% physically closed valve correspond to 100% inhibited fluid flow through the valve.
- Stiction is implemented as an S-function in the model. Choudhury's 2 parameter driven valve stiction algorithm gives a good relation between the parameters and the physical observations.
- The valve can develop internal leakage so that its *Kv* is higher than listed in its specifications. Fluid may pass through the valve even in closed position.
- Additional limit blocks in between blocks that may elevate or lower signal values above or below physically possible values are added to keep the model in bounds.

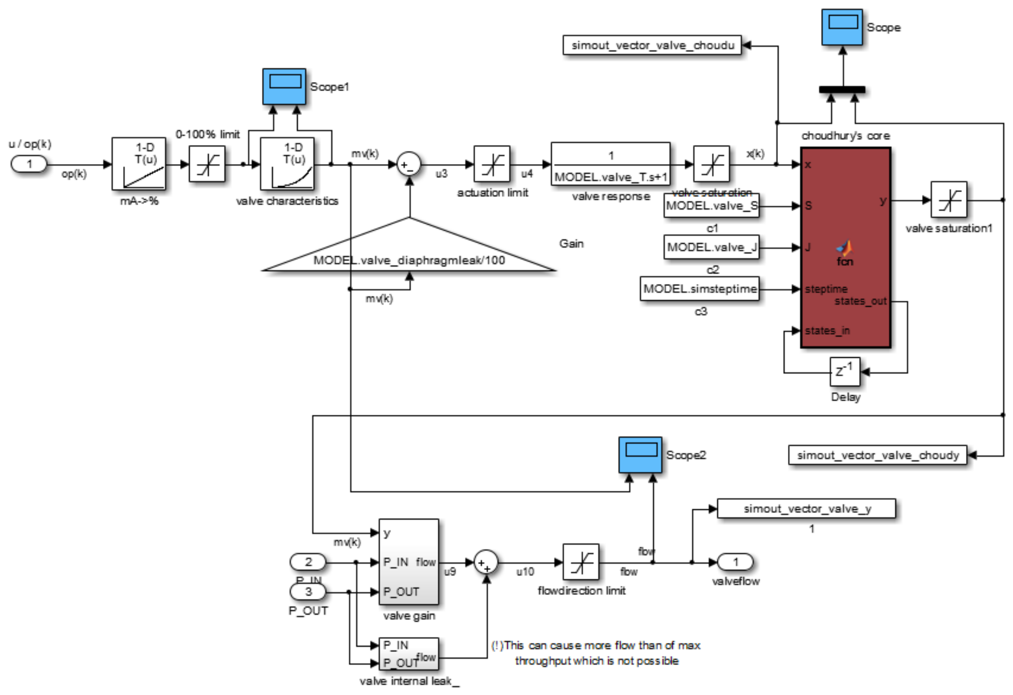


Figure 3-6: Practical valve model implemented in Simulink. Uses three inputs; Controller output, u/op(k), pressure on input side, P_IN, and pressure on output side, P_OUT.

Characteristics of the diaphragm air outlet are not included in the current model because this should be more properly modelled, and depends a lot on the throttling element's capabilities. The potentially arising air spring effect due to air entrapment in the diaphragm, and also the reduced "backward" travel capabilities are considered large.

3.2.4 Level transmitter model

The practical level transmitter model consists of:

- A saturation block. The transmitter's range of measurements is physically and digitally limited so that it e.g. outputs 4-20mA for levels 0-4 meters.
- A filter that may or may not be connected (boolean *transmitter_filter_enabled*). The filter is implemented as a first order continuous low pass filter with its cutoff frequency at *transmitter_Tm*.
- Transmitter characteristics. The physical transmitter can also be incorrectly calibrated so that the interpreted fluid level is biased and/or skewed.
- The transmitter has an IO out part, which has a fixed sample rate and a set resolution for quantizing the bits.

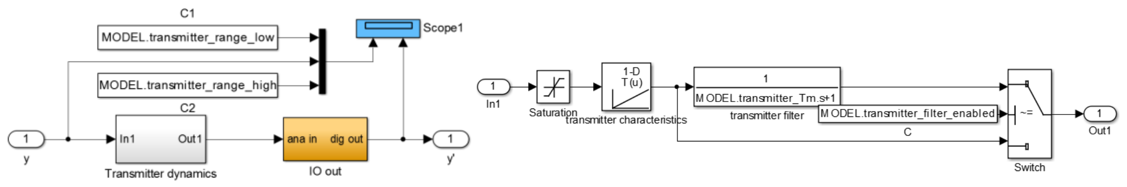


Figure 3-7: Transmitter model. Left is the complete transmitter model, while right is the subelements of the Transmitter dynamics block.

3.3 Design of test mechanism components

3.3.1 Matlab flow control and simulation setup

The Monte Carlo simulation (Figure 3-8) is mostly scripted, because the reusability of converting it to functions is considered small. In addition there are some problems if the programmatic run of the Simulink model exists within a function and the simulation state is saved, as it will be saved to the top level workspace and will be unreachable within the function. The script is divided into small scripts where it is appropriate. The program flow from the main simulation file continues to scripts with the prefix *"component_"*.

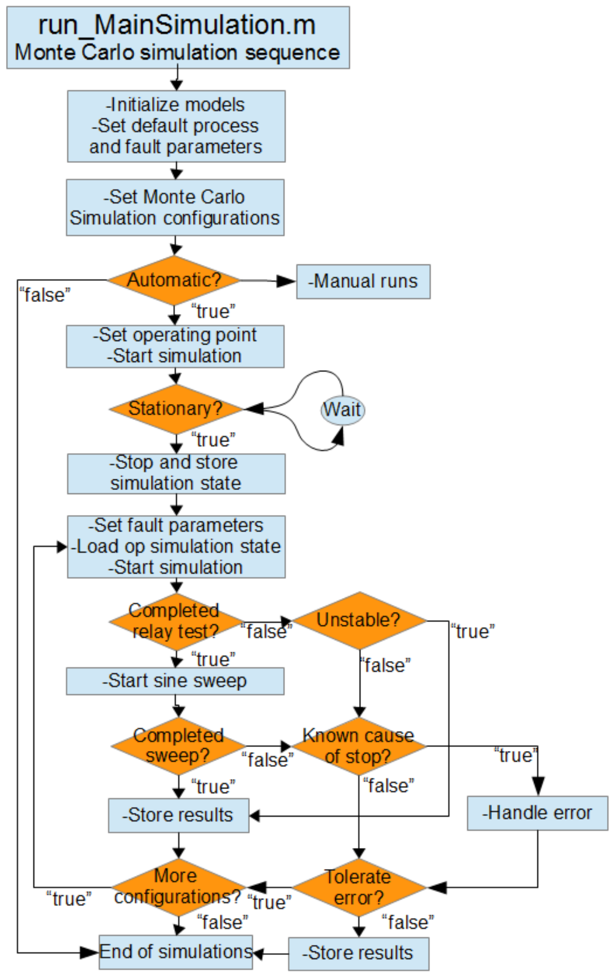


Figure 3-8: Main simulation flow diagram.

The Simulink models are initialized through running the main simulation file "*run_MainSimulation.m"* in Matlab. This initializes all the different elements with default settings. The initialization is divided into several files with the prefix "*init"* in order to make it faster and more intuitive for users to change process and test mechanism parameters. All automatic simulation parameters that are meant to be accessed and specified by users are initialized in the "*prep_simulation_parameters.m"* file. Among these parameters are the vectors specifying the fault parameter configurations the Monte Carlo simulation will step through. The main simulation file will then open the correct specified model and wait for user to make the desired choice of running in automatic mode, which is used for large simulations of many fault and process configurations, or manual mode, which stops the script and lets users manually run the Simulink model, which is good for quick testing.

After initializing the model and fault parameters the main simulation file will set the reference of the next simulation, and then set Simulink to output the simulation states at completion. Simulink simulation is then programmatically started with default parameters. The simulation is automatically stopped when the process output satisfies the conditions for stationarity. The script then continues to load the stationary states, set all the simulation parameter settings to all the pre specified configurations and then start the Simulink simulation again. Each simulation is stopped by the test mechanism when the indicators are obtained. The obtained results and some of the simulation values are stored for verification, analysis and post processing. Script then continues to the sine sweep procedure, which is an extra test to verify the validity of the main test mechanism (the modified relay test). Stationary state is loaded and then the fault parameters are set again. Then a pre specified amount of frequencies spanning from lower than ω_c to higher than $\omega_{-180°}$ are passed through the process model, obtaining gain and phase for each frequency.

After (and during if the processing takes more than a specified amount of time) all results are written to *.mat* files. The files are stored in a readable manner so one can load them to workspace and read the obtained results and see the degree of the faults used for each simulation. There are also some analysis tools for plotting responses and create bode plots for gain and phase responses obtained during sine sweep.

3.3.2 Simulink design

In the top level of the simulation model, digital processing is separated from the elements representing the physicality of the process plant. The system that is under control is easily swappable and is represented as a single block with specified inputs and outputs. Author has developed two models of the inlet separator in different simulation environments, one in Dymola and one in Simulink which are described in their own subsections. Most of the work during this thesis has been spent on the simplified Simulink process model to develop the test mechanism further. The Dymola model should use most of the same interface connections as the Simulink process model.

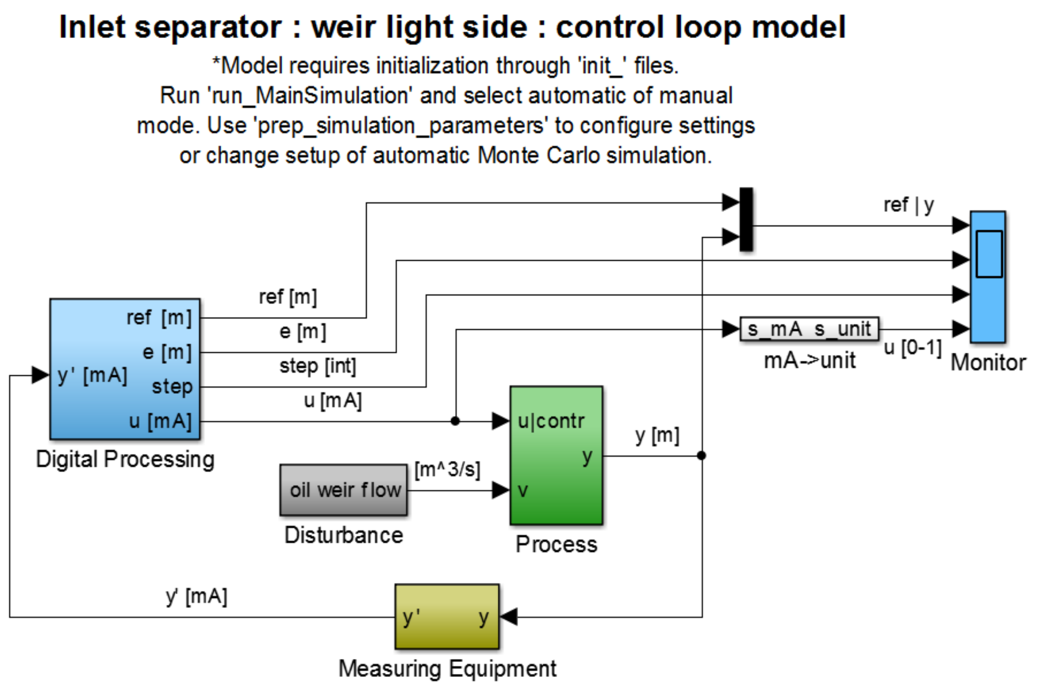

Inlet separator : weir light side : control loop model
*Model requires initialization through 'init_' files.
Run 'run_MainSimulation' and select automatic of manual
mode. Use 'prep_simulation_parameters' to configure settings
or change setup of automatic Monte Carlo simulation.
ref [m]
e [m]
y' [mA]
step
u [mA]
Digital Processing
ref [m]
e [m]
step [int]
u [mA]
ref | y
s_mA s_unit
mA->unit
u [0-1]
Monitor
u|contr
y
v
Process
y [m]
oil weir flow
[m^3/s]
Disturbance
y' [mA]
y'
y
Measuring Equipment

Figure 3-9: Top level view of Simulink model organized with subsystems.

The *Digital Processing* block (Figure 3-10) contains elements that represent the signal processing side of the control loop. Elements found here are:

- IO in; A calculation from mA to meters. The signal from the level transmitter is connected to its input port.
- Performance; Contains blocks for integrated absolute error (IAE) and integral of time multiplied by absolute error (ITAE) performance indicators.
- Reference; Desired level of the closed control loop.
- Stationarity; This block's objective is to indicate whenever the process has stabilized at a set operating point.
- Actuation; Contains the implementation for the PID controller and relay. Can select if the controller is to be used in series or in parallel with the relay.
- Property measurement; This holds the blocks for the real time control loop property estimation such as amplitude, phase, frequency and ultimate values.
- IO out; Quantification of the digital signal and conversion to the appropriate signal level on the physical communication line. Signal from the digital processing is also down sampled to specified rate handled by the process equipment.

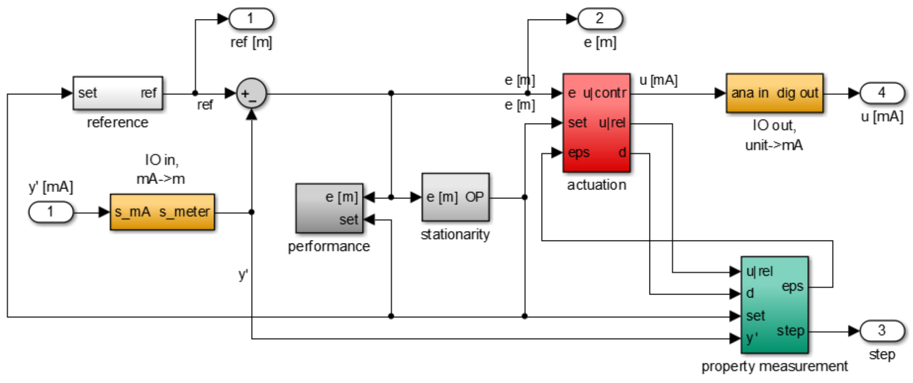


Figure 3-10: CD2001_model_vX/Digital Processing.

3.3.3 Detecting quiescence at process output

When Matlab commences a Monte Carlo simulation the process is brought to stationarity before enabling the relay. Stationarity is defined as the quiescent state of the process output; in other words where the operational states and the internal state of the controller balance the system at a fixed set point. The block “stationarity" in Figure 3-10 is responsible for this and outputs a 1 when criteria are satisfied, 0 else. Stationarity is considered to be when the process-output has stabilized around an operating point with constant process- and equipment- parameters. Stationarity is considered attained when several conditions are true:

- The average absolute deviation-signal is below a threshold [17]
- The average absolute derivate of the deviation-signal is below a threshold

When these conditions are satisfied an element will notify the rest of the simulation model and the relay will be connected into the closed loop. If the relay is set to run with the controller disconnected, the controller output used to hold the process at stationarity is sampled and held, and is used as an output-bias for the relay. The default setting is to have the relay in series with the controller so that the relay bias is given by the controller in order to increase robustness toward disturbances.

[17] The reference-process-output deviation-signal is also down-sampled to the same rate as the measuring equipment.

3.3.4 Detecting asymptotic stability

Detecting when the control system's performance is so degraded that it would have been manually noticed by the process operators is important. There is no relevance to run simulations for cases that will never have the time to develop, and it is important for the Monte Carlo simulation to not spend unnecessary time on such cases. The asymptotic stability criteria is therefore implemented in the test mechanism by checking if the closed loop process gain is larger than a given threshold, and in such a case the simulation will be prematurely terminated and marked as unstable. Stability properties are:

- Asymptotic stable system: $\omega_c < \omega_{-180°}$
- Marginally stable system: $\omega_c = \omega_{-180°}$
- Unstable system: $\omega_c > \omega_{-180°}$

Interpretation of these properties leads to the asymptotic stability threshold implemented in the test mechanism:

$$\textit{Asymptotic stable system: } \frac{y_m}{\frac{4M}{\pi}} < 1/2 \qquad (17)$$

Where y_m is the measured process output and M is the magnitude of the relay output. $\frac{4M}{\pi}$ corresponds to the magnitude of the first harmonic frequency produced by the relay, which is considered *r* (reference) for the controller. Following part of the interpretation is that $\omega_{-180°}$ is produced with zero hysteresis on the relay, which is the case when the test mechanism checks for stability, and that the definition of ω_c is given by:

$$|h_0(j\omega_c)| = 1 = 0dB \qquad (18)$$

Giving the implemented limit for stability:

$$|M(j\omega_c)| = \left|\frac{y(j\omega_c)}{r(j\omega_c)}\right| = \left|\frac{h_0(j\omega_c)}{1+h_0(j\omega_c)}\right| = \frac{1}{1+1} = \frac{1}{2} \qquad (19)$$

3.3.5 Preventing false relay switch behavior

The relay test is susceptible to noise, especially high frequent oscillatory noise while crossing over the reference-process output zero deviation point with zero hysteresis. This can propagate as rapid relay switching. The extra relay switching caused by the noise is outside of the process bandwidth and will not affect the process output at any such degree that it would change the response, but it has another drawback: It can be tricky to accurately read the frequency from the process output, and it is therefore read directly from the edges of the relay switch- points. If the relay then switches multiple times per deviation-zero-crossing this can be a weakness in the robustness of our test mechanism, even if we make a logical mechanism to ignore extra fast switching. Such a mechanism is proposed in the Simulink model, although it disabled and not considered necessary since the rate of the level transmitter and other process equipment con-nected to the process control and data acquisition system (PCDA) is so low.

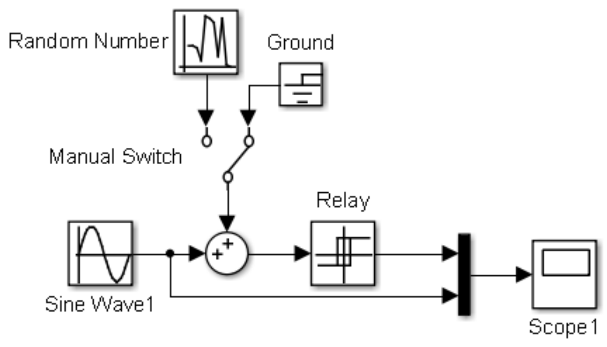


Figure 3-11: Setup for showing false relay switch behavior.

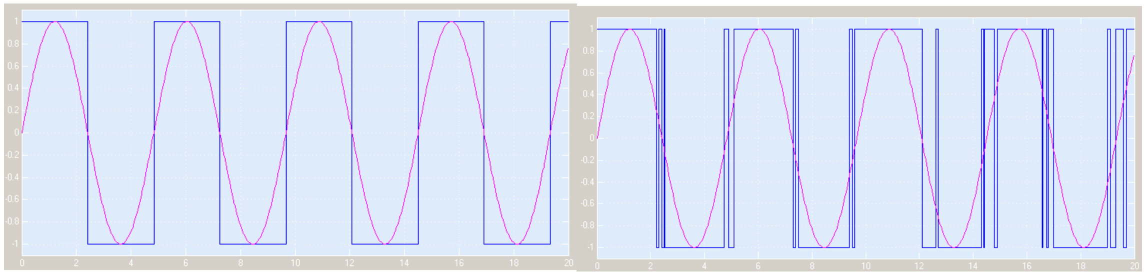


Figure 3-12: Left: Ideal relay switching with little or no hysteresis or filter, right: The effect on noise on relay with little or no hysteresis or filter.

In scenarios of rapid sample rates a small amount of hysteresis with amplitude larger than the noise can also be added so that the noise itself is no longer enough to make the relay switch. It is important to be aware of the issue, but since the simulation model is currently set to the same rate as of the level transmitter at SFC (Ts = 1), which suppresses the problem for now since the process output is then allowed to rise and fall enough between each sampled level that the rapid switching seen in **Error! Reference source not found.** does not occur. If the filter is enabled, it will affect the phase of the relay signal passed through.

3.3.6 Estimating frequency of process output

An element named “*PeriodEst”* in the Simulink simulation model is responsible for finding the frequency of the limit cycles. The element is implemented as an S-function block. The block has several connectors:

Inputs:

- “reset”; Boolean input which resets the block to its initial conditions. This is used when system parameters have changed, such as the relay hysteresis. Reset is connected to activate whenever *SimcycleOp* steps to the next simulation stage.
- “trigger”; Boolean input which causes the block to store the current input at “new_t”.
- “new_t”; The current simulation time, of type double.
- “std_thresh”; The threshold that must be satisfied by the standard deviation of the logged periods for the output “ready” to be true. Its value is given as a percentage of the mean logged period value. The threshold is specified by the Matlab workspace variable “T_std_tresh_pct”.
- “memory_in”; The S-function does not have an internal memory so the logged time values from last simulation time step are fed back as a vector through this input.

Outputs:

- “T_out”; The estimated period of the “trigger” signal. This is the mean value of all the logged periods. The amount of periods considered is specified by the Matlab workspace variable “T_n_relay_edges”.
- “std_out_pct”; Outputs the standard deviation of the estimated period. This can be used for verification and to quantify the threshold value from observations during simulations[18] when results are satisfactory.

[18] Note that when the estimated period is zero, the standard deviation will be *NaN* and will not be shown on the scope.

- "ready"; Boolean output which is true when following conditions are met:
    - Standard deviation < threshold
    - "T_n_relay_edges" periods are included in the average
    - The time since last period has not exceeded a factor time the current period estimate
- "memory_out"; Current logged time values as a vector.

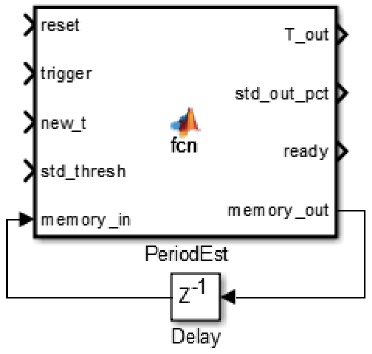


Figure 3-13: ***PeriodEst*** **Simulink block. Located in** ***CD2001_model_v2/property measurement/period estimation/.***

3.3.7 Estimating amplitude of process output

We are interested in finding the amplitude of the specific frequency caused by the relay, but we also want to find the frequency response of the whole system for additional analysis purposes, such as verification of the test mechanism accuracy. For simplicity "frequency of interest" will be shortened to "*FOI*".

The amplitude is defined as the distance between the peak of a period, and an equilibrium, which can be expected as the middle value of a peak and a valley. Since the process output may consist of multiple (and realistically unknown) frequencies caused by process disturbances in addition to the *FOI*, which will affect the height and depth of peaks and valleys consequently, we need to apply a filter to the signal. Since the frequencies can be anything ranging from lower to higher than the *FOI*, we will need a bandpass- filter. In theory the chances that the process disturbances have approximately the same frequency as our *FOI* is low (and hard to quantify) we want our passband to be tight around our *FOI*, with some tolerance toward inaccuracy in its estimation. To satisfactory neglect the influence of disturbance frequencies close to our passband we also want sideband attenuation to be as high as possible. Since we need to find the amplitude as accurately as possible, zero passband ripple is desired. Note that there is a tradeoff between sideband attenuation and passband ripple, and to achieve the filter requirements high order filters are necessary. There is no filter phase response requirement since the time lag of the peaks and valleys won't affect the obtained amplitude readings within each period.

An element named *AmpPhaEst* in the Simulink simulation model is responsible for finding the amplitude of the limit cycles. The element is implemented as an S-function block. The block has several connectors:

Inputs:

- "reset"; Boolean input which resets the block to its initial conditions. This is used when system parameters have changed, such as the relay hysteresis. Reset is connected to activate whenever *SimcycleOp* steps to the next simulation stage.
- "trigger"; Boolean input which causes the block to store the current maximum and minimum of the input signal and start the detection of a new max/min.
- "signal"; The input signal of interest.
- "time"; See the next section.
- "Ts"; See the next section.
- "Tp", See the next section.
- "std_thresh"; The threshold that must be satisfied by the standard deviation of the logged amplitudes for the output "ready" to be true. Its value is given as a percentage of the mean logged amplitude value. The threshold is specified by the Matlab workspace variable "A_std_tresh_pct".
- "variables_in"; The current maximum and minimum values of the input signal (and time values of these; see next section) since last reset/trigger.
- "memory_amp_in"; The S-function does not have an internal memory so the logged amplitude values from last simulation time step are fed back as a vector through this input. *memory_in* holds *A_n_relay_edges* amplitudes for a mean value.
- "memory_lag_in"; See the next section.

