Fluid Power Applications Using Self-Organising Maps in Condition Monitoring
Fluid Power Applications Using Self-Organising Maps in Condition Monitoring

Anders Zachrison

Division of Fluid and Mechanical Engineering Systems
Department of Management and Engineering
Linköping University
SE–581 83 Linköping, Sweden

Linköping 2008
To Ludwig and Åsa

‘Minds are like parachutes –
they only function when open’

Thomas Dewar
Condition monitoring of systems and detection of changes in the systems are of significant importance for an automated system, whether it is for production, transport, amusement, or any other application. Although condition monitoring is already widely used in machinery, the need for it is growing, especially as systems become increasingly autonomous and self-contained. One of the toughest tasks concerning embedded condition monitoring is to extract the useful information and conclusions from the often large amount of measured data. The use of self-organising maps, SOMs, for embedded condition monitoring is of interest for the component manufacturer who lacks information about how the component is to be used by the system integrator, or in what applications and load cases.

At the same time, there is also a potential interest on the part of the system builders. Although they know how the system is designed and will be used, it is still hard to identify all possible failure modes. A component does not break at all locations or in all functions simultaneously, but rather in one, more stressed, location. Where is this location? Here, the collection of as much data as possible from the system and then processing it with the aid of SOMs allows the system integrators to create a map of the load on the system in its operating conditions. This gives the system integrators a better chance to decide where to improve the system.

Automating monitoring and analysis means not only being able to collect prodigious amounts of measured data, but also being able to interpret the data and transform it into useful information, e.g. conclusions about the state of the system. However, as will be argued in this thesis, drawing the conclusions is one thing, being able to interpret the conclusions is another, not least concerning the credibility of the conclusions drawn. This has proven to be particularly true for simple mechanical systems like pneumatics in the manufacturing industry.
Acknowledgements

The work presented in this thesis has been carried out at the Division of Fluid and Mechanical Engineering Systems at Linköping University. I want to express my gratitude to several people who have been involved during the course of my work.

First of all, I would like to thank my co-supervisor, Dr. Magnus Sethson, for his support, encouragement, and last but not least our discussions, always leading to far too many new ideas. I would also like to thank Prof. Jan-Ove Palmberg both for his support and for giving me the opportunity to join the division.

A huge thank-you all to all the members and former members of the division! You have made this time very exciting, not least the discussions about everything and nothing over a cup of coffee. I'd like to mention Marcus Rösth and Lars Andersson especially for all our discussions and Ronnie Werndin and Henrik Petterson for many hands-on tips in the laboratory.

I would also like to mention the technical staff at the Department of Mechanical Engineering for their help, especially Ulf Bengtsson.

Finally, I would like to thank my family for their support over the years, Åsa for her patience and tolerance during the work and the writing of the thesis, and last but not least, Ludwig, our son, for all the joy he brings.

Linköping in January