Outputs:

- "amp_out"; The estimated amplitude of the input signal. This is the mean value of all the logged amplitudes. The amount of periods considered is specified by the Matlab workspace variable "A_n_relay_edges".

- “pha_out”; See the next section.
- “std_out_pct”; Outputs the standard deviation of the estimated amplitude. This can be used for verification and to quantify the threshold value from observations during simulations[19] when results are satisfactory.
- “ready”; Boolean output which is **true** when following conditions are met:
    - Standard deviation < threshold
    - “A_n_relay_edges” periods are included in the average
- “variables_out”; Current logged min/max values as a vector.
- “memory_lag_out”; See the next section.
- “memory_amp_out”; Current logged amplitude values as a vector.

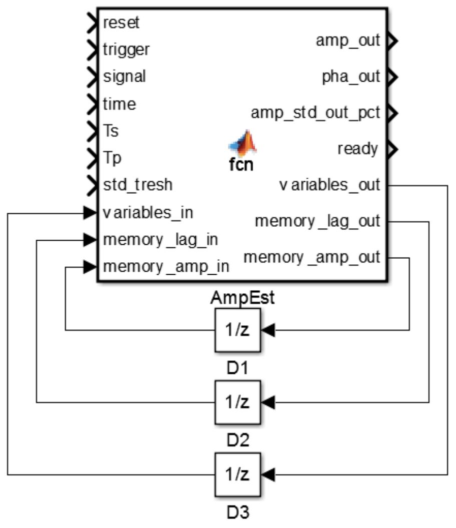


Figure 3-14: *AmpEst* Simulink block. Located in *CD2001_model_v2/property measurement/amp & phase estimation/.*

3.3.8 Estimating $\omega_{-180°}$, ω_c, ΔK, φ_m

The task of *SimcycleOp* in [20] block is to implement the stepwise procedure of finding the gain- and phase margins. The procedure in implemented as an S-function block in Simulink. The procedure is described in the preliminary Bachelor thesis [2] and is summarized as follows:

[19] Note that when the estimated period is zero, the standard deviation will be *NaN* and will not be shown on the scope.

(1) Perform a relay test without hysteresis ($\varepsilon_1 = 0$), calculate the gain margin, ΔK and ω_{-180° .

(2) Calculate first process parameter, β.

(3) Perform second relay test with hysteresis $0.1 * d < \varepsilon_2 < 0.4 * d$. d is the relay amplitude

(4) Calculate the hysteresis, $\varepsilon = \varepsilon_d$, that puts the process at the unit circle, $|G_L(j\omega)| = 1$.

(5) Estimated frequency is $\sim\omega_c$. Calculate the phase margin, φ_m.

The block listens to the amplitude- and period estimators, *AmpEst* and *PeriodEst,* which are located in the same subsystem as *SimcycleOp*. Whenever gain and period criteria are satisfied the block will step to the next current simulation stage. The stages implement the summarized procedure and are:

(1)

- Set $\varepsilon = \varepsilon_1 = 0$
- Wait for amplitude- and period estimators to return true on the *ready* output.

(2)

- $\Delta K = \frac{4*d}{\pi*a}$, where a is the amplitude estimate given by *AmpEst*.
- $w_{-180^\circ} = \frac{1}{Tp}$, where Tp is read from *PeriodEstimate.*
- $\beta = \frac{a}{d}$
- $\varepsilon = \varepsilon_2 = 0.1 * d$
- Wait for amplitude- and period estimators to return true on the *ready* output.

(3)

- $\alpha = \frac{a - \beta * d}{\varepsilon_2}$
- $a_d = \frac{4*d}{\pi}$
- $\varepsilon_d = \frac{a_d - \beta * d}{\alpha}$
- $\varphi_m = \sin^{-1}(\frac{\varepsilon_d}{a_d})$
- Wait for amplitude- and period estimators to return true on the *ready* output.

(4)

- $w_c = \frac{1}{Tp}$
- Mark simulation as completed. End simulation.

[20] Located in *CD2001_model_v2/property measurement/*

SimcycleOp starts the procedure by default whenever the process is considered to be stationary at the operating point.

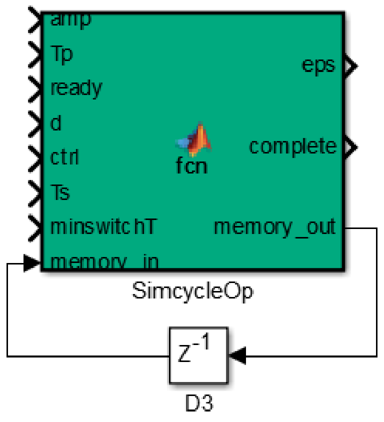


Figure 3-15: The simcycle operator, *SimcycleOp*, Simulink block. Located in *CD2001_model_v2/property measurement/.*

Inputs:

- "amp"; Amplitude estimated by *AmpEst* is connected to this port.
- "Tp"; Period estimated by *PeriodEst* is connected to this port.
- "ready"; SimcycleOp will progress to the next stage when this is true.
- "d"; Current specified amplitude of the relay is connected to this port.
- "ctrl"; Control signal which indicates that the process has reached stationarity.
- "Ts"; The simulation time step is connected to this port.
- "minswitchT"; This input port specifies the minimum time between each progress increment of the procedure.
- "memory_in"; In order for the block to keep track of progress and hold onto necessary variables during runtime, states of the last simulation time step are fed back to this port.

Outputs:

- "eps"; This port outputs the current hysteresis to be used by relay.
- "complete"; This port outputs true when the procedure is completed, which means all properties have been estimated. This is default connected to a *Simulation Stop* block, which terminates the simulation.
- "memory_out"; Outputs the current block states and internal variables.

3.3.9 Alternative estimation methods for further investigation

Ideally all implemented methods should provide equal results for verification, and there may be a difference of robustness between the proposed methods and other methods that are not mentioned in this thesis. Other methods for estimation may be interesting to further investigate for the final implementation of the test mechanism. Some methods may increase the accuracy of estimation for overall increased accuracy in results and diagnostics found by the test mechanism.

Finding the most reliable and precise estimation technique was not a topic of this thesis, but it is important nonetheless. Alternatives can either replace or work in collaboration with the current estimation techniques. If all methods are proven reliable they can back each other up so if none provide outlying results one can safely assume that the obtained results are correct, which is an important aspect for practical application. The amplitude and frequency of the limit cycles induced by the modified relay test can for example be estimated with an extended Kalman filter [19], or by least squares estimation.

When it comes to the practical appliance of the test mechanism in a physical plant it is advantageous that it is active for as little time as possible, and there will likely be small time windows of opportunity open for the actual testing. Considering that, letting properties be estimated during runtime and then let the test mechanism tell when it is complete and then end is a good idea. When it comes to the sine sweep procedure this is a topic of little concern since the sine sweep is only used during the simulation study for understanding and verification. In this case estimating in real time or post simulation does not matter.

3.4 Design of sine sweep components

3.4.1 Estimating amplitude and phase for sine sweep procedure

An estimation technique for obtaining the phase response for the Bode plot for a set configuration of process and fault parameter settings is required for the simulation study. Three techniques are suggested, whereas one is focused on a simple way of measuring phase during simulation, the other two are directed toward post simulation estimation. An important notice is that the first suggestion needs to extend the Simulink model with an additional amplitude- and phase estimation block at the controller output. Its strength is its simplicity and. To circumvent adding extra elements to the Simulink model, the results are obtained post simulation from method number three (FFT).

During simulation:

The *AmpPhaEst* block has an increased functionality and also detects the phase shift of the input-output signals. This is needed for the sine sweep procedure used to verify the results acquired using the test mechanism. This is done in virtual real time as well. Limitations to the phase estimation using this method is that it is unable to determine if the phase is more than 360°, which means that if the phase passes 360° during testing will cause it to be read as a much lower value. This should not be a huge concern though since it is only needed to sine sweep for frequencies between the edge points ω_c and $\omega_{-180°}$.

Algorithmic description of the method:

- If there is a reset signal detected, reset all values to the initial states.
- If there is a trigger signal set time of trigger, peak and valley variables to current time.
- Otherwise:
    - If new maximum value, set time of peak to current time value.
    - If new minimum value, set time of valley to current time.
- Recalculate output values with new variables

The phase calculation is done by taking the mean of the peak phase lag and the valley phase lag. This should compensate for constant nonlinearities around an operating point. The resulting phase is also a mean over the same amount of periods as the amplitude. Since a sine starts in 0 and increases until 1/4th of its period, corresponding to 90°, 90° is subtracted from the peak phase lag value. Same applies to the valley phase lag, but this is located at 3/4th on a sine with no phase lag, so that 270° is subtracted from the valley phase lag value.

Since the phase detection functionality is built into the *AmpPhaEst* block, the same inputs/outputs apply, although there are some inputs that are specific for the phase detection functionality:

- “Ts”; The sample rate of the *AmpPhaEst* block. This should be equal to the rate of the process transmitter. Its value is used to correct the estimated phase by subtracting $\frac{Ts}{2Tp}$ from the phase, so that the theoretically estimated phase is $\pm\frac{Ts}{Tp}$ with expectancy value 0.

Specific added inputs:

- “time”; The current simulation time.
- “Ts”; The sample time of the test mechanism/process transmitter.
- “Tp”; The currently estimated output signal period.
- “variables_in”; Also contains the time value of trigger/peak and valley.
- “memory_lag_in”; Also contains the *A_n_relay_edges* last estimated phase lag values for a mean.

Specific added outputs:

- “pha_out”; The currently estimated phase.
- “memory_lag_out”; Currently logged phase lag values.

Post simulation method 1 (sine fitting):

As an alternative to estimating the phase directly during processing, it is also estimated post simulation. There have been some problems with the method described above during the last stages of the work so that the estimated phase used for the simulation study is derived from the post simulation. There are numerous ways to estimate phase, but to keep it simple and by using already obtained results, the phase estimation is done by fitting sine waves on the stored input and output data. This method is easier to verify since the stored input and output data from simulation can simply be overlaid with the adapted sine waves. The frequency of the adapted sine is known since the frequency will remain that of the input, and the amplitude is obtained from the simulation amplitude estimation, although this can also be easily be read from the stored data. This leaves only one degree of freedom for the sine wave; the phase. The method iteratively approximates the input and output signals by overlaying sine waves with the estimated amplitudes and known frequencies. It then looks for the phase match that minimizes the sum:

$$\sum_{t=t1}^{t2}\left(signal(t) - A * sin\left(2\pi f t - phase\frac{\pi}{180}\right)\right)^2 \qquad (20)$$

Where the signal is de trended by removing its mean value. Notice that this method is also limited to search for phase lag $\in [0, 359]°$. The value of *t1* is chosen so that the response is stabilized before the sine approximation.

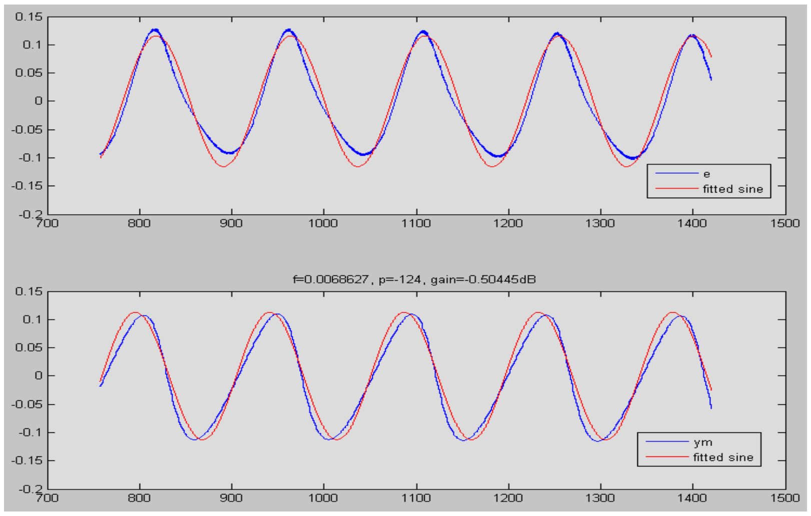


Figure 3-16: Example of control loop input (top) and output (bottom) fitted with sine waves to find phase.

3.5 Project component tests

The components of the Simulink model have been tested and analyzed according to an adaptation of *BS 7925-2:1998*, «A British standard for testing of software components and techniques for the design and measurement of that testing». The full standard should be considered for the final completion of the test mechanism, but due to the time constraint of this thesis some simplifications have to be done.

Some parts of the system are considered particularly prone to bugs and other faults. These components are tested and verified that the outputs for the specified inputs are as expected.

- Amplitude estimation
- Period estimation
- Modified relay method for known system
- Modelled valve behavior

The tests will be executed on a windows 7 pc. The components to be tested consist of a combination of Simulink blocks and Matlab (R2012b) code.

Components shall be tested in isolation with both normal and abnormal inputs on the connectors. Signals cannot exceed realistic range which is naturally limited by the physicality of the real system modelled. System as a whole shall be tested and verified semi- isolated and by locking most variables.

Primarily there are no dependencies between the components being tested since each component is tested in isolation. Lower priority components can tolerate more misbehavior than that of higher priority components. All test setups and scripts are found in the *“Component Tests”* project folder.

3.5.1 Amplitude and phase estimator (“AmpPhaEst”)

AmpPhaEst is located in *CD2001_model_v4/Digital Processing/property measurement/Amp & phase Estimation/*. This is considered a critical block for the system’s ability to accurately obtain the necessary indicators both for the final implementation of the relay test mechanism, but also to obtain the process footprint for the simulation study and validation of the test mechanism.

Stubs: The *Amp & phase Estimation* subsystem is isolated from its surroundings. The reset port is forced to 0.

Driver: A sine wave source on the input of a relay replaces the process feedback which results in the *Pv-Sp* deviation signal. The period of the sine waves are controlled by the test script. A different sine signal with known amplitude and phase acts as the *Pv* with the properties of interest.

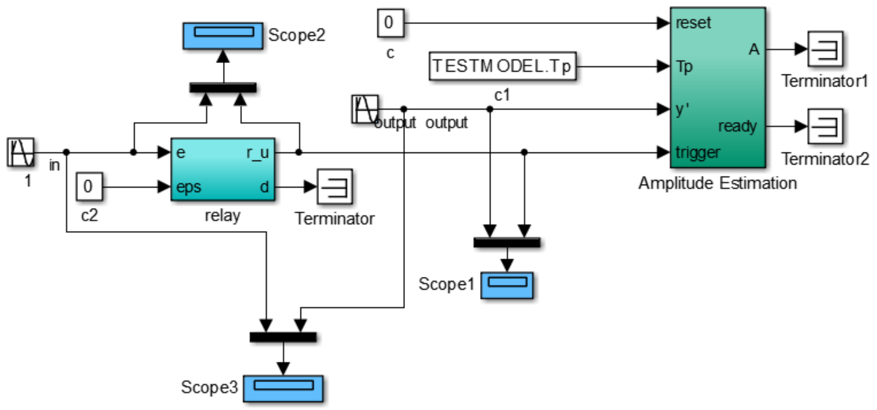


Figure 3-17: Test setup for *AmpPhaEst* with selected stubs and drivers.

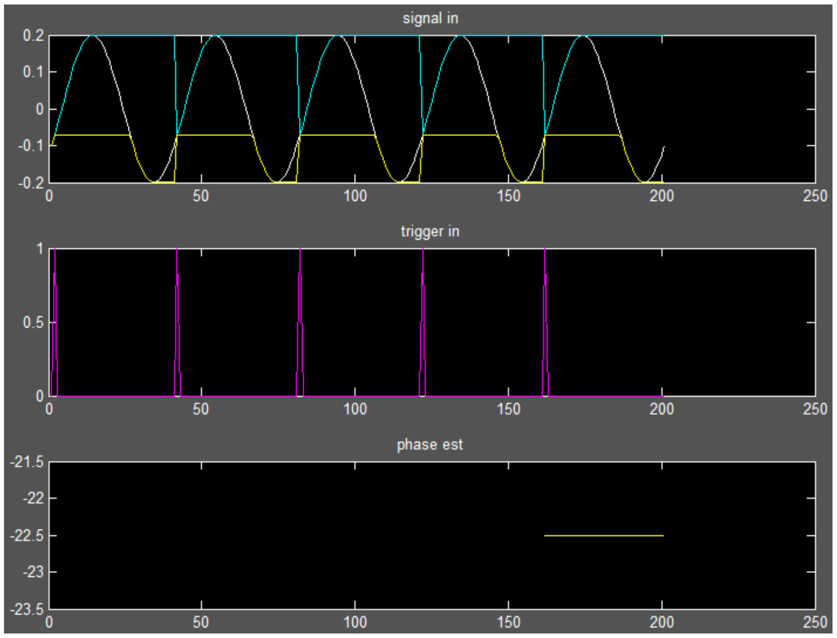


Figure 3-18: Upper plot: Current max value as cyan, min value as yellow and measured signal is white.
Middle plot: Triggers given by positive relay flanks (deviation signal to controller passes 0)
Bottom plot: Estimated phase. Phase vector is initially *NaN* and after reset and is therefore not displayed.

The block's typical behavior can be seen in Figure 3-18. It uses the pre specified amount of cycles before it outputs the control signal indicating that the output values can be read. The estimated phase is not used in the modelled test mechanism and "*footprint*" mapping algorithms, since using *FFT* was performing very accurately and reliably.

3.5.2 Period estimator ("PeriodEst")

PeriodEst is located in *CD2001_model_v4/Digital Processing/property measurement/Period Estimation/*. This is considered a critical block for the system's ability to accurately obtain the necessary indicators.

Stubs: The *Period Estimation* subsystem is isolated from its surroundings. The reset port is forced to 0.

Driver: Sine wave sources on the input of a relay. The period of the sine waves are controlled by the test script.

Results can be read from the output scope inside the *Period Estimation* subsystem block. The test is run for a few sets of different periods. Two different sine waves are applied to the system for each test run. The sine waves are each run for 7 periods. The estimation is set to average over the last 3 logged periods. The standard deviation threshold is set to 20%.

PeriodEst should with high accuracy be able to quickly estimate the period of the relay switches, indicating the frequency of the limit cycles induced by the test mechanism. The block should indicate when the standard deviation for the logged periods are below a threshold and output a logical 1 on its *ready* port accordingly.

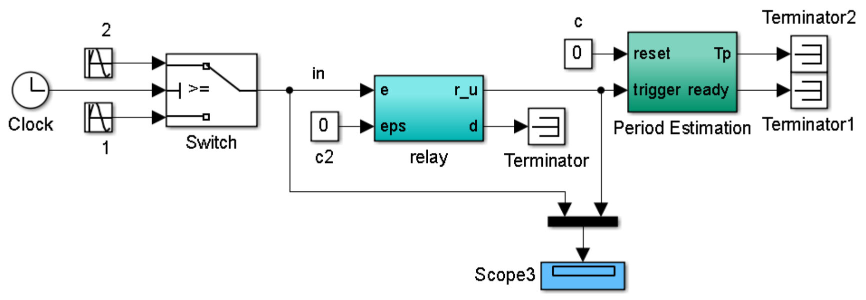


Figure 3-19: Test setup for *PeriodEst* (located in the *Period Estimation* subsystem).

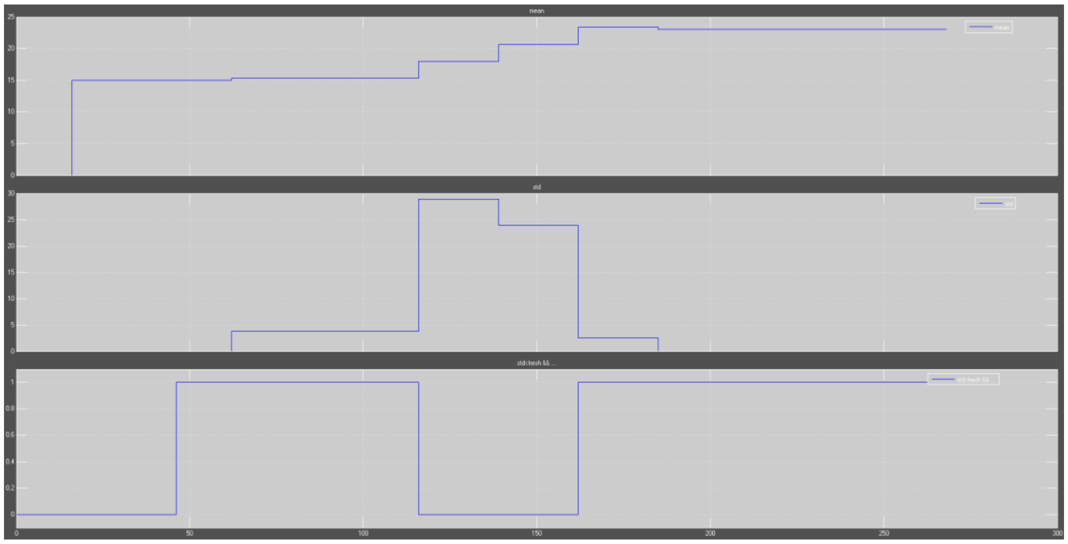


Figure 3-20: Scope from testing PeriodEst with stubs and drivers. Upper plot shows how the estimated period changes over time. Middle plot shows the standard deviation of the period samples in %. Lower plot shows the output signal of the *ready* port.

Normally during simulation with the test mechanism active, the *PeriodEst* is reset each time the test mechanism goes to the next stage of the test procedure, but during this test it is interesting to see how the estimate changes over time. There is implemented a standard deviation threshold of the period samples, which when satisfied will make the block output a logical 1 on the *ready* output port.

Exact Tp [s]	Applied at time, [s]	Indicating change of Tp at time (*ready* = 0)	Time of *ready* =1, [s]	Estimated Tp, [s]	Avg. error, [%]
15.3	0	NA	46	15.15±0.15	0,99
23	107.1	116	162	23.2 ±0.2	0,86

Table 2: Limit cycle period/frequency estimator test results.

Indicating the change of Tp is done by setting the ready output to 0 when the standard deviation is above the threshold. The threshold is set to 20%, but this adjustable. Another improvement could be to weight the newer samples more than the old ones in order to make it respond faster to fast new changes. Other than this, the measurements are satisfactory, and their accuracy is limited by the rate of the test mechanism, which is only allowed to run as the same rate as the process measurement equipment.

There is probably room for some improvements in the mechanic of the component, but its final design will be decided on a later stage of the technology development.

3.5.3 Valve

The valve has been isolated and tested according to the setup described in Figure 3-21. The valve model is detailed in 3.2.3.

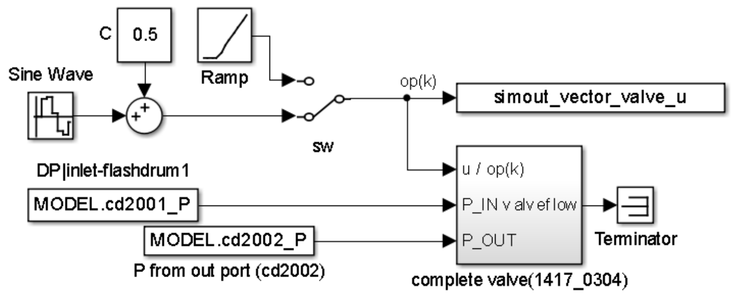


Figure 3-21: Test setup for testing of complete valve model.

The valve was thoroughly tested with the effects of adjusting the different fault parameters. Figure 3-22 and Figure 3-23 demonstrate the effects of adding slip jump and stick band characteristics to the valve. The effect the stiction has on the flow through the valve can be seen in Figure 3-24.

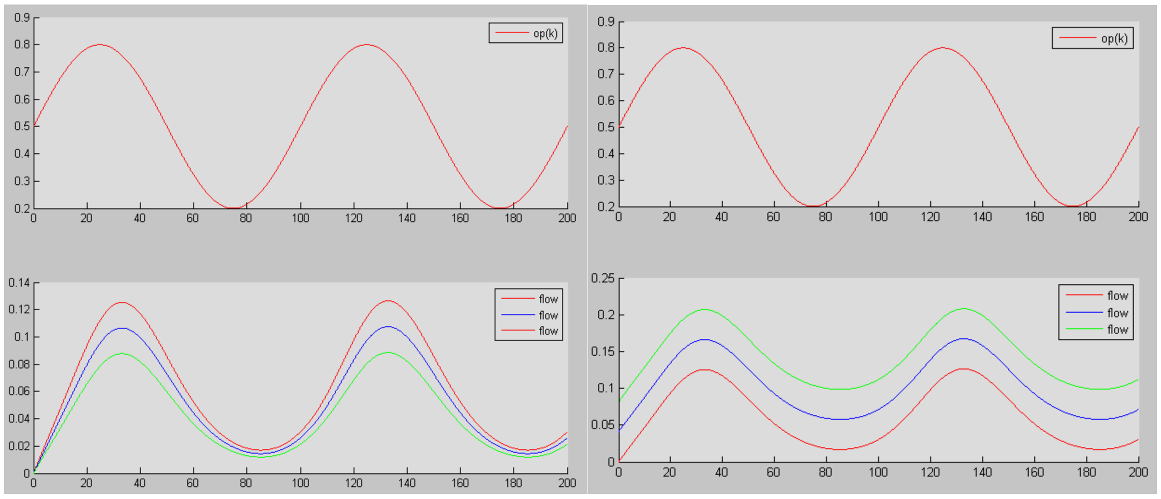


Figure 3-22: Left plots: Effect on flow by diaphragm leak, respectively (from top) 0%, 15% and 30%. The effect of the leak is assumed to be proportional with the controller output.
Right plots: Effect on flow by internal leak, respectively (from top) 30%, 15% and 0%. The effect increases the effective Kv for the valve, increasing throughput. In the case that the valve wants to close, fluid will still flow through.

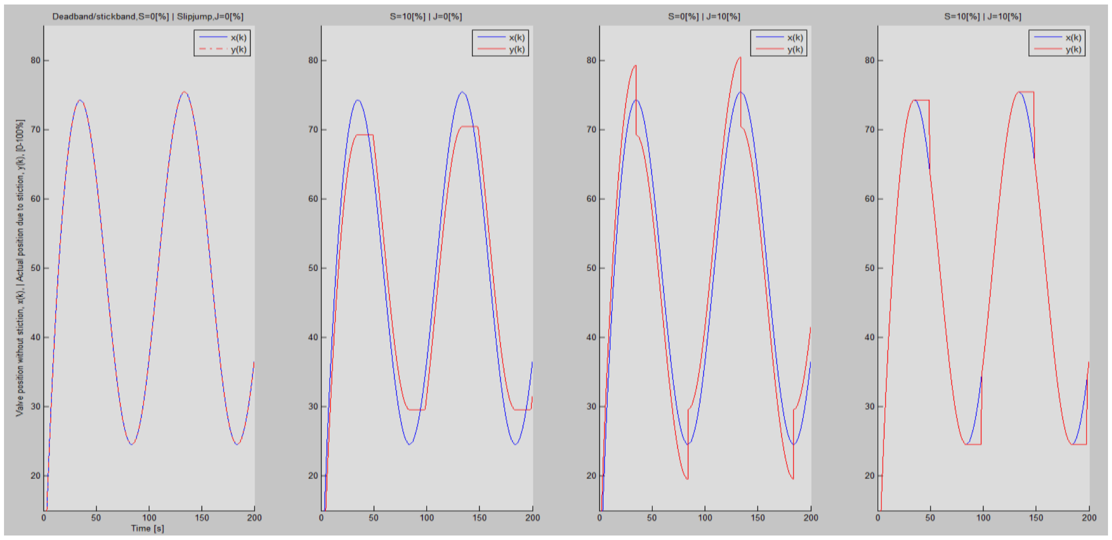


Figure 3-23: Effect of valve stiction. Plot descriptions from left to right: 0% stickband and slipjump, 10% stickband and 0% slipjump, 0% stickband and 10% slipjump, 10% stickband and 10% slipjump.

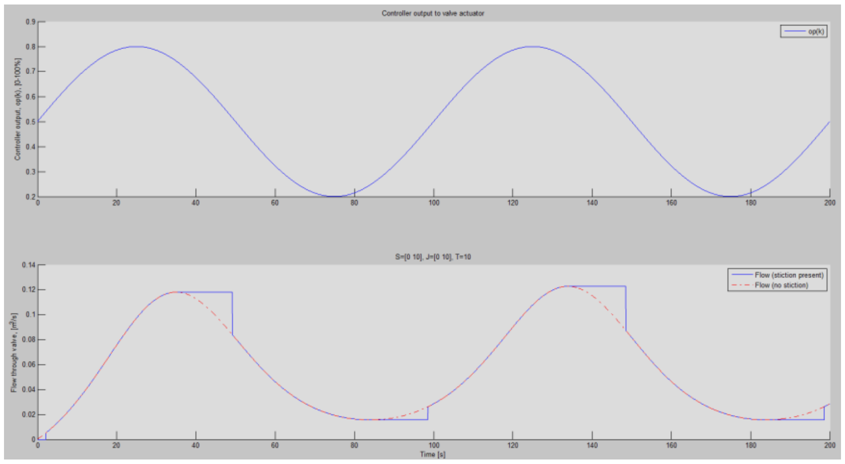


Figure 3-24: Effect of valve stiction on the flow through the valve. Plot for 0 stiction (red), and 10% slip jump and stick band in blue.

3.5.4 Modified relay testing

It was need to prove that the modified relay setup actually worked, and there were run several tests with the setup shown in Figure 3-25. Results were proved to be near identical to those indicated in Figure 3-26. This removes the concern of bugs in this part of the complete Simulink model and associated code.

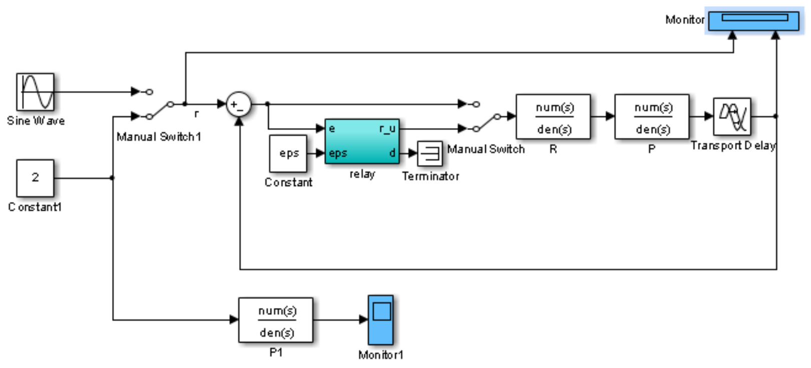


Figure 3-25: Modified relay test with additional sine sweep frequency testing for known system. Results are used to verify the methods used for finding the signatures (process "footprints").

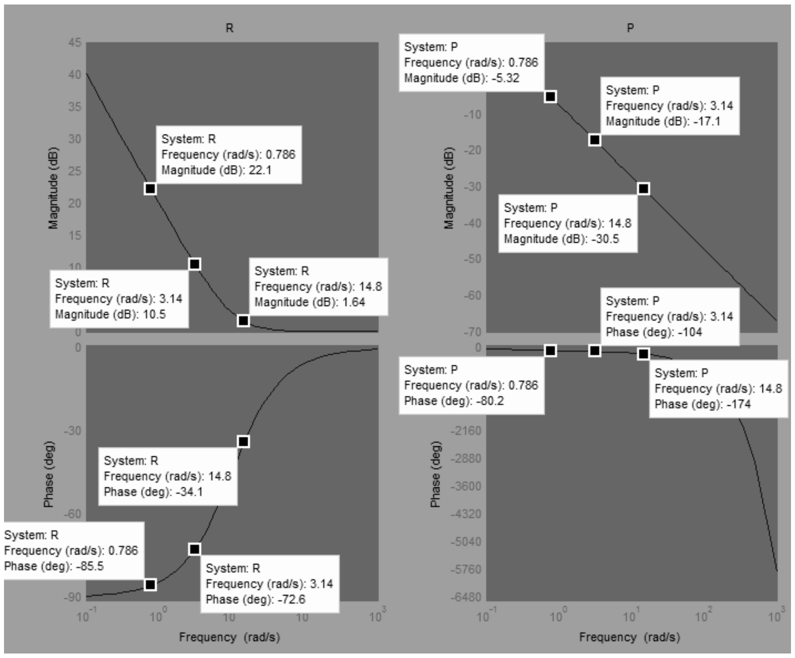


Figure 3-26: Exact Bode plot characteristics for the known controller and process.

3.5.5 Test of Simulink setup for known system, with relay test and sine sweep

The top level Simulink model with all the components, including relay, sine sweep and estimator blocks was tested on a known fictional system that replaced the simple Simulink process model. This was done in order to validate the estimation techniques before applying them to the continuous and more realistic process models.

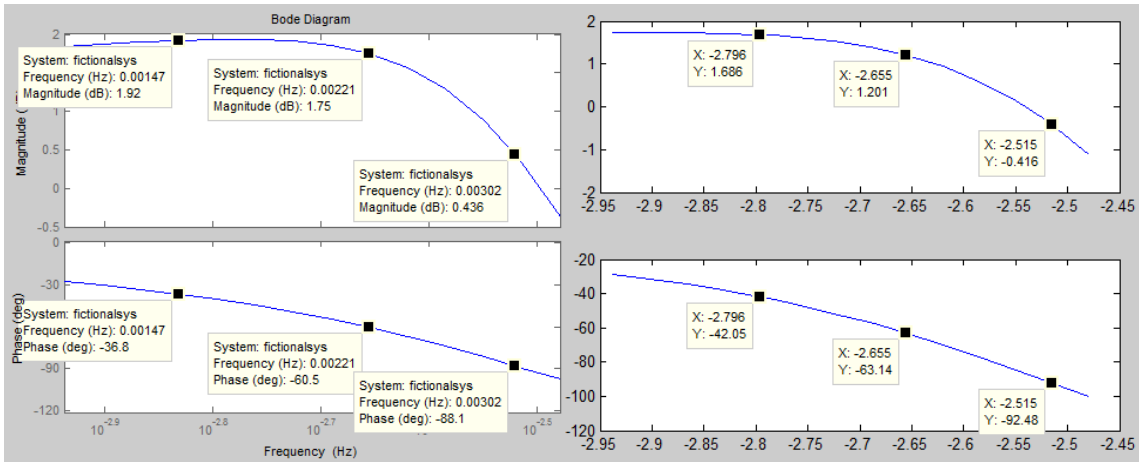


Figure 3-27: Left: The exact bode plot for the known system. Right: Frequency response found from sine sweep with post FFT processing on controller and process output. Note that the axis are not equal.

The fictional system created for testing is an under dampened second order system with time delay, in continuous (Laplace) form (step response and pole-zero plot can be seen in appendix):

$$\frac{0.00048}{s^2+0.024s+0.0004}e^{-3s} \qquad (21)$$

The results obtained from the sine sweep are closely similar. There will be some differences due to the discretization on the digital processing part of the top level Simulink model ($T_s = 1s$). Additionally the FFT estimation approach can produce some dissimilarity due to some of the applied sine wave energy is shifted into other frequency components due to the presence of nonlinearities. The time delay of the fictional system is one such nonlinearity.

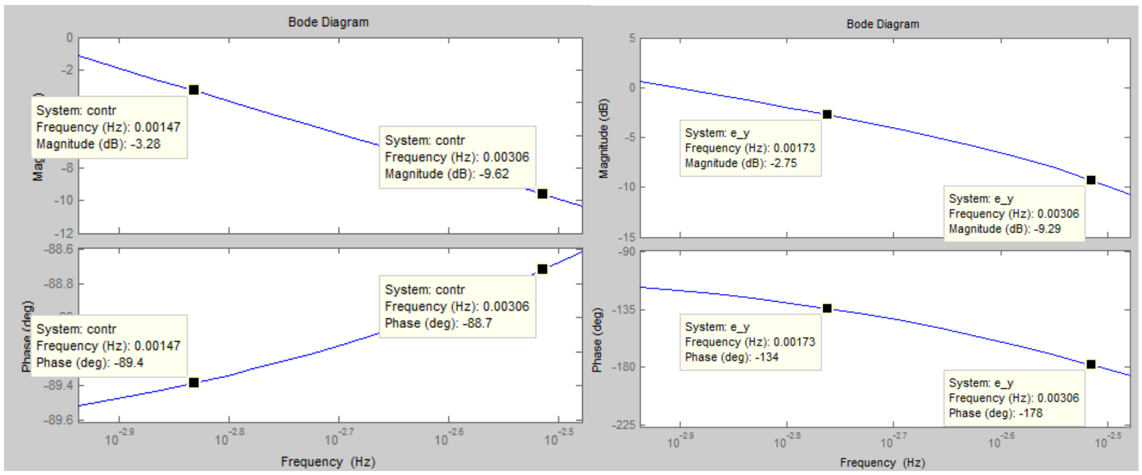


Figure 3-28: Left: Exact bode plot for the known controller, R(s). Right: Bode plot for controller and process in series (open loop), h0 = R(s)P(s). Marked data points are the indicated wc and w180 from the relay test. H0 should be equal to R(jw) (known) + P(jw) (unknown) for the estimated response, which we see is approximately true.

Obtained indicators from the relay test can be seen in Table 3. There are some slight variances introduced because of the Ts of the digital processing side. Allowing the relay test to run for a longer amount of time can increase accuracy, but for testing purposes it is kept low with respect to the required simulation time.

Indicators	*Relay test*	*From bode plots*	*~Error*
ω_{180}	$0.0030\ [hz]$	$\sim 0.0030\ [hz]$	$+0\ [hz]$
ω_c	$0.0017\ [hz]$	$\sim 0.00125\ [hz]$	$+0.00045\ [hz]$
ΔK	$9.83dB$	$\sim 9dB$	$+0.83$
φ_m	$32.3°$	$58°$	$-25.7°$

Table 3: Results from the modified relay feedback test versus the exact results from the bode plot for the known systems.

As seen by Table 3, ω_c and the phase margin, φ_m, are more prone to error than the other readings. This is because of added sources of errors due to the approximations and estimations used. It is therefore important to give an extra weight on the techniques used for the final implementation of the technology.

3.5.6 Test of exactly known system with relay test and relay sweep

The modified relay test was extended with the relay sweep "footprint" estimation techniques. The transfer functions for the processes tested were:

$$P(s) = \frac{2.2}{50s+1}e^{-0.1s}$$

$$R(s) = \frac{0.1s+1}{0.1s}$$

Results can be seen in Figure 3-30. The transport delay sweep was later replaced with *FFT*, since the *FFT* was more accurate and reliable in most cases.

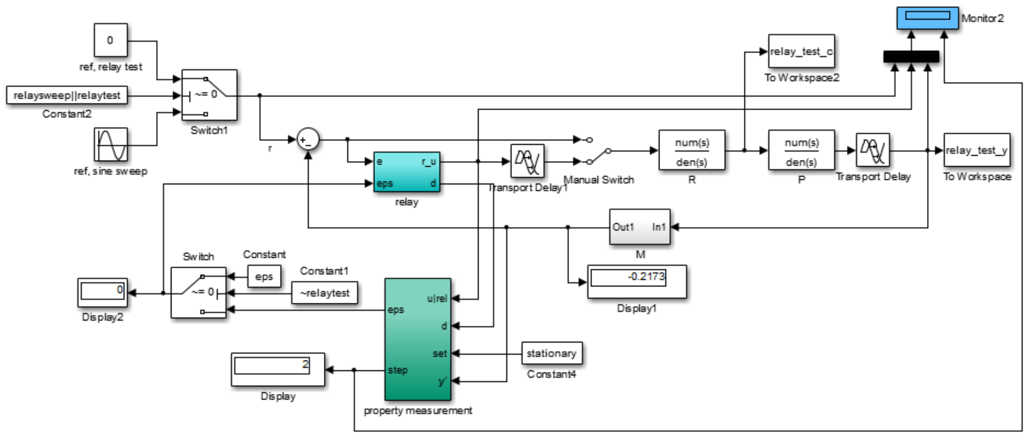


Figure 3-29: Test setup for modified relay test and relay sweep adjusting hysteresis and transport delay.

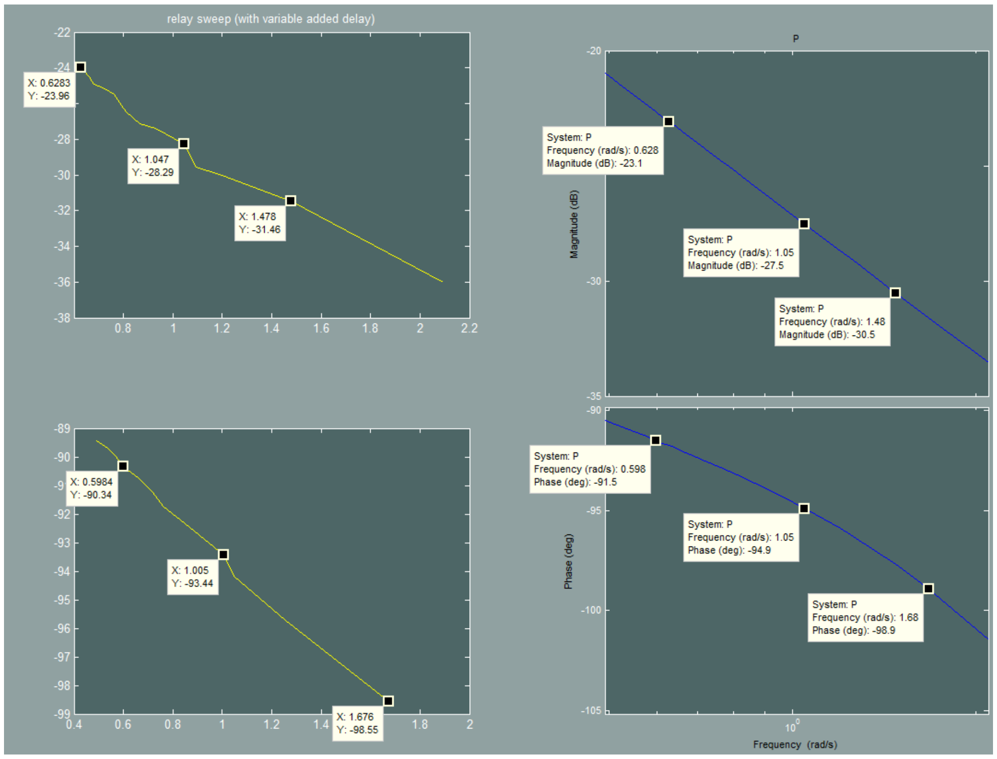


Figure 3-30: Left: Obtained response with relay sweep. Hysteresis was adjusted to obtain the amplitude response and transport delay from relay was adjusted to obtain the phase response. Right: Exact bode plots. As seen, the obtained values are very similar to the exact values.

Chapter 4
Simulation study

4.1 Simulation setup and some regarding comments

For the final simulations the decided estimation techniques to obtain the “process footprint“ was relay sweep by adjusting the hysteresis between ε_{ω_c} and 0 (described in 2.8.2) to obtain the process gain, and FFT on the *Op* and *Pv* signals to find the phase lag. These methods proved reliable and effective in combination. This also removed the need for the sine source since the oscillations were generated by the relay alone throughout the simulations. In retrospect, the added hysteresis should not have been linearly incremented, but rather inverse exponentially incremented. At first, small increments of hysteresis result in larger shifts in logged frequencies, so the process “*footprint*” resolution is poorer close to ω_{-180° than at ω_c.

The simulations were ran for all combinations of the parameters listed in Table 4, except for $cd2001\_P < cd2002\_P$ (this would cause reverse flow which is not relevant) and $S < J$ (this causes the valve to behave like a relay, so that there are now effectively two relays in the loop during testing; would have been manually detected by process engineers).

Parameter	**Type**	**Value(s)**	**Run priority**
Reference	Operational	0.7, 1.2	[m]
Oil flow in	Operational	0.05, 0.07	$[\frac{m^3}{s}]$
Pressure CD2001	Operational	19, 27	[barG]
Pressure CD2002	Operational	16, 18	[barG]
Diaphragm leak	Technical, valve	0, 10	[%]
Internal leak	Technical, valve	0, 10, 20	[%]
Upper saturation	Technical, valve	100	[%]
Lower saturation	Technical, valve	0	[%]
Time constant	Technical, valve	10, 15	[s]
Stick band	Technical, valve	0, 10, 20	[%]
Slip jump	Technical, valve	0, 10, 20	[%]

Filter time constant	Technical, transmitter	0, 20, 60	$[\frac{1}{w_f}]$
Bias	Technical, transmitter	0, 0.05	[m]
Skew	Technical, transmitter	1, 0.9, 1.1	[Factor]

Table 4: “Big” Simulation parameter configurations.

The resulting configurations with the exceptions add up to 20’736 simulations. In order to produce results for analysis for a variety of configurations, the resolution (amount of values for each parameter) had to be low. This was the last big (thus nicknamed “big”) simulation run producing a lot of interesting results although there wasn’t enough time to let it finish. Different configurations had different priorities, so those considered to be of higher importance were run first. The last Monte Carlo simulation finished 308 simulations in approximately 16 hours. This indicates an optimization issue and it is not feasible to run it for all the listed configurations in its current state, but there are some things that can be adjusted to further lower it. The most time saving fix would likely be to compile the Simulink model and run it in either “*accelerator”* or “*rapid accelerator”* mode, but this was not investigated due to the time constraint of the thesis. These modes should only be used when the models are completely finished. This is not something that was considered since there were still possibilities in tweaking the models first in order to optimize. It is possible to do analysis while the simulations are ongoing since results are stored to the hard drive for the completion of each simulation loop (depends on simulation settings).

4.2 Results

In order to analyze results some “*analyzer*” functions were created:

- “*combineSegments.m*”; During the Monte Carlo simulation, results are as mentioned regularly stored as data packets, or “*segments”*, (*.mat* files) in order to prevent loss of results if a crash occurs or other reasons. For a vast amount of simulations it is more practical to combine the segments to larger files, which also lowers the processing time required to search through the data.
- “*analyzer_lookupResults.m*”; This function is a practical way to search through specified configurations of technical and operational states. For example one can search for all com-

binations of stick band, and require all other parameters to be set to a certain value among the ones used for the Monte Carlo simulations.

- "*analyzer_plotResponses.m*"; To verify that signals are behaving as expected during simulations there was need for a tool to quickly visualize signal behaviors from different simulation measurements. Such signals that are constantly logged are *Op*, *Pv* and *Sp-Pv* deviation etc.
- "*analyzer_plotFootprintBode.m*"; This function visualizes the data obtained from the relaysweep with hysteresis for gain, and post FFT of *Op* and *Pv* for phase lag. This is the most revealing visualization for seeing how different operational and technical states affect the process of interest. Function also produces a 3D plot with frequency, gain and phase along the x-,y- and z- axes respectively which simplifies seeing trends in the "*footprints*".
- "*analyzer_displayResultsAsText.m*"; This is a fast way to look up specified parameters and results for the modified relay test. Indicator values can be read from this text display in the Matlab command window.
- "analyzer_plotIndicatorXY"; Plots gain- and phase margins along the x- and y axes respectively. These margins are obtained from the test mechanism and plotting them in this way simplifies seeing trends by adjusting single or combinations of parameters.

4.2.1 Simulation example: Zero fault/"commissioning state"

Parameters	**Values**	**Units**
Reference	0.7	[m]
Oil flow in	0.05	$[\frac{m^3}{s}]$
Pressure CD2001	19	[barG]
Pressure CD2002	16	[barG]
Diaphragm leak	0	[%]
Internal leak	0	[%]
Upper saturation	100	[%]
Lower saturation	0	[%]
Time constant	10	[S]

Stick band	0	[%]
Slip jump	0	[%]
Filter time constant	0	$[\frac{1}{w_f}]$
Bias	0	[m]
Skew	1	[factor]

Table 5: Default configuration of all operational and technical parameters. This corresponds to the “commissioning state”.

Indicators	**Values (test mechanism)**	**Values (footprint)**	**Units**
Wc	0.0098	$\lvert\omega_c(0.0098\rvert = -3dB$	Hz
W180	0.0638	0.0638	Hz
Ampl. margin	30.84	26.82	dB
Phase margin	28.39	47.84	°

Table 6: Indicators obtained from the test mechanism and from the “footprint”. There is some differences in the margins due to wc being slightly improperly estimated. This could be improved on in the final test mechanism by e.g. letting the test mechanism have a feedback that adjusts the hysteresis and thus close in on the exact frequence.

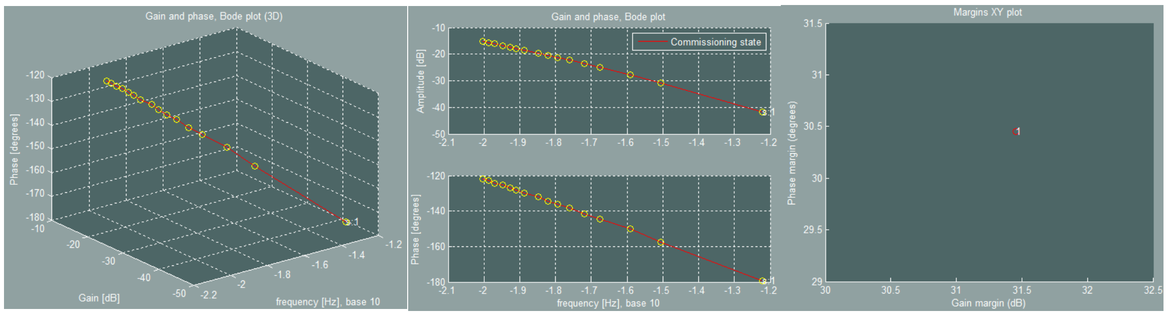


Figure 4-1: Gain and phase plots show the response of the process, while the margins XY plot is derived from the test mechanism. A similar plot can be created for the “footprint”.
Leftmost: 3D plot with XYZ corresponding to frequency, gain and phase respectively.
Middle: Standard Bode plot with gain and phase versus frequency.
Rightmost: Margins XY plot with respectively gain and phase margins along the axis.

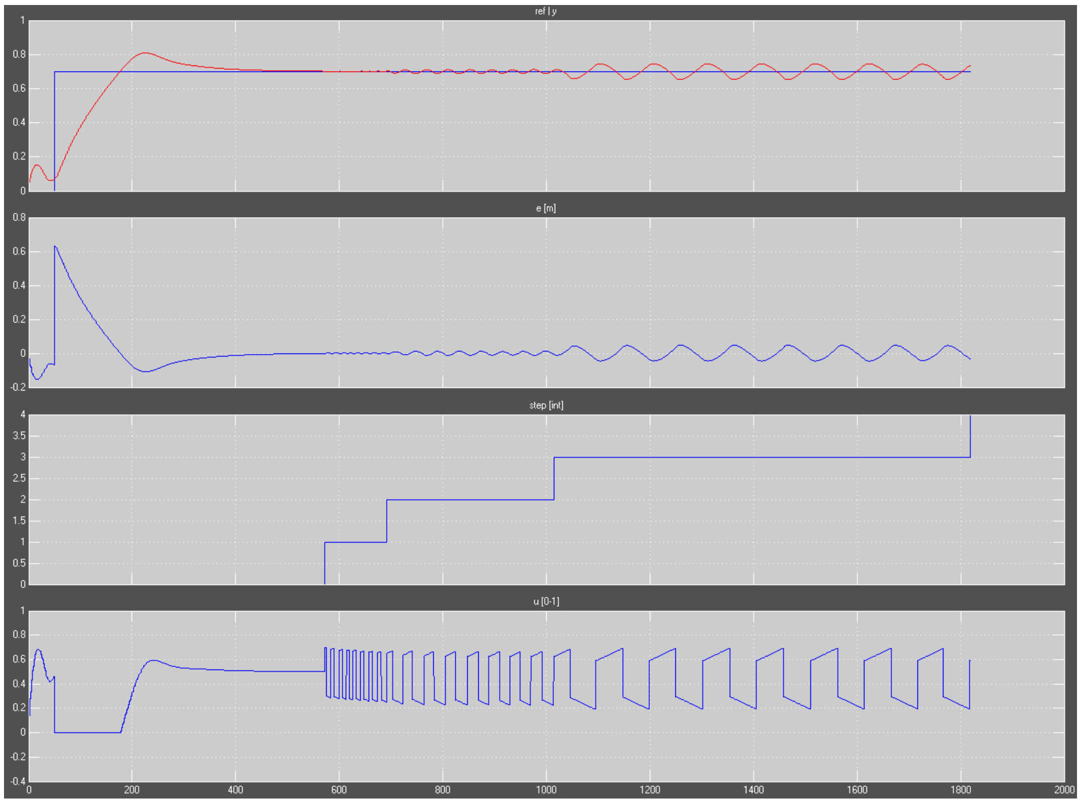


Figure 4-2: Response from Simulink scope for a typical test mechanism run. In this case, the zero fault run is displayed.
#1 from top: Pv (red) vs Sp (blue)
#2 from top: Pv-Sp deviation
#3 from top: Test mechanism status.
#4 from top: Op response. Notice the relay in series with the controller effect.

4.2.2 (Small) Simulation example: 0-20% stiction, 5% increments

Config.- index	**S [%]**	**J [%]**	**Wc [hz]**	**W180 [hz]**	**Gain. - margin, ΔdB**	**Phase - margin, $\Delta°$**
1	0	0	0.0096	0.0638	31.46	37.39
2	5	0	0.0076	0.0268	18.17	31.02
3	5	5	0.0093	0.0435	22.26	33.67
4	10	0	0.0060	0.0168	13.66	27.12
5	10	5	0.0073	0.0226	15.13	32.51
6	10	10	0.0085	0.0273	14.07	35.65
7	15	0	0.0045	0.0109	10.85	19.97
8	15	5	0.0060	0.0153	11.76	25.50

9	15	10	0.0068	0.0173	10.00	33.12
10	15	15	0.0086	0.0190	8.34	29.84
11	20	0	Unstable	-	-	-
12	20	5	0.0045	0.0097	8.46	17.57
13	20	10	0.0059	0.0116	6.80	21.75
14	20	15	Unstable	-	-	-
15	20	20	Unstable	-	-	-

Table 7

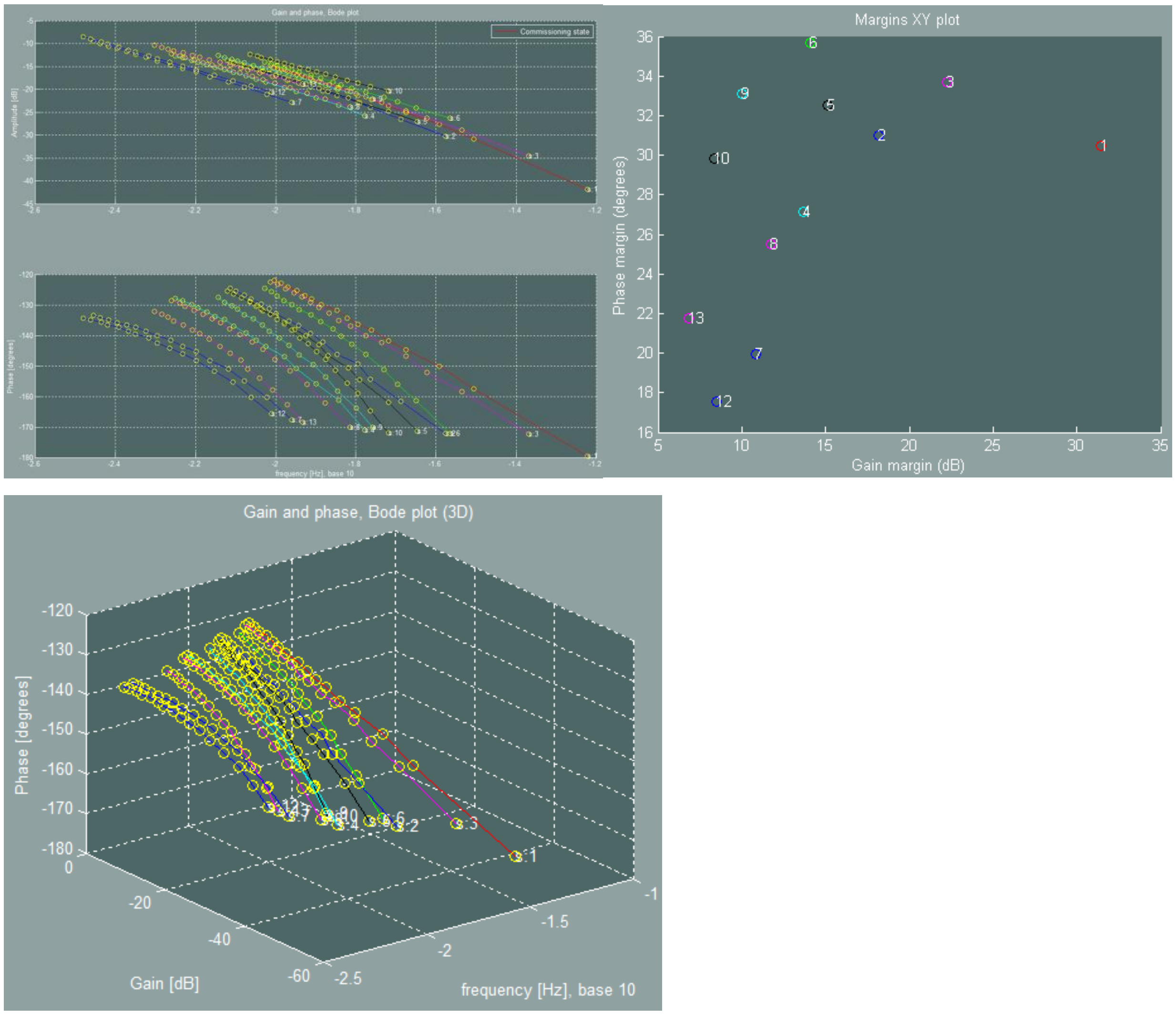


Figure 4-3:
Upper left: Process footprint for stick band, S, and slip jump, J,each being adjusted for 0 to 20%, with 5% increments.
Upper right: Margins XY plot for 0-20% stiction with 5% increments.
Lower: 3D representation

4.2.3 (Big) Simulation example: Stiction configurations

The simulations were run only for J<=S.

Config.-index	**S [%]**	**J [%]**	**Wc [hz]**	**W180 [hz]**	**Gain. -margin, ΔdB**	**Phase -margin, $\Delta°$**
1	0	0	0.0096	0.0638	31.46	37.39
2	10	0	0.0060	0.0168	13.66	27.12
3	10	10	0.0085	0.0273	14	35.65
4	20	0	Unstable	-	-	-
5	20	10	0.0059	0.0116	6.81	21.75
6	20	20	Unstable	-	-	-

Table 8

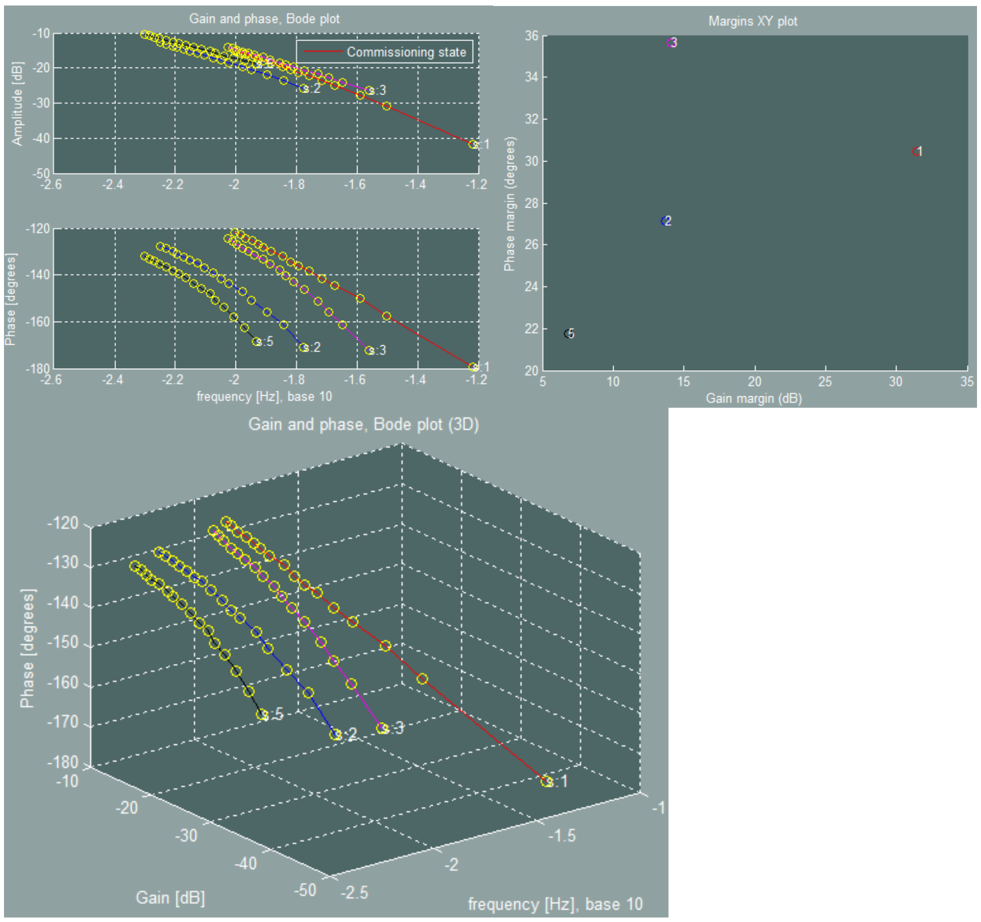
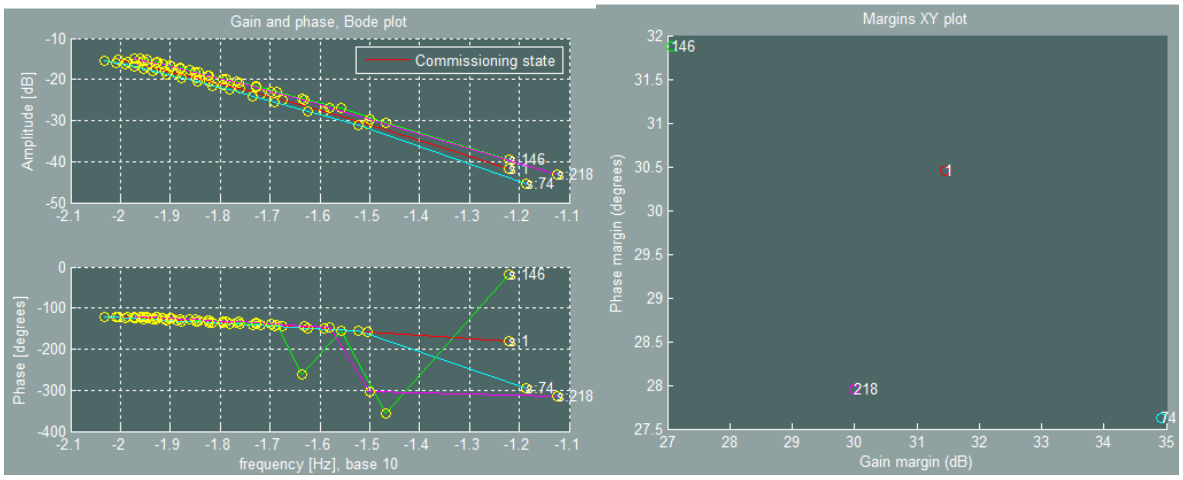


Figure 4-4: Combinations of stiction for the first (big) simulation loops.
The "footprints" are numbered for convenience.

4.2.4 (Big) Simulation example: Different pressures (operational states)

Config.- index	**CD2001 P [barG]**	**CD2002 P[barG]**	**Wc [hz]**	**W180 [hz]**	**Gain. - margin, ΔdB**	**Phase - margin, Δ°**
1	19	16	0.0096	0.0638	30.84	28.39
74	19	18	0.0098	0.0698	34.90	27.62
146	27	16	0.0106	0.0577	27.07	31.88
218	27	18	0.0112	0.0714	29.99	27.95

Table 9:



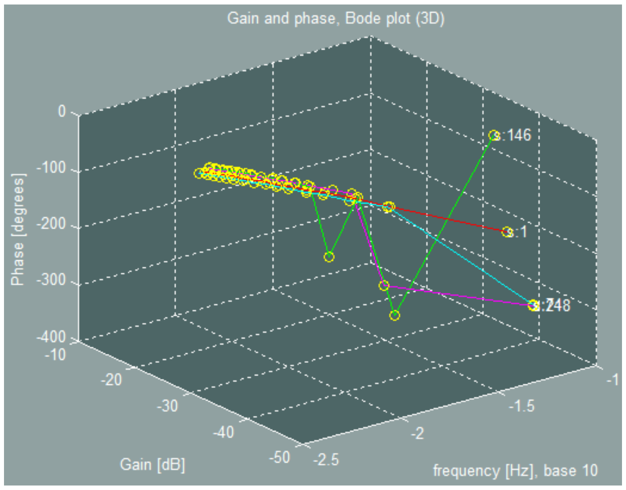
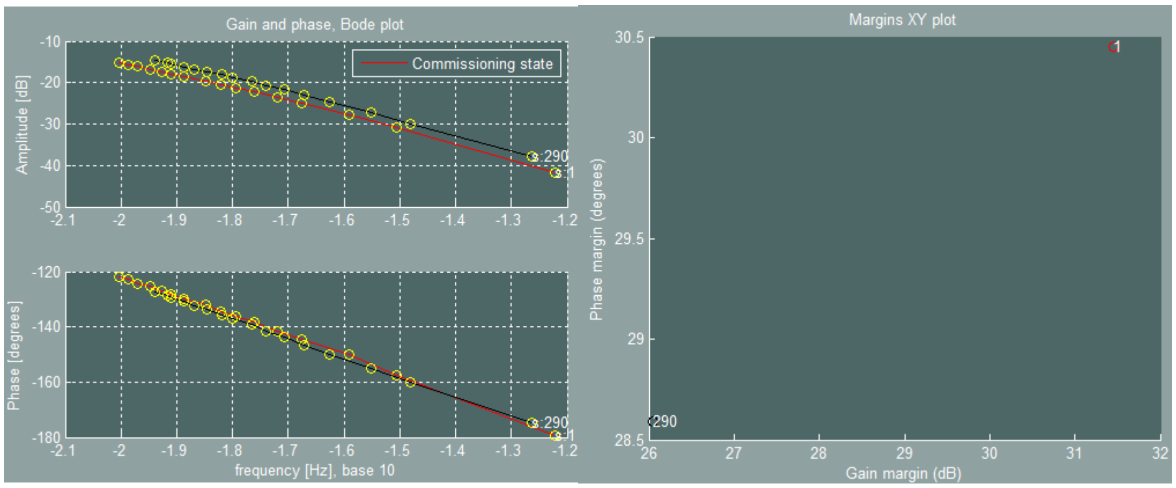


Figure 4-5:
Upper left: Gain and phase plot. The FFT misread the phase for some of the sample points for the "footprint" for these configurations.
Upper right: Margins XY plot.
Lower: 3D representation.

4.2.5 (Big) Simulation example: Different oil flow rates (operational states)

Config.- index	**Oil flow** $[\frac{m^3}{s}]$	**Wc [hz]**	**W180 [hz]**	**Gain. - margin, ΔdB**	**Phase - margin, Δ°**
1	0.05	0.0096	0.0638	31.46	37.39
290	0.07	0.0119	0.0526	26.04	28.59

Table 10



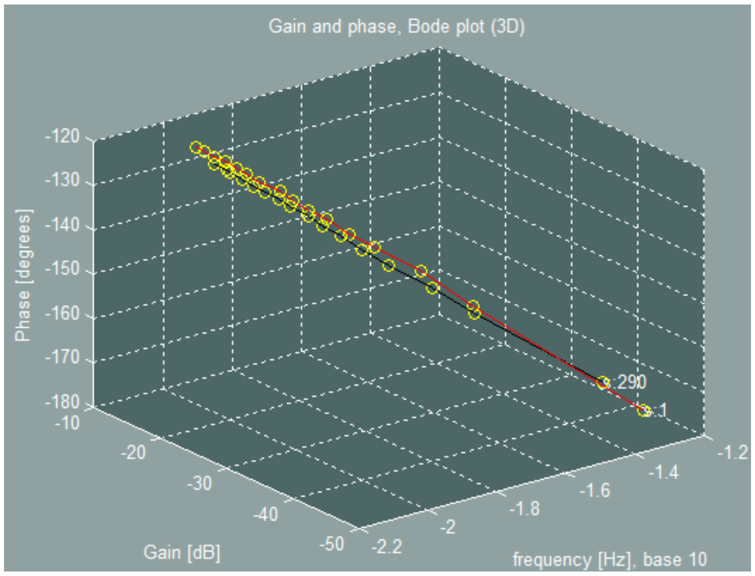


Figure 4-6:

4.2.6 (Big) Simulation example: Diaphragm leak and internleak

Config.- index	Diaphragm leak	Intern leak	Wc [hz]	W180 [hz]	Gain. - margin, ΔdB	Phase - margin, Δ°
1	0	0	0.0096	0.0638	31.46	37.39
13	0	10	0.0060	0.0612	33.12	45.29
25	0	20	*DNF*	-	-	-
37	10	0	0.0097	0.0625	30.06	30.52
49	10	10	0.0058	00625	33.60	36.66

Table 11

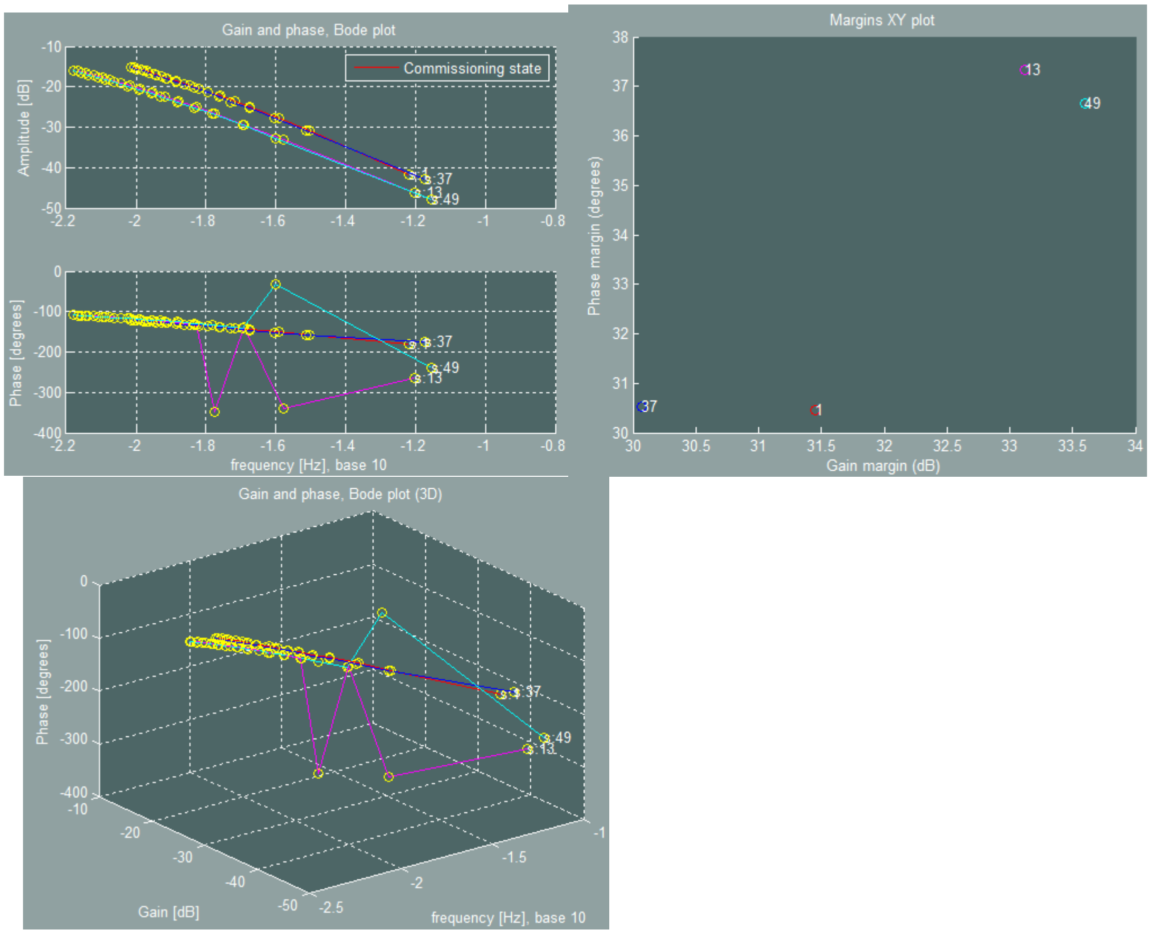
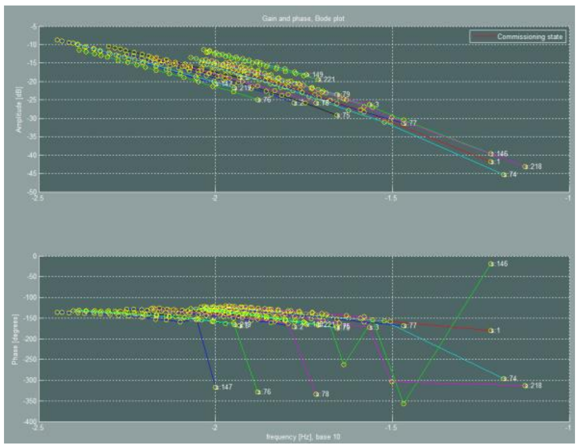
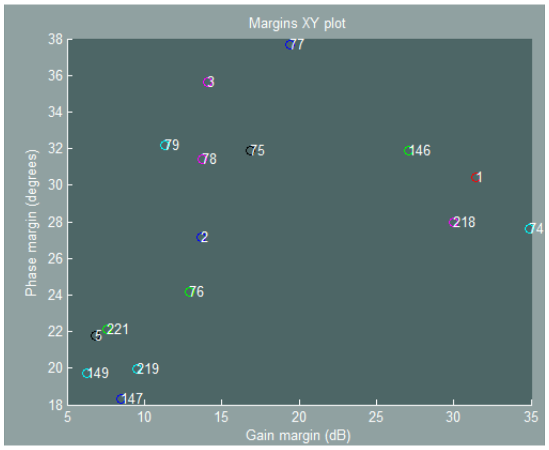


Figure 4-7: The FFT misread some of the "footprint" phase samples.

4.2.7 (Big) Simulation example: Stiction (technical) and different pressure (operational)

This produced a lot of results (24) and weren't tabulated for this section.


Gain and phase, Bode plot
Commissioning state
Amplitude [dB]
Phase [degrees]
frequency [Hz], base 10


Margins XY plot
Phase margin (degrees)
Gain margin (dB)
77
3
79
78
75
146
1
218
74
2
76
221
5
149
219
147

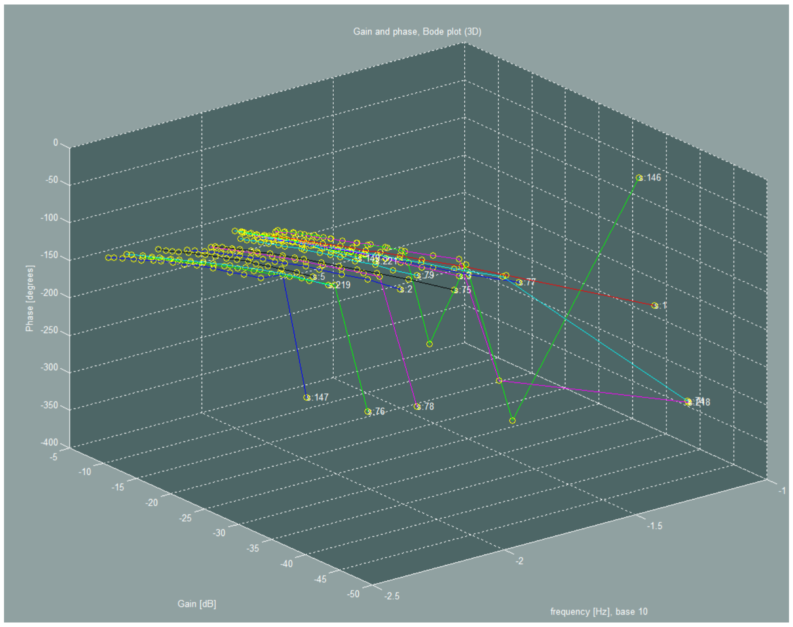

Gain and phase, Bode plot (3D)
Phase [degrees]
Gain [dB]
frequency [Hz], base 10

Figure 4-8

Chapter 5
Conclusion

The obtained results show great promise of being able to separate theeffects of the different operational and technical states. None of the observed results have been identical in any configuration, which indicates that the configurations are separable. For all observed cases of increased technical state parameters (faults etc.), the frequency of ω_{180} and ω_c go down. This behavior is as expected, since the bandwidth, and margins, are in practice reduced during operation of the process plant.

In the case of valve stiction, observed tendencies are that the frequency of ω_c goes down, but at a slower rate than that of ω_{-180°. This indicates that $\Delta\omega_{c,-180^\circ}$ is reduced. Additionally, when the stiction consists purely of slip jump, or if slip jump is gradually applied to the valve, the gain- and phase margins are increased. The XY- plots are good for looking at the trajectories of margins for the increase of the technical degradation and change in operational state. For the valve stiction, the gain margins are reduced, while the phase margins can both increase or decrease depending on the specific composition of a certain stiction. General tendencies are that the margins change a lot for small increments of the operational and technical parameters. This is very desirable since the test mechanism will be more sensitive toward detecting small changes. This also suggests that the test mechanism should run as often as possible (without adding any cost or lowered performance).

In the case of the operational state configurations, it seems that the “footprints” mainly parallel shift, such as an increase of process gain if the differential pressure across the valve is increased as demonstrated by #218 in Figure 4-5. The change of the inlet oil flow acts in the same manner.

The footprints’ resolutions could be increased for later simulations to increase accuracy and to open up for some error removal mechanics for samples that are known to be wrong. The FFT estimation technique for process phase lag sometimes fails, and there is need to go some in depth of this to see why, and how to robustify it (The major cause is assumed to be handling of phase results according to the crossover of unit cycle quadrants).

The developed tools prove reliable and very helpful in the understanding of the effects of developed faults and change of operational states. This will be used to develop the limits required to separate the presence of multiple technical states and varying operational states from each other, and to further simplify process plant control and controller retuning. The completed Monte Carlo simulation setup is handed over to Statoil, and the supervisor receives extensive training required to use the developed tools.

Chapter 6
Bibliography

[1] Statoil, Control Invention Prestudy 2014, Idea reference:K4064.

[2] Espen Svandalsflona, Frode Tuen, Condition Monitoring of Control Loops, University of Stavanger, Spring 2010.

[3] Horch, Alexander, Condition Monitoring of Control Loops, Stockholm: School of Electrical Engineering, Royal Institute of Technology, 2000.

[4] Statoil, Råoljeproduksjon, DOCID=1017540.

[5] Haugen, Finn, Regulering av dynamiske systemer, Tapir forlag, 1994.

[6] Shoukat M. A. A. Choudhury, Sirish L. Shah, Nina F. Thornhill, Diagnosis of Process Nonlinearities and Valve Stiction, Berlin: Springer, 2008.

[7] E. P. Management, Fisher 667 Diaphragm Actuator Sizes 30-76 and 87, http://www.documentation.emersonprocess.com/groups/public/documents/instruction_manuals/d100310x012.pdf, 2013.

[8] Erin Knight, Matthew Russell, Dipti Sawalka, Spencer Yendell, Valve Modeling, 2006.

[9] Mean time between failures, Wikipedia, the free encyclopedia, 2010.

[10] Wikipedia, the free encyclopedia, "Reliability centered maintenance," http://en.wikipedia.org/wiki/Reliability_centered_maintenance, 2014.

[11] R. I. G. Mohamed A. Sharif, Fault Diagnosis in Industrial Control Valves and Actuators, Minnesota, USA: IEEE Instrumentation and Measurement, 1998.

[12] Sanders, D., Control-valve seat leakage, Atlanta, Georgia: http://www.hydrocarbonprocessing.com/Article/2880440/Control-valve-seat-leakage.html, 2011.

[13] Rosemount , Rosemount 3051 Pressure Transmitter, 2014: http://www2.emersonprocess.com/siteadmincenter/PM%20Rosemount%20Documents/00813-0100-4001.pdf.

[14] Haugen, Finn, Anvendt reguleringsteknikk, Trondheim: Tapir forlag, 1990.

[15] Hägglund, K. J. Åström and T., Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins, Great Britain: Pregamon Press Ltd., 1984.

[16] Zhu, Ming-Da Ma & Xin-Jian, Performance Assessment and Controller Design based on Modified Relay Feedback, Shanghai: Shanghai Jiaotong University, 2005.

[17] Team, MathWorks Support, How do I calculate the amplitude ratio and phase lag for two sinusoidal signals in MATLAB?, Forum post: http://www.mathworks.com/matlabcentral/answers/91647-how-do-i-calculate-the-amplitude-ratio-and-phase-lag-for-two-sinusoidal-signals-in-matlab, 2013.

[18] Statoil, *Process general arrangement for inlet separator CD2001,* TIPS: CP-C05-GD-047.001.

[19] P. K. Dasg, R. K. Jena, G. Panda, and Aurobinda Routray, An Extended Complex kalman Filter for Frequency Measurement of Distorted Signals, IEEE transaction on instrumentation and measurement, vol 49, 2000.

[20] O. A. Olsen, Instrumenteringsteknikk, Trondheim: Tapir akademisk forlag, 6. opplag, 2005.

[21] Statoil, Gass rekompresjonstoget, DOCID=1017924.

[22] Christopher B. Lynch, Control Loop Performance Monitoring, Vancouver, Canada: The University of British Columbia, 1992.

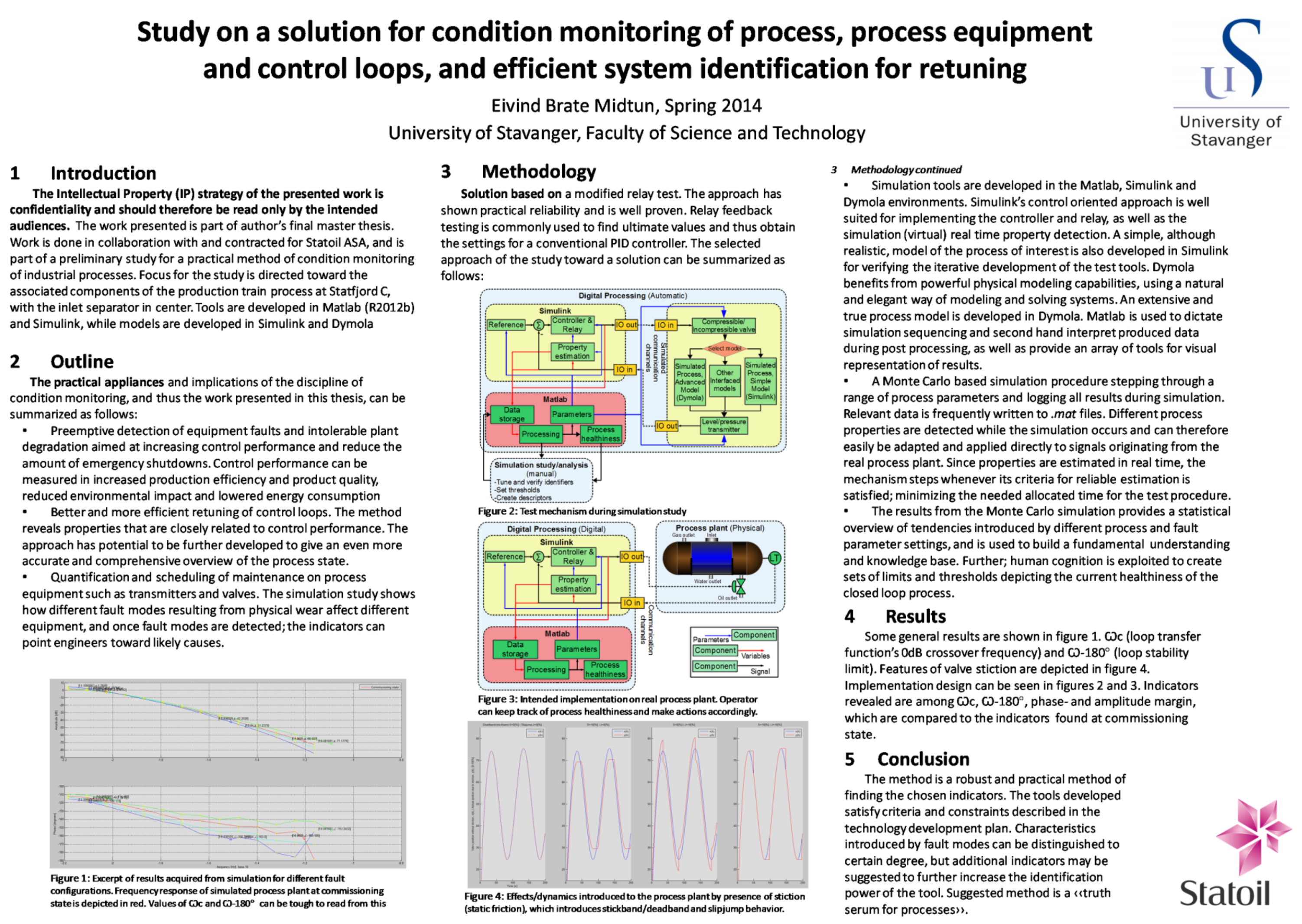
[23] Fritzson, Peter, "Introduction to Modelica," 2001.

[24] Wikipedia, the free encyclopedia, "Statfjord oil field," http://en.wikipedia.org/wiki/Statfjord_oil_field, 2014.

Chapter 1

Appendix

1.1 (Early) Poster presentation of Thesis



Study on a solution for condition monitoring of process, process equipment and control loops, and efficient system identification for retuning

Eivind Brate Midtun, Spring 2014

University of Stavanger, Faculty of Science and Technology

University of Stavanger

1 Introduction

The Intellectual Property (IP) strategy of the presented work is confidentiality and should therefore be read only by the intended audiences. The work presented is part of author's final master thesis. Work is done in collaboration with and contracted for Statoil ASA, and is part of a preliminary study for a practical method of condition monitoring of industrial processes. Focus for the study is directed toward the associated components of the production train process at Statfjord C, with the inlet separator in center. Tools are developed in Matlab (R2012b) and Simulink, while models are developed in Simulink and Dymola

2 Outline

The practical appliances and implications of the discipline of condition monitoring, and thus the work presented in this thesis, can be summarized as follows:

- Preemptive detection of equipment faults and intolerable plant degradation aimed at increasing control performance and reduce the amount of emergency shutdowns. Control performance can be measured in increased production efficiency and product quality, reduced environmental impact and lowered energy consumption
- Better and more efficient retuning of control loops. The method reveals properties that are closely related to control performance. The approach has potential to be further developed to give an even more accurate and comprehensive overview of the process state.
- Quantification and scheduling of maintenance on process equipment such as transmitters and valves. The simulation study shows how different fault modes resulting from physical wear affect different equipment, and once fault modes are detected; the indicators can point engineers toward likely causes.

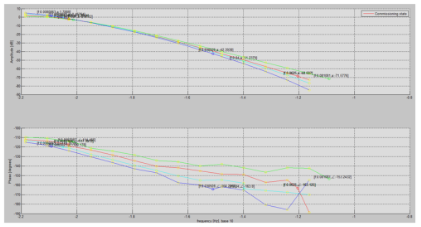


Figure 1: Excerpt of results acquired from simulation for different fault configurations. Frequency response of simulated process plant at commissioning state is depicted in red. Values of ωc and ω-180° can be tough to read from this

3 Methodology

Solution based on a modified relay test. The approach has shown practical reliability and is well proven. Relay feedback testing is commonly used to find ultimate values and thus obtain the settings for a conventional PID controller. The selected approach of the study toward a solution can be summarized as follows:

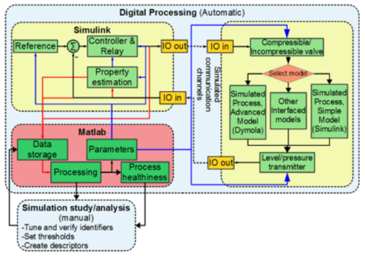

Digital Processing (Automatic)
Simulink
Reference
Controller & Relay
Property estimation
IO out
IO in
Simulated communication channels
Compressible/ Incompressible valve
Select model
Simulated Process, Advanced Model (Dymola)
Other interfaced models
Simulated Process, Simple Model (Simulink)
Level/pressure transmitter
Matlab
Data storage
Parameters
Processing
Process healthiness
Simulation study/analysis (manual)
-Tune and verify identifiers
-Set thresholds
-Create descriptors

Figure 2: Test mechanism during simulation study

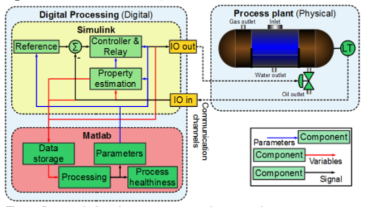


Figure 3: Intended implementation on real process plant. Operator can keep track of process healthiness and make actions accordingly.

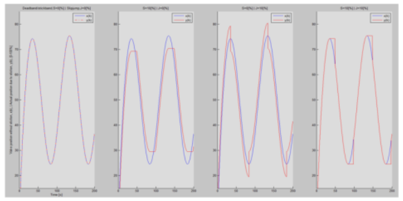


Figure 4: Effects/dynamics introduced to the process plant by presence of stiction (static friction), which introduces stickband/deadband and slipjump behavior.

3 Methodology continued

- Simulation tools are developed in the Matlab, Simulink and Dymola environments. Simulink's control oriented approach is well suited for implementing the controller and relay, as well as the simulation (virtual) real time property detection. A simple, although realistic, model of the process of interest is also developed in Simulink for verifying the iterative development of the test tools. Dymola benefits from powerful physical modeling capabilities, using a natural and elegant way of modeling and solving systems. An extensive and true process model is developed in Dymola. Matlab is used to dictate simulation sequencing and second hand interpret produced data during post processing, as well as provide an array of tools for visual representation of results.
- A Monte Carlo based simulation procedure stepping through a range of process parameters and logging all results during simulation. Relevant data is frequently written to *.mat* files. Different process properties are detected while the simulation occurs and can therefore easily be adapted and applied directly to signals originating from the real process plant. Since properties are estimated in real time, the mechanism steps whenever its criteria for reliable estimation is satisfied; minimizing the needed allocated time for the test procedure.
- The results from the Monte Carlo simulation provides a statistical overview of tendencies introduced by different process and fault parameter settings, and is used to build a fundamental understanding and knowledge base. Further; human cognition is exploited to create sets of limits and thresholds depicting the current healthiness of the closed loop process.

4 Results

Some general results are shown in figure 1. ωc (loop transfer function's 0dB crossover frequency) and ω-180° (loop stability limit). Features of valve stiction are depicted in figure 4. Implementation design can be seen in figures 2 and 3. Indicators revealed are among ωc, ω-180°, phase- and amplitude margin, which are compared to the indicators found at commissioning state.

5 Conclusion

The method is a robust and practical method of finding the chosen indicators. The tools developed satisfy criteria and constraints described in the technology development plan. Characteristics introduced by fault modes can be distinguished to certain degree, but additional indicators may be suggested to further increase the identification power of the tool. Suggested method is a ‹‹truth serum for processes››.

Statoil

1.2 List of figures

1.3 Main( )

1.3.1 “run_MainSimulation.m”

```
%   ***MainSimulation governs the entire simulation procedure***
%----------------------------------------------------------------
% Description and comments:
%   Script is divided into three components. Simulation state object is
%   automaticly stored in the outer workspace making attempts at clustering
%   relevant code together in functions hard as long as the 'sim()' is
%   located in the function.
%----------------------------------------------------------------
% DO:
%   -Add simulation loop index to each stored data segment.
%   -Add the stopcause and store in data segment
%   -Store the test matrix in the segment folder
%   -Save controller setup

%% Program initialization
if exist('TEMP.progressbar_handle','var'), if ishandle(TEMP.progressbar_handle),
close(TEMP.progressbar_handle); end, end % Prevent multiple progress bars
% clear all;
close all; clc;
warning('off','Simulink:Engine:SimStateParameterChecksumMisMatch') % turns off
unnecessary warning

% Directories
addpath('Program functions')     % essential
addpath('S-Functions')            % essential
addpath('Init files')            % essential
addpath('Simulink models')       % essential
addpath('Analysis functions')
addpath('Script components')     % essential (will be gradually converted to func-
tions)
if ~(exist('Simulation results','dir')==7), mkdir('Simulation results'); end
addpath('Simulation results')    % essential for automatic procedure
if ~(exist('Simulation restore files','dir')==7), mkdir('Simulation restore
files'); end

%% Initialize model
disp('Started..')
run('init_default_values.m');
run('prep_simulation_parameters.m');    % simulation adjustable parameters
run('init_other_parameters.m');         %
run('init_CD2001.m');                   % matlab init file
run('init_valve_complete.m');           % lv20024A
run('init_transmitter.m');              % init before controller
run('init_contr_and_rel.m');            %

openModels = find_system('SearchDepth', 0); % get current open models
if ~ismember(USERPARAM.mdl,openModels) % check if selected model is open, else
open
    statusdisp(['Opening model ',USERPARAM.mdl,'...'],2)
    open(USERPARAM.mdl);
    % remove old open scopes for cleanliness
    shh = get(0,'ShowHiddenHandles');
    set(0,'ShowHiddenHandles','On');
    hscope = findobj(0,'Type','Figure','Tag','SIMULINK_SIMSCOPE_FIGURE');
    close(hscope);
    set(0,'ShowHiddenHandles',shh);
    clear('shh','hscope');
    statusdisp('Done',2)
end
set_param(USERPARAM.mdl, 'LoadInitialState', 'off'); % prevents loading at first
run
clear('openModels')

% Pause and ask for permission to continue
dlgansw = questdlg('Continue processing (Automatic)?','Initialization complet-
ed','Yes','No','Yes');
if  strcmp(dlgansw,'No')
    clear('dlgansw')
    return
else clear('dlgansw')
end

%% Simulation
component_simloops2; %( DATA,MODEL,USERPARAM );

%% Summary
if exist('TEMP.progressbar_handle','var'),close(TEMP.progressbar_handle), end

if USERPARAM.mail.notify_by_mail && ~(isempty(USERPARAM.mail.address) || is-
empty(USERPARAM.mail.pw))
    message = 'Processing complete';
    notifyEmail( message,USERPARAM.mail.address,USERPARAM.mail.pw )
end
```

1.4 Init files

1.4.1 “prep_simulation_parameters.m”

```
%%  INFOPANEL
%   unique() makes sure each vector is optimized by removing repeating values
%   linspace(a,b,c) creates c values equally spaced between a and b

%% Global
global GLOBALDATA
GLOBALDATA.display_priority_requirement = inf;   % default inf -> displays all
messages. 1 is highest priority normal messages, 0 is error.

%% Simulation setup
USERPARAM.simulation_timeout = 20000;
USERPARAM.mdl                = 'CD2001_model_v4'; % simulink model
USERPARAM.system             = 1;                 % 1:simple, 2:advanced,
3:fictional
USERPARAM.include_sinesweep = 0;                 % default 1
USERPARAM.include_relaysweep = 1;                % default 1
USERPARAM.include_transportdelaysweep = 0;      % default 0, not necessary with
post fft for phase
USERPARAM.save_results      = 1;                % default 1
USERPARAM.prevent_load_states = 0;              % default 0
USERPARAM.START_AT_IDX = 49;                    % default 1

USERPARAM.ERROR_LIMIT              = 1; % default 1. Amount of errors before ter-
minating
USERPARAM.perform_save_frequency  = 0; % default 0 [min], save after every run
USERPARAM.save_minimum_loops      = 1; % default 1. Least amount of loops before
a save

% does currently not work on the Statoil network
USERPARAM.mail.notify_by_mail = 0; % send default message that the processing is
completed to selected recipient upon simulation completion
USERPARAM.mail.address       = ''; % mail client address. Gmail is accepted
USERPARAM.mail.pw            = ''; % client password

%% Operational states
DATA.oper.reference              = unique(linspace(0.7,1,2));
DATA.oper.oilflow_in             = [0.05 0.07];
DATA.oper.cd2001_P               = [19 27];
DATA.oper.cd2002_P               = [16 18];

%% Valve
% Relevant values in [%]
DATA.valve.diaphragmleak    = unique(linspace(0,10,2));     % default lin-
space(0,0,1) % partly blockage of air intake/supply
DATA.valve.internleak       = unique(linspace(0,20,3));     % default lin-
space(0,0,1) % internal leakage / bypass
DATA.valve.uppersaturation  = unique(linspace(100,100,1)); % default lin-
space(100,0,1)
DATA.valve.lowersaturation  = unique(linspace(0,0,1));     % default lin-
space(0,0,1)
DATA.valve.T                = unique(linspace(10,15,2));   % default lin-
space(10,10,1) % Too low value will cause problems in valve model
DATA.valve.S                = unique(linspace(0,20,3));     % default lin-
space(0,0,1) % Stiction
DATA.valve.J                = unique(linspace(0,20,3));     % default lin-
space(0,0,1) % Slipjump

%% Transmitter
DATA.transmitter.Tm         = [0 20 60]; % default 0
DATA.transmitter.bias       = [0 0.05]; % [m] b, default 0
DATA.transmitter.skew       = [1 0.9 1.1]; % [unit] a, default 0, y = ax + b

%% Sine sweep (only applicable if include_sinesweep==1)
DATA.sinesweep.amplitude    = 0.1;  % [m]
DATA.sinesweep.resolution   = 16;   % number of sine sweeps performed for verifi-
cation, wc and w180 is automaticly added
DATA.sinesweep.wcfactor     = 0.7;  % lowest frequency of sine sweep equals 0.7 *
wc
DATA.sinesweep.w180factor   = 1.1;  % highest frequency of sine sweep equals 1.1
* w180
```

```
%% Summary
statusdisp(['simulating system : ',num2str(USERPARAM.system)])
statusdisp('initialized prep simulation parameters');
```

1.4.2 “init_CD2001.m”

```
%% CD2001
MODEL.oilflow_in = 0.05;% [m^3/s], oil in

MODEL.cd2001_rho_oil    = 836;      %[kg/m^3] Density of oil in separator
MODEL.cd2001_weirheight = 1.8;      %[m] Height of weir
MODEL.cd2001_hsl        = 1.2+1.222; %length of drum lightside of weir west side
+ east side
MODEL.cd2001_d          = 3.8;      %internal diameter of drum

MODEL.cd2001_P=19;
MODEL.cd2002_P=16;
%%
statusdisp('initialized CD2001',1);
```

1.4.3 “init_default_values.m”

```
DEFAULT = struct();

% Operational
DEFAULT.oper.refhigh = 0.7;
DEFAULT.oper.oilflow_in = 0.05;
DEFAULT.oper.cd2001_P = 19;
DEFAULT.oper.cd2002_P= 16;

% Valve
DEFAULT.valve.diaphragmleak = 0;
DEFAULT.valve.internleak = 0;
DEFAULT.valve.uppersaturation = 100;
DEFAULT.valve.lowersaturation = 0;
DEFAULT.valve.T = 10;
DEFAULT.valve.S = 0;
DEFAULT.valve.J = 0;

% Transmitter
DEFAULT.transmitter.Tm = 0;
DEFAULT.transmitter.bias = 0;
DEFAULT.transmitter.skew = 1;
DEFAULT.transmitter.filter_enabled = 1;

% Summary
disp('initialized default values');
```

1.4.4 “init_contr_and_rel.m”

```
%% Relay parameters
%relay and controller in series
MODEL.relay_d_fixed = 0.05; %  [m] was 0.1 until 24.06.14. d_fixed corresponds to
the 'e' that the controller sees. For relay in series with controller.

%% Controller parameters

%D portion- filter
w_f = 100; %filter any frequencies with higher frequence than bandwidth of our
system
MODEL.T_f = 2 * pi / w_f; % corresponding filter time constant
clear('w_f')

%Default PID params

% simple system contr param, found by testing
MODEL.controller_invert = 1;
Kp1 = 4;
Ti1 = 100;
Td1 = 0;

% fictional system contr param, found by pidtool(sys) + adjustment by trial and
error
% pidtool(fictionalsys);
Kp2 = 0.0073806; % 0.5867 from pidtool
Ki2 = 0.0063453; % 0.0034824 from pidtool
Ti2 = Kp2/Ki2; % Kp2/Ki2 if using pidtool
Td2 = 0;

MODEL.system = USERPARAM.system;
if USERPARAM.system == 1
    MODEL.Kp = Kp1;
    MODEL.Ti = Ti1;
    MODEL.Td = Td1;
elseif USERPARAM.system == 2
    MODEL.Kp = Kp2;
    MODEL.Ti = Ti2;
    MODEL.Td = Td2;
end
clear('Kp1','Ti1','Td1','Kp2','Ki2','Ti2','Td2')

num = MODEL.Kp*[MODEL.Ti*MODEL.Td MODEL.Ti 1];
den = [MODEL.Ti 0];
TEMP.R = tf(num,den);
clear('num','den')

% IO
MODEL.controller_range_low = 0;
MODEL.controller_range_high = 1;
MODEL.controller_signal_low = 4;
MODEL.controller_signal_high = 20;
MODEL.controller_resolution = 16; % [bits]
MODEL.controller_quant_interval = (MODEL.controller_range_high - MOD-
EL.controller_range_low)/(2^MODEL.controller_resolution);
MODEL.controller_IO_rate = 1; % [hz], frequency of IO update
MODEL.unit_to_mA = (MODEL.transmitter_signal_high-
MODEL.transmitter_signal_low)/...
        (MODEL.transmitter_range_high-MODEL.transmitter_range_low);
MODEL.mA_to_unit = 1 / MODEL.unit_to_mA;

%% Summary
% plots
if false
    figure()
    h = bodeplot(TEMP.R);
    p=getoptions(h);     % Create a plot options handle p.
    p.FreqUnits = 'Hz'; % Modify frequency units.
    setoptions(h,p);     % Apply plot options to the Bode plot and render.
end

% text output
str = '';
if MODEL.Kp ~= 0, str = [str,'P']; end
if MODEL.Ti < inf, str = [str,'I']; end
if MODEL.Td ~= 0, str = [str,'D']; end
if MODEL.controller_invert, statusdisp('controlled input inverted',2), end
statusdisp([str,' controller selected'],2), clear('str')
statusdisp('initialized controller and relay',1);
```

1.4.5 “init_other_parameters.m”

```
%% Comment section
%   Find unlinearity dynamics as linear dynamics in textbook and implement in
model.
%   Move nonparameter sections to functions. Generalize the input-output
%   relation algorithm.
%
%   Acquire data for 'oilflow_in'
%

%% Simulation
MODEL.simsteptime=0.01; % simulation resolution / accuracy
MODEL.simstoptime = USERPARAM.simulation_timeout; % max simulation time
MODEL.stop_at_OP = 0; % default continue
MODEL.quiescence_limit = 30; % maximum allowed time without a change of output
value

%% Reference
MODEL.refsteptime=50;%Time of step for reference (N/A for sinus)
MODEL.reflow=0;%Lowest output amplitude of reference
MODEL.refhigh = 0.7;

%% Property measurement general parameters
MODEL.property_measurement_Ts = 1; % [s] Should match transmitter rate. Sample
time of property measurement elements
% MODEL.property_measurement_Ts = MODEL.simsteptime;
MODEL.Extra_n_relay_edges = 4; % Should be 3-4. Count extra before logging values
in the simcycle operator
```

```
%% Relaysweep
MODEL.eps = 0;
MODEL.relay_Tdelay_enabled = 0; % use this to disconnect the transport delay
MODEL.relay_Tdelay = MODEL.simsteptime; % must always be > than 0, else loading
states won't work because the block changes behavior when switching from 0 to ~0
MODEL.relaysweep = 0;

%% Sinesweep
MODEL.sinesweep_enabled = 0; % [bool] default 0/false
MODEL.sinesweep_amplitude = 0.1; % [m]
MODEL.sinesweep_frequency = 0.064; % [Hz]
if USERPARAM.system == 1; MODEL.sine_phase = pi;
else MODEL.sine_phase = 0;
end

%% Filter on physical side, for sinesweep
Fs = 1/MODEL.simsteptime;
Ws = MODEL.sinesweep_frequency / (Fs/2);
[MODEL.sinesweep.num,MODEL.sinesweep.den] = createLowPass(Ws,0.2);
% fvtool(MODEL.sinesweep.num,MODEL.sinesweep.den)
clear('Fs','Ws')

%% fictional system
K = 1.2;
w0 = 1/50;
zhetta = 0.6;
MODEL.fiction_timedelay = 3;
MODEL.fiction_num = K*w0^2;
MODEL.fiction_den = [1 2*zhetta*w0 w0^2];

%% World
MODEL.g = 9.81;%gravitational constant

%% Control performance
MODEL.performance_integratefrom = MODEL.refsteptime;

%% Stationarity
MODEL.stationarity_startevalat = MODEL.refsteptime;

% MODEL.stationary_absdedt_threshold = 0.015; % (e2-e1)/(t2-t1) < tresh
% MODEL.stationary_abse_threshold = 0.02; % [m]
MODEL.stationary_absdedt_threshold = 0.2*10^-3;
MODEL.stationary_abse_threshold = 0.002;

MODEL.moving_average_length = 50;%[s] This average is downsampled by 1/Ts

%% Period prefilter
MODEL.perprefilt_smallestontime = 2; % [s] Smallest amount of time the relay must
output high for output to be caused by true zero crossing and not noise. For
equal on/off time this value equals T/2 or 2*F

%% Automaticly find T from relay
MODEL.T_n_relay_edges = 4; %Was 10, which took a lot of time for low frequencies.
MODEL.T_std_tresh_pct = 20;%[%]
```

```
%% Find Gain
%filter with passband located at frequency from relay

% Chebyshev filters
% nfilters = 100;
% Fs = 1/MODEL.property_measurement_Ts; % Sample frequence
% bwoverlap = 0.5; % Bandwidth overlapping between target frequencies
% [ MODEL.filter_ws,MODEL.numerators,MODEL.denominators ] = createBandpassFil-
ters2( nfilters,bwoverlap,Fs ); % ,'plot'
MODEL.filter_ws = 0;
MODEL.numerators = 0;
MODEL.denominators = 0;
% clear('nfilters','Fs','bwoverlap')

%Gain
MODEL.gainfilter_sampletime = MODEL.simsteptime;

%Gain2
MODEL.A_n_relay_edges = MODEL.T_n_relay_edges;
MODEL.A_std_tresh_pct = 30; %[%]

%% Simcycle operator
MODEL.simcycle_minswitchT = 30;%[s] cannot step more often than this

%% Summary
statusdisp(['T std treshold : ',num2str(MODEL.T_std_tresh_pct),', A std treshold
: ',num2str(MODEL.A_std_tresh_pct)],2)
statusdisp(['T average over : ',num2str(MODEL.T_n_relay_edges),', A average over
: ',num2str(MODEL.A_n_relay_edges)],2)
if MODEL.sinesweep_enabled, statusdisp('Sinesweep is forced on, default is off
for automatic',2); end
statusdisp('initialized ''other'' parameters',1);
```

1.4.6 “init_transmitter.m”

```
%% Measurement
%--
MODEL.transmitter_disconnected = 0; % Level measured is the exact real level
MODEL.transmitter_filter_enabled = 0; % Enable filter in measurement equipment or
not
%---

f_m = 5; % [Hz] Measure equipment bandwidth
MODEL.transmitter_bias = 0; % [m] b
MODEL.transmitter_skew = 1; % [] a, 1 = zero skew, y = ax + b
MODEL.transmitter_Tm = 1/f_m; % Corresponding time constant of 1st order filter
MODEL.transmitter_range_low = 0; % [m]
MODEL.transmitter_range_high = 1; % [m]
MODEL.transmitter_signal_low = 4; % [mA]
MODEL.transmitter_signal_high = 20; % [mA]
MODEL.transmitter_resolution = 16; % [bits]
MODEL.transmitter_quant_interval = (MODEL.transmitter_range_high-
MODEL.transmitter_range_low)/(2^MODEL.transmitter_resolution);
MODEL.transmitter_IO_rate = 1; % [hz], frequency of IO update
clear('f_m')

if ~MODEL.transmitter_disconnected
    MODEL.meter_to_mA = (MODEL.transmitter_signal_high-
MODEL.transmitter_signal_low)/...
        (MODEL.transmitter_range_high-MODEL.transmitter_range_low); % adjustment
according to transmitter
else meter_to_mA = 1;
end
MODEL.mA_to_meter = 1 / MODEL.meter_to_mA;

%% Summary
if MODEL.transmitter_disconnected, statusdisp('Measurement equipment is discon-
nected',2);
else
    statusdisp('Measurement equipment is connected',2);
    if MODEL.transmitter_filter_enabled, statusdisp('Transmitter filter is ena-
bled',2);
    else statusdisp('Transmitter filter is not enabled',2);
    end
    statusdisp('initialized level transmitter',1);
end
```

1.4.7 “init_valve_complete.m”

```
%% Complete with choudhury's:
%   Comments:
%   Unable to respond for relay test (instant steps) if 'valve_T'=0 due to
%   stiction model. Set it low rather than 0 for instant valve response.

MODEL.lv20024_Kv=565;

MODEL.valve_IO_rate = 1; % [s], frequency of IO update

MODEL.valve_diaphragmleak = 0; % [%] of lost force
MODEL.valve_T = 10; % Cannot be 0 (Unknown why...), must be very small instead
(for ideal valve!)

MODEL.valve_uppersaturation=100;
MODEL.valve_lowersaturation=0;
MODEL.valve_internleak=0;

%valve characteristics
R=20;%[20-50]
x=0:0.01:1;
%equal percentage flow characteristic
MODEL.valve_characteristiclookup=( R.^(x-1)-R^-1 ) * ( 1/(1-R^-1) )*100; % 0-100%
%plot(x,MODEL.valve_characteristiclookup);
clear('R','x')
```

```
%controlsignal convertion
MODEL.controlsignal_low = 0;
MODEL.controlsignal_high = 1;
MODEL.controloutput_to_percentage_lookup = 0:100;%control output 0-1->0-100%

MODEL.valve_S = 0;%  [%] stickband
MODEL.valve_J = 0;%   [%] slipjump

%%
statusdisp('initialized oil fluid valve');
```

1.5 Script components

1.5.1 "component_create_testmatrix.m"

```
% pre
pcm_index.transmitter_Tm = 1;
pcm_index.transmitter_bias = 2;
pcm_index.transmitter_skew = 3;
pcm_index.oper_oilflow_in = 4;
pcm_index.oper_cd2001_P = 5;
pcm_index.oper_cd2002_P = 6;
pcm_index.valve_diaphragmleak = 7;
pcm_index.valve_internleak = 8;
pcm_index.valve_uppersaturation = 9;
pcm_index.valve_lowersaturation = 10;
pcm_index.valve_T = 11;
pcm_index.valve_S = 12;
pcm_index.valve_J = 13;
parameter_configuration_matrix = zeros(1,13);
simulation_order_index = 1;
% generate test matrix
for prep_transmitter_Tm = DATA.transmitter.Tm
    for prep_transmitter_bias = DATA.transmitter.bias
        for prep_transmitter_skew = DATA.transmitter.skew
            % operational
            for prep_oper_oilflow_in = DATA.oper.oilflow_in
                for prep_oper_cd2001_P = DATA.oper.cd2001_P
                    for prep_oper_cd2002_P = DATA.oper.cd2002_P
                        % valve
                        for prep_valve_diaphragmleak = DATA.valve.diaphragmleak
                            for prep_valve_internleak = DATA.valve.internleak
                                for prep_valve_uppersaturation = DA-
TA.valve.uppersaturation
                                    for prep_valve_lowersaturation = DA-
TA.valve.lowersaturation
                                        for prep_valve_T = DATA.valve.T
                                            for prep_valve_J = DATA.valve.J
                                                for prep_valve_S = DATA.valve.S
                                                    % special case; J
                                                    % cannot be higher than
                                                    % S due to double relay
                                                    % behavior.
```

```
                                                  if ...
                                                        prep_valve_J <=
prep_valve_S &&...
                                                        prep_oper_cd2002_P <
prep_oper_cd2001_P
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.transmitter_Tm) =
prep_transmitter_Tm;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.transmitter_bias) =
prep_transmitter_bias;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.transmitter_skew) =
prep_transmitter_skew;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.oper_oilflow_in) =
prep_oper_oilflow_in;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.oper_cd2001_P) =
prep_oper_cd2001_P;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.oper_cd2002_P) =
prep_oper_cd2002_P;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_diaphragmleak) =
prep_valve_diaphragmleak;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_internleak) =
prep_valve_internleak;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_uppersaturation)
= prep_valve_uppersaturation;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_lowersaturation)
= prep_valve_lowersaturation;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_T) =
prep_valve_T;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_S) =
prep_valve_S;
                                                     parame-
ter_configuration_matrix(simulation_order_index,pcm_index.valve_J) =
prep_valve_J;
                                                     simulation_order_index =
simulation_order_index + 1;
                                                  end % end special case
                                              end % valve S
                                          end % valve J
                                      end % valve T
                                  end % valve lower saturation
                              end % valve upper saturation
                          end % valve internal leak
                      end % valve diaphragm leak
                  end % CD2002 pressure
              end % CD2001 pressure
          end % oilflow in
```

```
        end % transmitter skew
    end % transmitter bias
end % transmitter Tm
```

1.5.2 “component_simloops2.m”

```
%% 'save' ID setup
default_simulation_ID = createSimID(USERPARAM.mdl);   % create a default simula-
tion ID which should be satisfactory to distinguish between sessions
valid = 0;
string = 'Create an ID (name) for the simulation:';
while ~valid
    dlganswer = inputdlg({string},'ID dialogue',1,{default_simulation_ID}); %
prompt user
    if isempty(dlganswer), return
    elseif ~(exist(['Simulation results/',dlganswer{1}],'dir') == 7)
        TEMP.simulation_ID = dlganswer{1}; % assign user input
        valid = 1;
    else string = 'ID (name) already in use. Please choose another:';
    end
end
clear('string','valid','dlganswer','default_simulation_ID');

%% Initalizations
TEMP.segmentnum = 1;             % init
TEMP.last_save_tic = tic;        % init
TEMP.simOut = 0;

TEMP.error_count = 0;        % init
TEMP.last_param_loop_duration = nan; % init
TEMP.last_sinesweep_loop_duration = nan; % init
TEMP.time_of_start = tic;       % init
results_container = {}; % init
CURRENT_CONFIGURATION_IDX = 0;          % init
TEMP.not_saved_count = 0;

if USERPARAM.START_AT_IDX == 1
    TEMP.zero_fault_run = 1;     % init, first run with zero faults. Assuming
that the first simulation will be with default settings
else TEMP.zero_fault_run = 0;
end
% create configuration matrix
component_create_testmatrix;

%% Status
total_configurations_count =
length(DATA.oper.reference)*size(parameter_configuration_matrix,1);
% Create progress bar / update info
if exist('TEMP.progressbar_handle','var')
    if ishandle(TEMP.progressbar_handle)
        waitbar(0,TEMP.progressbar_handle,'...');
    else TEMP.progressbar_handle = waitbar(0,'...');
    end
else TEMP.progressbar_handle = waitbar(0,'...');
```

```
end

statusdisp('> > > Simulation started < < <',1);
statusdisp(['    Will process for ',num2str(total_configurations_count),' config-
uration(s)'],2);

%% Loops
% reference / operating points loop
% Note: depth of loop decides priority
current_ref_conf_idx = 0;
for prep_refhigh = DATA.oper.reference
    % reset configuration matrix pointer
    CURRENT_CONFIGURATION_IDX = USERPARAM.START_AT_IDX;
    current_ref_conf_idx = current_ref_conf_idx + 1;
    MODEL.refhigh = prep_refhigh;
    if ~USERPARAM.prevent_load_states
        set_param(USERPARAM.mdl, 'LoadInitialState', 'off'); % prevents loading
at first run
        MODEL.stop_at_OP = 1; % this will notify simulation to stop once operat-
ing point has been reached

        %save simulation state
        TEMP.op_state = [USERPARAM.mdl '_SimState']; % name of stationary state
data holder
        set_param(USERPARAM.mdl, 'SaveFinalState', 'on', 'FinalStat-
eName',TEMP.op_state,'SaveCompleteFinalSimState', 'on');

        if ishandle(TEMP.progressbar_handle)
            wait-
bar((CURRENT_CONFIGURATION_IDX*current_ref_conf_idx)/total_configurations_count,T
EMP.progressbar_handle,'Simulating to stationary response..')
        end

        TEMP.successful_completion = 0;
        try
            TEMP.simOut = sim(USERPARAM.mdl); % starts simulation
            TEMP.successful_completion = 1;
        catch err,
            TEMP.error_count = inf;
            statusdisp('Failed to reach stability and terminated..',0);
        end

        if TEMP.successful_completion == 1
            % check cause of termination
            [TEMP.checkstate,str] = checkTerminationCause( si-
mout_stopcause_OP,simout_stopcause_simcycle,...
                si-
mout_stopcause_sinesweep,simout_stopcause_drumcapacity,simout_stopcause_instabili
ty,simout_stopcause_timeoutnochange,TEMP.simOut, MODEL.simstoptime );
            statusdisp(str,2), clear str

            set_param(USERPARAM.mdl, 'SaveFinalState', 'off'); % prevent over-
writing
            if ishandle(TEMP.progressbar_handle)
```

```
                wait-
bar((CURRENT_CONFIGURATION_IDX*current_ref_conf_idx)/total_configurations_count,T
EMP.progressbar_handle,['Currently processing from stationary.. Last loop
time:',num2str(TEMP.last_param_loop_duration),'s'])
            end
        end
    else statusdisp('Simulating response (Skipping load of states)',2)
    end
    if TEMP.successful_completion
        while CURRENT_CONFIGURATION_IDX <=
size(parameter_configuration_matrix,1)...
                && TEMP.error_count < USERPARAM.ERROR_LIMIT
            % Simulations preparation sets parameters after loading stationary
states
            TEMP.successful_completion = 0;
            %% Set params
            if ~USERPARAM.prevent_load_states
                set_param(USERPARAM.mdl, 'LoadInitialState', 'on', 'Initial-
State', TEMP.op_state); % load operating point
            end
            component_simloops_setparams2; % subscript for setting prep parame-
ters after state has been loaded
            MODEL.sinesweep_enabled = 0;
            MODEL.stop_at_OP = 0;

            %% Simulation resumes (default) / starts (if load is inhibited)
            try
                statusdisp(['Simulating from stationarity
[',num2str(CURRENT_CONFIGURATION_IDX),'/',num2str(total_configurations_count),']'
],2)
                TEMP.time_of_loop_start = tic;
                TEMP.simOut = sim(USERPARAM.mdl); % start simulation
                TEMP.last_param_loop_duration =
round(toc(TEMP.time_of_loop_start));
                str = calcVirtualTimeToRealTimeFactor(
TEMP.simOut,TEMP.last_param_loop_duration );
                statusdisp(str,2);
                clear('str')
                TEMP.successful_completion = 1;
            catch err
                statusdisp('Error occured during inner loop simulation..',0);
                TEMP.error_count = TEMP.error_count + 1;
            end
            % check cause of termination
            [TEMP.checkstate,str] = checkTerminationCause( si-
mout_stopcause_OP,simout_stopcause_simcycle,...
                si-
mout_stopcause_sinesweep,simout_stopcause_drumcapacity,simout_stopcause_instabili
ty,simout_stopcause_timeoutnochange,TEMP.simOut, MODEL.simstoptime );
            statusdisp(str,2)

            %% Store results
            rundata = struct;
            rundata.stopcause = str; clear str
            if TEMP.successful_completion
                rundata.successful = 1;
            else
                rundata.successful = 0;
                rundata.error = err;
            end
            TEMP.time = round(clock);
            rundata.timestamp = TEMP.time(4:6);

            %-------------------
            % parameters
            rundata.parameters.refhigh = MODEL.refhigh;
            rundata.parameters.oilflow_in = MODEL.oilflow_in;
            rundata.parameters.cd2001_P = MODEL.cd2001_P;
            rundata.parameters.cd2002_P = MODEL.cd2002_P;

            % valve
            rundata.parameters.valve_diaphragmleak  = MODEL.valve_diaphragmleak;
            rundata.parameters.valve_internleak     = MODEL.valve_internleak;
            rundata.parameters.valve_uppersaturation = MOD-
EL.valve_uppersaturation;
            rundata.parameters.valve_lowersaturation = MOD-
EL.valve_lowersaturation;
            rundata.parameters.valve_T              = MODEL.valve_T;
            rundata.parameters.valve_S              = MODEL.valve_S;
            rundata.parameters.valve_J              = MODEL.valve_J;
            % transmitter
            rundata.parameters.transmitter_filter_enabled = MOD-
EL.transmitter_filter_enabled;
            rundata.parameters.transmitter_bias = MODEL.transmitter_bias;
            rundata.parameters.transmitter_skew = MODEL.transmitter_skew;
            rundata.parameters.transmitter_Tm   = MODEL.transmitter_Tm;

            %-------------------
            if TEMP.successful_completion
                % store signals
                rundata.results.signal.e = simout_vector_e;
                rundata.results.signal.ym = simout_vector_lowrate_ym;
                rundata.results.signal.c = simout_vector_lowrate_c;
                rundata.results.signal.ref = simout_vector_lowrate_ref;
                if TEMP.checkstate == 6 % unstable
                    rundata.stability = 0;
                    rundata.timeout = 0;
                elseif TEMP.checkstate == 7 % timeout
                    rundata.stability = 1;
                    rundata.timeout = 1;
                else
                    rundata.stability = 1;
                    rundata.timeout = 0;

                    % results
                    A_m     = simout_property_A_m.Data(end);
                    phi_m   = simout_property_phi_m.Data(end);
                    wc      = simout_property_wc.Data(end);
                    w180    = simout_property_w180.Data(end);
                    rundata.results.A_m     = A_m;
                    rundata.results.phi_m   = phi_m;
                    rundata.results.wc      = wc;
                    rundata.results.w180    = w180;
                    if TEMP.zero_fault_run % first run with process at commis-
sioning state
                        TEMP.wc_comm = wc;
                        TEMP.w180_comm = w180;
                        TEMP.wc_eps = simout_property_epswc.Data;
                        statusdisp(['Commissioning w180 : ',num2str(w180)],3)
                        statusdisp(['Commissioning wc : ',num2str(wc)],3)
                        statusdisp(['Commissioning A_m : ',num2str(A_m)],3)
                        statusdisp(['Commissioning phi_m : ',num2str(phi_m)],3)
                        rundata.results.commissioning_state = 1;
                    else rundata.results.commissioning_state = 0;
                    end
                    rundata.results.wc_commissioning = TEMP.wc_comm;
                    rundata.results.w180_commissioning = TEMP.w180_comm;

                    %% Footprint
                    % sinesweep
                    if USERPARAM.include_sinesweep && TEMP.error_count <
USERPARAM.ERROR_LIMIT
                        DATA.sinesweep.testvector = createSweepVec-
tor(TEMP.wc_comm,DATA.sinesweep.wcfactor,...
                            TEMP.w180_comm, DATA.sinesweep.w180factor, DA-
TA.sinesweep.resolution,wc,w180);
                        component_simloops_sinesweep; % enter subscript
                        if TEMP.sinesweepsuccessful
                            rundata.sinesweep.postestimation   =
TEMP.postestimation;
                            rundata.sinesweep.testfrequencies   = DA-
TA.sinesweep.testvector;
                            rundata.sinesweep.source_amplitude  = DA-
TA.sinesweep.amplitude;
                            rundata.sinesweep.signal = TEMP.signal;
                        end
                    end
                    % relaysweep
                    if USERPARAM.include_relaysweep && TEMP.error_count <
USERPARAM.ERROR_LIMIT
                        statusdisp('Mapping frequency response (Relaysweep)',2)
                        % Estimate process gain by
                        % adjusting relay hysteresis
                        component_hysteresis_relaysweep;
                        rundata.relaysweep.gain_process_vec = gain_process_vec;
                        rundata.relaysweep.w_hyst_vec_hz = w_hyst_vec_hz;
                        rundata.relaysweep.hystsignals = hystsignals;
                        % Estimate process phase by adjusting
                        % time delay
                        if USERPARAM.include_transportdelaysweep
                            component_transportdelay_relaysweep;
                            rundata.relaysweep.pha_process_vec = pha_process_vec;
                            rundata.relaysweep.w_delay_vec_hz = w_delay_vec_hz;
                            rundata.relaysweep.delaysignals = delaysignals;
                        end
                        MODEL.relaysweep = 0;
                    end
```

```
clear('simout_A_m','simout_phi_m','simout_wc','simout_w180','A_m','phi_m','wc','w
180');
                end
            end
            TEMP.zero_fault_run = 0;

            %% End of cycle
            if TEMP.successful_completion
                TEMP.not_saved_count = TEMP.not_saved_count + 1;
            end
            results_container{length(results_container)+1,1} = rundata;
            clear('rundata')
            if 60 * USERPARAM.perform_save_frequency  <
toc(TEMP.last_save_tic)...
                    && 0 < TEMP.not_saved_count
                TEMP.perform_save = true;
            else TEMP.perform_save = false;
            end
            if TEMP.perform_save && USERPARAM.save_results
                component_save;
            end
            statusdisp(['Completed simulation step,
loop:',num2str(CURRENT_CONFIGURATION_IDX+1)],2)

            % count the loop
            CURRENT_CONFIGURATION_IDX = CURRENT_CONFIGURATION_IDX + 1;
            if ishandle(TEMP.progressbar_handle)
                wait-
bar((CURRENT_CONFIGURATION_IDX*current_ref_conf_idx)/total_configurations_count,T
EMP.progressbar_handle,['Currently processing from stationary.. Last loop
time:',num2str(TEMP.last_param_loop_duration),'s'])
            end
        end
    end % WHILE configurations
end % reference
clear('prep_refhigh','prep_valve_diaphragmleak','prep_valve_internleak','prep_val
ve_uppersaturation',...

'prep_valve_lowersaturation','prep_valve_T','prep_valve_S','prep_valve_J','prep_s
inesweep_frequency')

%% Completed
statusdisp('> > > Completed all simulations < < <',1);
if 0 < TEMP.not_saved_count && USERPARAM.save_results
    component_save;
end
clear('results_container')

TEMP.total_duration = toc(TEMP.time_of_start);
waitbar(1,TEMP.progressbar_handle,['Processing completed after
',num2str(round(TEMP.total_duration)),'sec!'])
```

1.5.3 “component_simloops_setparams2.m”

```
% reset states
MODEL.eps                    = 0; % default
MODEL.relay_Tdelay_enabled   = 0; % default

% operational states
MODEL.refhigh                = prep_refhigh;
MODEL.oilflow_in             = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.oper_oilflow_in);
MODEL.cd2001_P               = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.oper_cd2001_P);
MODEL.cd2002_P               = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.oper_cd2002_P);

% valve properties
MODEL.valve_diaphragmleak    = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_diaphragmleak)
;
MODEL.valve_internleak       = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_internleak);
MODEL.valve_uppersaturation  = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_uppersaturatio
n);
MODEL.valve_lowersaturation  = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_lowersaturatio
n);
MODEL.valve_T                = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_T);
MODEL.valve_S                = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_S);
MODEL.valve_J                = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.valve_J);

% transmitter properties
if 0 < parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.transmitter_Tm)
    MODEL.transmitter_filter_enabled = 1;
    MODEL.transmitter_Tm      = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.transmitter_Tm);
else
    MODEL.transmitter_filter_enabled = 0;
    MODEL.transmitter_Tm      = 0.2; %must be ~0, else model changes and 'state'
can't be loaded
end
MODEL.transmitter_bias       = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.transmitter_bias);
MODEL.transmitter_skew       = parame-
ter_configuration_matrix(CURRENT_CONFIGURATION_IDX,pcm_index.transmitter_skew);
```

1.5.4 “component_hysteresis_relaysweep.m”

```
M = MODEL.relay_d_fixed;

hyst_reso = 16;
A_out_vec = nan(hyst_reso,1);
```

```
T_out_vec = nan(hyst_reso,1);
hystvec = linspace(0,TEMP.wc_eps,hyst_reso);
hystsignals = cell(16,1);
for i = 1:hyst_reso
    set_param(USERPARAM.mdl, 'LoadInitialState', 'on', 'InitialState',
TEMP.op_state);
    component_simloops_setparams2;
    MODEL.sinesweep_enabled = 0;
    MODEL.stop_at_OP = 0;
    MODEL.relaysweep = 1;
    MODEL.relay_Tdelay_enabled = 0;

    MODEL.eps = hystvec(i);
    statusdisp(['Relaysweep (hysteresis)
[',num2str(i),'/',num2str(hyst_reso),']'],2);
    sim(USERPARAM.mdl)
    A_out_vec(i) = simout_property_A.Data;
    T_out_vec(i) = simout_property_T.Data;

    signal = struct;
    signal.c = simout_vector_lowrate_c;
    signal.ym = simout_vector_lowrate_ym;
    hystsignals(i) = {signal};
end
w_hyst_vec_hz = 1./flipdim(T_out_vec,1);
w_hyst_vec_rad = w_hyst_vec_hz*2*pi;
a_hyst_vec = flipdim(A_out_vec,1);
gain_hyst_vec_dB = 20*log10(a_hyst_vec/(4*M/pi));
hystsignals = flipdim(hystsignals,1);

[g_r,~] = bode(TEMP.R,w_hyst_vec_rad);
gain_reg_vec_dB = 20*log10(squeeze(g_r(1,1,:)));
gain_process_vec = gain_hyst_vec_dB - gain_reg_vec_dB;
```

1.5.5 “component_sinesweep.m”

```
sw_vec_length = length(DATA.sinesweep.testvector);
% create empty containers
TEMP.postestimation.gain_vector = nan(sw_vec_length,1);
TEMP.postestimation.phase_vector = nan(sw_vec_length,1);
% output signals
TEMP.signal.e = cell(sw_vec_length,1);
TEMP.signal.highrate_y = cell(sw_vec_length,1);
TEMP.signal.lowrate_ym = cell(sw_vec_length,1);
TEMP.signal.highrate_ref = cell(sw_vec_length,1);
TEMP.signal.lowrate_ref = cell(sw_vec_length,1);
TEMP.signal.lowrate_c = cell(sw_vec_length,1);
%
TEMP.sinesweep_idx = 1;
for prep_sinesweep_frequency = DATA.sinesweep.testvector
    if  TEMP.error_count < USERPARAM.ERROR_LIMIT
        if ~USERPARAM.prevent_load_states, set_param(USERPARAM.mdl, 'LoadInitial-
State', 'on', 'InitialState', TEMP.op_state); end
        % set sinesweep parameters
        component_simloops_setparams;
        MODEL.sinesweep_frequency = prep_sinesweep_frequency;
        MODEL.sinesweep_enabled = 1;
        MODEL.stop_at_OP = 0;
        % update progress bar
        if ishandle(TEMP.progressbar_handle)
            wait-
bar(TEMP.loopcount/TEMP.total_loop_count,TEMP.progressbar_handle,['Sinesweep
[',...
                num2str(TEMP.sinesweep_idx),'/',num2str(sw_vec_length),...
                ']..  Last loop
time:',num2str(TEMP.last_sinesweep_loop_duration),'s'])
        end
        try
            TEMP.sinesweepsuccessful = false;
            % start simulation
            TEMP.time_sinesweep_loop = tic;
            TEMP.simOut = sim(USERPARAM.mdl); % start simulation
            % save signals
            TEMP.signal.e(TEMP.sinesweep_idx) = {simout_vector_e}; % unnecessary
            TEMP.signal.highrate_y(TEMP.sinesweep_idx) = {si-
mout_vector_highrate_y}; % unnecessary
            TEMP.signal.highrate_ref(TEMP.sinesweep_idx) = {si-
mout_vector_highrate_ref}; % unnecessary
            TEMP.signal.lowrate_ym(TEMP.sinesweep_idx) = {si-
mout_vector_lowrate_ym};
            TEMP.signal.lowrate_ref(TEMP.sinesweep_idx) = {si-
mout_vector_lowrate_ref};
            TEMP.signal.lowrate_c(TEMP.sinesweep_idx) = {si-
mout_vector_lowrate_c};
            % save phase and amplitude
            x = TEMP.signal.lowrate_c{TEMP.sinesweep_idx}.Data;
            y = TEMP.signal.lowrate_ym{TEMP.sinesweep_idx}.Data;
            [TEMP.gain,TEMP.phas_rad] = post_fft_estAmpPha(x,y);
            TEMP.phas_deg = TEMP.phas_rad*180/pi;
            if 0<TEMP.phas_deg, TEMP.phas_deg=TEMP.phas_deg-360; end
            TEMP.postestimation.gain_vector(TEMP.sinesweep_idx) = TEMP.gain;
            TEMP.postestimation.phase_vector(TEMP.sinesweep_idx) = TEMP.phas_deg;
            % cause of termination
            [TEMP.checkstate,str] = checkTerminationCause( si-
mout_stopcause_OP,simout_stopcause_simcycle,...
                si-
mout_stopcause_sinesweep,simout_stopcause_drumcapacity,simout_stopcause_instabili
ty,simout_stopcause_timeoutnochange,TEMP.simOut, MODEL.simstoptime );
            statusdisp([str,'[',num2str(TEMP.sinesweep_idx),']'],2), clear str
            % mark successful sinesweep (could check termination cause and
            % check correct termination state)
            TEMP.sinesweepsuccessful = true;
            TEMP.last_sinesweep_loop_duration =
round(toc(TEMP.time_sinesweep_loop));
        catch err
            statusdisp('Error occured during sinesweep simulation..',0);
            TEMP.error_count = TEMP.error_count + 1;
        end
    end
    TEMP.sinesweep_idx = TEMP.sinesweep_idx + 1; % update index
end
```

```
MODEL.sinesweep_enabled = 0; % back to default
```

1.5.6 “component_save.m”

```
% pre/create sub directory
if ~(exist(['Simulation results/',TEMP.simulation_ID],'dir')==7),
mkdir('Simulation results',TEMP.simulation_ID); end
TEMP.savefilename = ['Simulation Results\',TEMP.simulation_ID,'\results-
segment#',num2str(TEMP.segmentnum)];
% save results
save(TEMP.savefilename,'results_container');
% backup workspace(overwrite last)
save(['Simulation restore files\backup_',TEMP.simulation_ID])
% post/summary
results_container = {};
TEMP.last_save_tic = tic;
statusdisp(['Performed save operation. Saved results as : ',TEMP.savefilename],1)
TEMP.segmentnum = TEMP.segmentnum + 1;
TEMP.not_saved_count = 0;
```

1.5.7 “component_transportdelay_relaysweep.m”

```
relaysweep = 1;
lastloggedfreq = inf;

targetresolution = 16;
A_out_vec = nan(targetresolution,1); % allocation of memory
T_out_vec = nan(targetresolution,1);
T_delay_vec = nan(targetresolution,1);
delaysignals = cell(16,1);
i = 1;
MODEL.relay_Tdelay = MODEL.property_measurement_Ts;

multTdelay = 0.1; % This should be dynamicly found (different from each process
and configuration)
if TEMP.zero_fault_run
    while TEMP.wc_comm < lastloggedfreq
        set_param(USERPARAM.mdl, 'LoadInitialState', 'on', 'InitialState',
TEMP.op_state);
        component_simloops_setparams;
        MODEL.sinesweep_enabled = 0;
        MODEL.stop_at_OP = 0;
        MODEL.relaysweep = 1;
        MODEL.eps = 0;
        MODEL.relay_Tdelay_enabled = 1;

        MODEL.relay_Tdelay = MODEL.relay_Tdelay + (i-1)*multTdelay;
        statusdisp(['Relaysweep (timedelay)
[',num2str(i),'][',num2str(wc),'(wc)<',num2str(lastloggedfreq),'][',num2str(MODEL
.relay_Tdelay),']'],2);
        T_delay_vec(i) = MODEL.relay_Tdelay;
        sim(USERPARAM.mdl)
        A_out_vec(i) = simout_property_A.Data;
        T_out_vec(i) = simout_property_T.Data;
        lastloggedfreq = 1/T_out_vec(i);
        i = i+1;

        signal = struct;
        signal.c = simout_vector_lowrate_c;
        signal.c1 = simout_vector_lowrate_c1;
        signal.ym = simout_vector_lowrate_ym;
        delaysignals(i) = {signal};
    end
    TEMP.T_delay_vec = T_delay_vec;
else
    for i = 1:length(TEMP.T_delay_vec)
        set_param(USERPARAM.mdl, 'LoadInitialState', 'on', 'InitialState',
TEMP.op_state);
        component_simloops_setparams;
        MODEL.sinesweep_enabled = 0;
        MODEL.stop_at_OP = 0;
        MODEL.relaysweep = 1;
        MODEL.eps = 0;
        MODEL.relay_Tdelay_enabled = 1;

        MODEL.relay_Tdelay = TEMP.T_delay_vec(i);
        statusdisp(['Relaysweep (timedelay)
[',num2str(i),'][',num2str(wc),'(wc)<',num2str(lastloggedfreq),'][',num2str(MODEL
.relay_Tdelay),']'],2);
        T_delay_vec(i) = MODEL.relay_Tdelay;
        sim(USERPARAM.mdl)
        A_out_vec(i) = simout_property_A.Data;
        T_out_vec(i) = simout_property_T.Data;
        lastloggedfreq = 1/T_out_vec(i);

        signal = struct;
        signal.c = simout_vector_lowrate_c;
        signal.c1 = simout_vector_lowrate_c1;
        signal.ym = simout_vector_lowrate_ym;
        delaysignals(i) = {signal};
    end
end
delaysignals = flipdim(delaysignals,1);

w_delay_vec_hz = 1./flipdim(T_out_vec(~isnan(T_out_vec)),1);
w_delay_vec_rad = w_delay_vec_hz*2*pi;
a_delay_vec = flipdim(A_out_vec(~isnan(A_out_vec)),1);
pha_delay_vec = -flipdim(T_delay_vec(~isnan(T_delay_vec)),1).*w_delay_vec_hz*360;

[~,p_r] = bode(TEMP.R,w_delay_vec_rad);
pha_reg_vec = squeeze(p_r(1,1,:));
pha_process_vec = -180 - pha_reg_vec - pha_delay_vec;
```

1.6 Program functions

1.6.1 “post_fft_estAmpPha.m”

```
function [amplitude_ratio, phase_lag] = post_fft_estAmpPha(x,y)
x = x - mean(x);
y = y - mean(y);
% FFTs
X=fft(x);
Y=fft(y);
% location
[mag_x,idx_x] = max(abs(X));
[mag_y,idx_y] = max(abs(Y));
%
px = angle(X(idx_x));
py = angle(Y(idx_y));
phase_lag = py - px;
%
amplitude_ratio = mag_y/mag_x;
```

1.6.2 “checkTerminationCause.m”

```
function [checkstate,str] = checkTerminationCause( ...
    simout_stopcause_OP,simout_stopcause_simcycle,...
    simout_stopcause_sinesweep,simout_stopcause_drumcapacity,...
    simout_stopcause_instability,simout_stopcause_timeoutnochange,simOut, sim-
stoptime )

%CHECKTERMINATIONCAUSE Summary of this function goes here
%   checkstate can be used to create specific handlers for events by
%   outside program

descript_str = '<Stopcause>: ';
if simOut(end) == simstoptime
    checkstate = 1;
    cause_str = 'Simulation timed out.';
elseif simout_stopcause_simcycle.Data
    checkstate = 2;
    cause_str = 'Simulation stopped by completion of simcycle.';
elseif simout_stopcause_sinesweep.Data
    checkstate = 3;
    cause_str = 'Simulation stopped by completion of sinesweep.';
elseif simout_stopcause_drumcapacity.Data
    checkstate = 4;
    cause_str = 'Simulation stopped because separator reached capacity limit.';
elseif simout_stopcause_OP.Data
    checkstate = 5;
    cause_str = 'Simulation stopped by reaching operating point.';
elseif simout_stopcause_instability.Data
    checkstate = 6;
    cause_str = 'Simulation stopped due to asymptotic instability.';
elseif simout_stopcause_timeoutnochange.Data
    checkstate = 7;
    cause_str = 'Simulation stopped due to timeout (output did not change)';
else
    checkstate = 0;
    cause_str = 'Unknown error. Should be considered by user.';
end
str = [descript_str,cause_str];
end
```

1.6.3 “createSimID.m”

```
function [ string_out ] = createSimID( mdl )
%CREATESIMID Summary of this function goes here
%   Detailed explanation goes here

time = round(clock);
bsymb = '+'; % break symbol
msymb = '@'; % model symbol
dsymb = '-'; % date symbol
tsymb = '-'; % time symbol
datestring = [num2str(time(3)),dsymb,num2str(time(2)),dsymb,num2str(time(1))];
timestring = [num2str(time(4)),tsymb,num2str(time(5)),tsymb,num2str(time(6))];
string_out = ['Sim',msymb,mdl,bsymb,datestring,bsymb,timestring];
end
```

1.6.4 “post_find_P.m”

```
function [P_K,P_phase] = post_find_P(R,f,in_x,in_y)
f_rad = 2*pi*f;
%
excerptsize = 70; % uses last 70[%] of datasamples (needs time to settle at
first)
fac = (100-excerptsize)/100;
idx1 = max([1,fac*length(in_x)]);
idx2 = length(in_x);
elements = round(idx1:idx2);
x = in_x(elements);
y = in_y(elements);
% figure,plot(x),hold on,plot(y,'r'),hold off
%
[yx_gain,pha_rad] = post_fft_estAmpPha(x,y);
s2 = yx_gain*exp(1i*pha_rad);
%
[mag_R,pha_deg_R,~] = bode(R,f_rad); % gain and phase of R
R_jw = mag_R*exp(1i*pha_deg_R*pi/180);
%
abs_R_jw = abs(R_jw);
% abs_R_jw_dB = 20*log10(abs_R_jw);
%
```

```
% yx_gain = rt_amp; % test
P_K = abs(yx_gain) / abs((1-yx_gain)*abs_R_jw);
P_phase = angle(s2 / ((1 - s2)*R_jw))*180/pi;
end
```

1.6.5 “createSweepVector.m”

```
function [ sweepvector ] = createSweepVector(
wc_comm,wcfac,w180_comm,w180fac,res,wc,w180 )
%CREATESWEEPVECTOR Summary of this function goes here
%   Detailed explanation goes here

lin2log = @(x)log(x)./log(10);
lowfreq                  = wc_comm * wcfac;
highfreq                 = w180_comm * w180fac;
uselogspace = true; % can use linspace or logspace for vector
if uselogspace, sweepvector = logspace(lin2log(lowfreq),lin2log(highfreq),res-2);
else sweepvector              = linspace(lowfreq,highfreq,res-2); %subtracting 2
from res since wc and w180 are also added
end
sweepvector(end+1:end+2) = [wc,w180]; % add wc and w180 freq
sweepvector              = sort(unique(sweepvector)); % resort in ascending order
and remove identical entries
statusdisp(['Will sinus sweep with ',num2str(length(sweepvector)),...
    ' frequencies from ',num2str(lowfreq),' to ',num2str(highfreq)],2)
end
```

1.6.6 “calcVirtualTimeToRealTimeFactor.m”

```
function [ string ] = calcVirtualTimeToRealTimeFactor( simOut,looptime )
%CALCVIRTUALTIMETOREALTIMEFACTOR Summary of this function goes here
%   Detailed explanation goes here

factor = (simOut(end) - simOut(1)) / looptime;
string = ['Processed in average ',num2str(factor),' virtual seconds per real sec-
ond.'];
end
```

1.6.7 “postFindPhase.m”

```
function [ phase_out ] = postFindPhase( a_est,f,signal,t,addphase )
%POSTFINDPHASE Summary of this function goes here
%   Detailed explanation goes here

sign_detrend = detrend(signal);
minphase = 0;
maxphase = -359;
phavec = linspace(minphase,maxphase,360);
minsum = inf;
```

```
for phase = phavec
    cmpsin = a_est*sin(2*pi*f.*t + (addphase + phase)*pi/180);
    sumdiff = sum((cmpsin - sign_detrend).^2);
    if sumdiff < minsum
        phase_out = phase;
        minsum = sumdiff;
    end
end

if false
    figure
    plot(sign_detrend); title('signal out')
    hold on
    plot(a_est*sin(2*pi*f.*t + (addphase + phase_out)*pi/180),'r'); title('approx
out')
    hold off
end
end
```

1.7 Analysis functions

1.7.1 “analyzer_displayResultsAsText.m”

```
function [  ] = analyzer_displayResultsAsText( varargin )
%READRESULTS_DISPLAYTEXT Summary of this function goes here
%   Detailed explanation goes here
%   Outputs measured phase and amplitude margin

results_container = [];
idx_list = [];
R = [];
if nargin == 0
    try
        [results_container,idx_list] = loadResults();
    catch e
        return
    end
else
    % input
    while 0 < length(varargin)
        removeentries = 1;
        if isa(varargin{1},'char')
            switch(varargin{1})
                case 'container'
                    results_container = varargin{2};
                case 'indexes'
                    idx_list = varargin{2};
                case 'controller'
                    R = varargin{2};
                otherwise, disp('Unknown field ''',varargin{2},'''')
            end
            if removeentries, varargin(1:2) = []; end
        else
            varargin(1) = [];
            warning('Each input should have a descripting string first')
        end
    end
end

dowarn1 = true;
for i=1:length(idx_list)
    idx=idx_list(i);
    disp(['**********Configuration ',num2str(idx),'**********'])
    disp('              Parameters:           ')
    disp(results_container{idx}.parameters)
    if results_container{idx}.successful
        % test mechanism
        disp('             Results:')
        disp(results_container{idx}.results)
        % footprint
        if ~isempty(R)
            try
            % wc
            wc = results_container{idx}.relaysweep.w_hyst_vec_hz(1);
            [gain_R_wc,pha_R_wc] = bode(R,wc*2*pi);
            % gain
            gain_P_wc = results_container{idx}.relaysweep.gain_process_vec(1);
            gain_RPM_wc = 10^(gain_P_wc/20) + gain_R_wc;
            % phase
            wc_signals = results_container{idx}.relaysweep.hystsignals{1};
            [~, wc_pha_rad] =
post_fft_estAmpPha(wc_signals.c.Data,wc_signals.ym.Data);
            pha_P_wc = 180/pi*wc_pha_rad;
            if 0<pha_P_wc,pha_P_wc=pha_P_wc-360; end
            pha_RPM_wc = pha_P_wc + pha_R_wc;

            % w180
            w180 = results_container{idx}.relaysweep.w_hyst_vec_hz(end);
            [gain_R_w180,pha_R_w180] = bode(R,w180*2*pi);
            % gain
            gain_P_w180 = re-
sults_container{idx}.relaysweep.gain_process_vec(end);
            gain_RPM_w180 = 10^(gain_P_w180/20) + gain_R_w180;
            % phase
            w180_signals = results_container{idx}.relaysweep.hystsignals{end};
            [~, w180_pha_rad] =
post_fft_estAmpPha(w180_signals.c.Data,w180_signals.ym.Data);
            pha_P_w180 = 180/pi*w180_pha_rad;
            if 0<pha_P_w180,pha_P_w180=pha_P_w180-360; end
            pha_RPM_w180 = pha_P_w180 + pha_R_w180;

            fpresults.gainmargin = abs(gain_RPM_w180-gain_RPM_wc);
            fpresults.phasemargin = abs(pha_RPM_w180-pha_RPM_wc);
            disp('             Results (footprint)')
            disp(fpresults)
            catch err % likely to be missing relay sweep
            end
        elseif dowarn1
            dowarn1 = false;
            warning('Could not show results from footprint since the used control
parameter were not given as input')
        end
    else disp('             Incomplete')
    end
    disp(['-------------       End ',num2str(idx),'     -------------'])
    disp(' ')
    disp(' ')
end
end
```

1.7.2 “analyzer_lookupResults.m”

```
function analyzer_lookupResults( varargin )
%LOOKUPRESULTS Summary of this function goes here
%   Detailed explanation goes here

% initialize other
results_container = [];
R = [];

% initialize param
refhigh = [];
oilflow_in = [];
cd2001_P = [];
cd2002_P = [];
valve_diaphragmleak = [];
valve_internleak = [];
valve_uppersaturation = [];
valve_lowersaturation = [];
valve_T = [];
valve_S = [];
valve_J = [];
transmitter_filter_enabled = [];
transmitter_bias = [];
transmitter_skew = [];
transmitter_Tm = [];

% input
while 0 < length(varargin)
    removeentries = 1;
    if isa(varargin{1},'char')
        switch(varargin{1})
            case 'container'
                % container
                results_container = varargin{2};
            case 'controller'
                %controller
                R = varargin{2};
                % param
            case 'refhigh'
                refhigh = varargin{2};
            case 'oilflow_in'
                oilflow_in = varargin{2};
            case 'cd2001_P'
                cd2001_P = varargin{2};
            case 'cd2002_P'
                cd2002_P = varargin{2};
            case 'valve_diaphragmleak'
                valve_diaphragmleak = varargin{2};
            case 'valve_internleak'
                valve_internleak = varargin{2};
            case 'valve_uppersaturation'
                valve_uppersaturation = varargin{2};
            case 'valve_lowersaturation'
                valve_lowersaturation = varargin{2};
            case 'valve_T'
                valve_T = varargin{2};
            case 'valve_S'
                valve_S = varargin{2};
            case 'valve_J'
                valve_J = varargin{2};
            case 'transmitter_filter_enabled'
                transmitter_filter_enabled = varargin{2};
            case 'transmitter_bias'
                transmitter_bias = varargin{2};
            case 'transmitter_skew'
                transmitter_skew = varargin{2};
            case 'transmitter_Tm'
                transmitter_Tm = varargin{2};
            otherwise, disp('Unknown field ''',varargin{2},'''')
        end
        if removeentries, varargin(1:2) = []; end
    else
        varargin(1) = [];
        warning('Each input should have a descripting string first')
    end
end

% load if not prespecified
if isempty(results_container)
    [ results_container,idx_list,~ ] = loadResults();
end

% lookup routine
dowarn1 = true;
dowarn2 = true;
indexlist = [];

%tmp fix
orig.cd2001_P = cd2001_P;
orig.cd2002_P = cd2002_P;
orig.oilflow_in = oilflow_in;
orig.transmitter_bias = transmitter_bias;
orig.transmitter_skew = transmitter_skew;
orig.transmitter_Tm = transmitter_Tm;
orig.transmitter_filter_enabled = transmitter_filter_enabled;
for idx=idx_list
    param = results_container{idx,1}.parameters;
    if idx==290
       disp('')
    end

    % handle bug in old version (some missing fields)
    cd2001_P = orig.cd2001_P; % part of tmp fix
    cd2002_P = orig.cd2002_P;
    oilflow_in = orig.oilflow_in;
    transmitter_bias = orig.transmitter_bias;
    transmitter_skew = orig.transmitter_skew;
    transmitter_Tm = orig.transmitter_Tm;
    transmitter_filter_enabled = orig.transmitter_filter_enabled;
    warstr = '';
    if ~isfield(param,'cd2001_P'), cd2001_P=[]; param.cd2001_P=[]; war-
str=[warstr,'cd2001_P ']; end
    if ~isfield(param,'cd2002_P'), cd2002_P=[]; param.cd2002_P=[]; war-
str=[warstr,'cd2002_P ']; end
    if ~isfield(param,'oilflow_in'), oilflow_in=[]; param.oilflow_in=[]; war-
str=[warstr,'oilflow_in ']; end
    if ~isfield(param,'transmitter_bias'), transmitter_bias=[];
param.transmitter_bias=[]; warstr=[warstr,'transmitter_bias ']; end
    if ~isfield(param,'transmitter_skew'), transmitter_skew=[];
param.transmitter_skew=[]; warstr=[warstr,'transmitter_skew ']; end
    if ~isfield(param,'transmitter_Tm'), transmitter_Tm=[];
param.transmitter_Tm=[]; warstr=[warstr,'transmitter_Tm ']; end
    if ~isfield(param,'transmitter_filter_enabled'), transmit-
ter_filter_enabled=[]; param.transmitter_filter_enabled=[]; war-
str=[warstr,'transmitter_filter_enabled ']; end
    if ~isempty(warstr) && dowarn1, warning(['Some parameter fields were missing
and ignored by default [ ',warstr,']']),dowarn1=false; end

    % special case of transmitter
    if transmitter_Tm == 0
        transmitter_Tm = param.transmitter_Tm;
        transmitter_filter_enabled = 0;
    else
        transmitter_filter_enabled = 0;
        if dowarn2
            warning('Ambiguity since transmitter Tm was larger than 0, but filter
was disabled')
            dowarn2 = false;
        end
    end

    % check if result is interesting
    if ...
            (isempty(refhigh) || isequal(refhigh,param.refhigh)) &&...
            (isempty(oilflow_in) || isequal(oilflow_in,param.oilflow_in)) &&...
            (isempty(cd2001_P) || isequal(cd2001_P,param.cd2001_P)) &&...
            (isempty(cd2002_P) || isequal(cd2002_P,param.cd2002_P)) &&...
            (isempty(valve_diaphragmleak) || ise-
qual(valve_diaphragmleak,param.valve_diaphragmleak)) &&...
            (isempty(valve_internleak) || ise-
qual(valve_internleak,param.valve_internleak)) &&...
            (isempty(valve_uppersaturation) || ise-
qual(valve_uppersaturation,param.valve_uppersaturation)) &&...
```

```
            (isempty(valve_lowersaturation) || ise-
qual(valve_lowersaturation,param.valve_lowersaturation)) &&...
            (isempty(valve_T) || isequal(valve_T,param.valve_T)) &&...
            (isempty(valve_S) || isequal(valve_S,param.valve_S)) &&...
            (isempty(valve_J) || isequal(valve_J,param.valve_J)) &&...
            (isempty(transmitter_filter_enabled) || ise-
qual(transmitter_filter_enabled,param.transmitter_filter_enabled)) &&...
            (isempty(transmitter_bias) || ise-
qual(transmitter_bias,param.transmitter_bias)) &&...
            (isempty(transmitter_skew) || ise-
qual(transmitter_skew,param.transmitter_skew)) &&...
            (isempty(transmitter_Tm) || ise-
qual(transmitter_Tm,param.transmitter_Tm))
        indexlist(end+1,1) = idx;
    end
end
disp(['Found ',num2str(length(indexlist)),' matches for your query'])

% visualize results
analyzer_plotBode( 'container',results_container,'indexes',indexlist )

% display margins XY plot
analyzer_plotIndicatorXY( 'container',results_container,'indexes',indexlist )

% display results as text
analyzer_displayResultsAsText( 'contain-
er',results_container,'indexes',indexlist,'controller',R )

end
```

1.7.3 “analyzer_plotBode.m”

```
function [ ] = analyzer_plotBode( varargin )
%READRESULTS_PLOTBODE Summary of this function goes here
%   Detailed explanation goes here

lin2log = @(x)log(x)./log(10); % anonymous function to convert from linear space
to log space
colorvec = 'gbmck';
R = [];
progresstate = 1;
figno = [1 2];
text_on = false;
results_container = [];
idx_list = [];
post_phase = 1;
% General code for reading variable input
while 0 < length(varargin)
    removeentries = 1;
    if isa(varargin{1},'char')
        switch(varargin{1})
            case 'container'
                results_container = varargin{2};
            case 'indexes'
```

```
                idx_list = varargin{2};
            case 'plottext'
                text_on = cell2mat(varargin{2});
            case 'figure'
                figno = varargin{2};
            case 'controller'
                R = varargin{2};
            case 'rtphase'
                removeentries = 0;
                varargin(1) = [];
                post_phase = 0;
            otherwise, disp('Unknown field ''',varargin{2},'''')
        end
        if removeentries, varargin(1:2) = []; end
    else
        varargin(1) = [];
        disp('Each input should have a descripting string first')
    end
end

if isempty(results_container)
    [ results_container,idx_list,progresstate ] = loadResults();
end

%% Process
% open/create figure
h1=figure(figno(1));
subplot(2,1,1)
title('Gain and phase, Bode plot')
whitebg([.3 .4 .4])
h2=figure(figno(2));
title('Gain and phase, Bode plot (3D)')
view(3)
whitebg([.3 .4 .4])

for idx = 1 : length(idx_list)
    cont_idx = idx_list(idx);
    % points along the x dimension
    try
        cont = results_container{cont_idx}.relaysweep;

        gain_dB = cont.gain_process_vec;
        f_gain = cont.w_hyst_vec_hz;
        f_gain_base10 = lin2log(f_gain); % logarithmic 10 base axis

        if ~post_phase
            phase = cont.pha_process_vec;
            f_pha = cont.w_delay_vec_hz;
            f_pha_base10 = lin2log(f_pha);
        else
            phase = nan(16,1);
            for i = 1:length(cont.gain_process_vec)
                x = cont.hystsignals{i}.c.Data;
                y = cont.hystsignals{i}.ym.Data;
                [~, pha_rad] = post_fft_estAmpPha(x,y);
                pha_deg = pha_rad*180/pi;
                if 0 < pha_deg, pha_deg = pha_deg - 360; end
                phase(i) = pha_deg;
            end
            f_pha_base10 = f_gain_base10;
        end
%                       phase = flipdim(phase,1); % bug in earlier version of saved
signals (fixed now)

        % plots
        comm_state = results_container{cont_idx}.results.commissioning_state;
        if comm_state, color = 'r';
        else color = colorvec(mod(cont_idx-1,length(colorvec))+1);
        end
        figure( figno(1) )
        subplot(2,1,1), hold on, plot(f_gain_base10,gain_dB,color), yla-
bel('Amplitude [dB]'), grid on
        subplot(2,1,1), plot(f_gain_base10,gain_dB,'oy')
        if comm_state, legend('Commissioning state'); end
        subplot(2,1,2), hold on, plot(f_pha_base10,phase,color), ylabel('Phase
[degrees]'), xlabel('frequency [Hz], base 10'), grid on
        subplot(2,1,2), plot(f_gain_base10,phase,'oy')

        % 3d plot
        figure( figno(2) ), hold on, plot3(f_gain_base10,gain_dB,phase,color),
        xlabel('frequency [Hz], base 10'), ylabel('Gain [dB]'), zlabel('Phase
[degrees]')
        grid on
        plot3(f_gain_base10,gain_dB,phase,'oy')

        figure( figno(1) )
        subplot(2,1,1),
text(f_gain_base10(end),gain_dB(end),['s:',num2str(cont_idx)])
        subplot(2,1,2),
text(f_pha_base10(end),phase(end),['s:',num2str(cont_idx)])
        figure( figno(2) ),
text(f_pha_base10(end),gain_dB(end),phase(end),['s:',num2str(cont_idx)])
    catch err
    end
end
end
```

1.7.4 “analyzer_plotIndicatorXY.m”

```
function analyzer_plotIndicatorXY( varargin )
%ANALYZER_PLOTINDICATORXY Summary of this function goes here
%   Detailed explanation goes here

colorvec = 'gbmck';
results_container = [];
idx_list = [];
while 0 < length(varargin)
    removeentries = 1;
    if isa(varargin{1},'char')
        switch(varargin{1})
```

```
            case 'container'
                results_container = varargin{2};
            case 'indexes'
                idx_list = varargin{2};
            otherwise, disp('Unknown field ''',varargin{2},'''')
        end
        if removeentries, varargin(1:2) = []; end
    else
        varargin(1) = [];
        warning('Each input should have a descripting string first')
    end
end

if isempty(results_container)
    [ results_container,idx_list,~ ] = loadResults();
end

figure
xlabel('Gain margin (dB)')
ylabel('Phase margin (degrees)')
title('Margins XY plot')
for i = 1 : length(idx_list)
    idx = idx_list(i);
    results = results_container{idx}.results;
    try
        XdB = 20*log10(results.A_m);
        Ydeg = results.phi_m;

        if results.commissioning_state, col='r';
        else col = colorvec(mod(idx-1,length(colorvec))+1);
        end

        hold on
        plot(XdB,Ydeg,['o',col])
        text(XdB,Ydeg,num2str(idx))
        hold off
    catch err
    end
end
end
```

1.7.5 “analyzer_plotResponses.m”

```
function analyzer_plotResponses( varargin )
%PLOTRESPONSES ( varargin ) plots inputs of type 'timeseries'
% Accepts variable size input of type timeseries
%   Not very good code... should be refined

figno = [];
% General code for reading variable input
while 0 < length(varargin)
    if isa(varargin{1},'char')
        switch(varargin{1})
```

```
            case 'signals'
                signals = varargin{2};
            case 'figure'
                figno = varargin{2};
            otherwise, disp('Unknown field ''',varargin{2},'''')
        end
        varargin(1:2) = [];
    else
        varargin(1) = [];
        disp('Each input should have a descripting string first')
    end
end

if isempty(figno), figure
else figure(figno)
end
whitebg([0 .5 .6])
colorvec = 'rgbmck';
legends = {};
for i = 1 : size(signals,1)
    color = colorvec(mod(i-1,length(colorvec))+1);
    prop = signals(i);
    if isa(prop,'timeseries')
        hold on, plot(prop.Time,prop.Data,color), hold off
        legends{i} = prop.Name;
    elseif isa(prop,'cell')
        hold on, plot(prop{1}.Time,prop{1}.Data,color), hold off
        legends{i} = prop{1}.Name;
    end
end
legend(legends)
end
```

1.7.6 “lookup_footprint_script.m”

```
clc
% Script for analysing results
Kp=4;
Ti=100;
Td=0;
num = Kp*[Ti*Td Ti 1];
den = [Ti 0];
R = tf(num,den);
% set parameter configurations of interest
refhigh = DEFAULT.oper.refhigh;
oilflow_in = DEFAULT.oper.refhigh;
cd2001_P =  [];
cd2002_P =  [];
valve_diaphragmleak = DEFAULT.valve.diaphragmleak;
valve_internleak = DEFAULT.valve.internleak;
valve_uppersaturation = DEFAULT.valve.uppersaturation;
valve_lowersaturation = DEFAULT.valve.lowersaturation;
valve_T = DEFAULT.valve.T;
valve_S = DEFAULT.valve.S;
valve_J = DEFAULT.valve.J;
```

```
transmitter_filter_enabled = DEFAULT.transmitter.filter_enabled;
transmitter_bias = DEFAULT.transmitter.bias;
transmitter_skew = DEFAULT.transmitter.skew;
transmitter_Tm = DEFAULT.transmitter.Tm;

% lookup function
analyzer_lookupResults( ...
    'controller',R,...
    'refhigh',refhigh,...
    'oilflow_in',oilflow_in,...
    'cd2001_P',cd2001_P,...
    'cd2002_P',cd2002_P,...
    'valve_diaphragmleak',valve_diaphragmleak,...
    'valve_internleak',valve_internleak,...
    'valve_uppersaturation',valve_uppersaturation,...
    'valve_lowersaturation',valve_lowersaturation,...
    'valve_T',valve_T,...
    'valve_S',valve_S,...
    'valve_J',valve_J,...
    'transmitter_filter_enabled',transmitter_filter_enabled,...
    'transmitter_bias',transmitter_bias,...
    'transmitter_skew',transmitter_skew,...
    'transmitter_Tm',transmitter_Tm...
    )
```

1.7.7 “combineSegments.m”

```
function [ ] = combineSegments( )
%COMBINDSEGMENTS Summary of this function goes here
%   Detailed explanation goes here
%   Combines result file segments into one larger file

% browse for files
disp('User must select multiple files to merge')
[filename,pathname] = uigetfile('*.mat','Select a .m file that holds results from
a completed simulation','MultiSelect','on');
if isequal(filename,0) || isequal(pathname,0)
    disp('User selected Cancel')
    return
end
if ~isa(filename,'cell')
    disp('User did not select multiple files to merge')
    return
end

% combine data
results_container = {};
for i = 1 : length(filename)
    tmp = load([pathname,filename{i}]);
    if isfield(tmp,'results_container')
        for j = 1 : length(tmp.results_container)
            % prevent duplicates from being added
            is_included = 0;
            for k = 1 : length(results_container)
```

```
                if isequal(results_container(k),tmp.results_container(j))
                    is_included = 1;
                    disp('Result was already part of structure')
                end
            end
            % add if not already added
            if ~is_included
                results_container(end+1,1) = tmp.results_container(j);
            end
        end
    else disp(['Missing field results_container in ',filename{i}])
    end
end

% save
[filename2, pathname2] = uiputfile([pathname,'*.mat'],'Save as','Combined.mat');
if isequal(filename2,0) || isequal(pathname2,0)
    disp('User selected Cancel')
else
    try
        savename = fullfile(pathname2,filename2);
        save(savename,'results_container');
        disp(['Saved as ',savename])
    catch err
        disp('Error occured')
    end
end
end
```

1.7.8 “loadResults.m”

```
function [ results_container,idx_list,progresstate ] = loadResults()
%LOADRESULTS Helper function that loads a results container

results_container = [];
%% load
progresstate = 0;
[filename,pathname] = uigetfile('*.mat','Select a .m file that holds results from
a completed simulation');
if isa(filename,'char')
    load([pathname,filename]);
    if exist('results_container','var')
        cont_length = length(results_container);
        if 1 < cont_length
            prompt1 = ['Index of first simulation results you want to display.
Min 1'];
            prompt2 = ['Index of last simulation results you want to display. Max
',num2str(cont_length)];
            answer = inputdlg({prompt1,prompt2},'Plot di-
alog',1,{num2str(1),num2str(cont_length)});
            if isempty(answer)
                disp('User cancelled')
                return
            else
```

```
                % check if several results are specified

                % evaluation of each result index
                idx_1 = str2double(answer{1});
                idx_2 = str2double(answer{2});
                if idx_1 < 1 ||  cont_length < idx_2
                    disp('Index out of bounds')
                    return
                else idx_list = idx_1 : idx_2;
                end
            end
        else idx_list = 1;
        end
        progresstate = 1;
    else disp('The file contained an unfamiliar structure')
    end
else disp('No .mat file returned from browser')
end
end
```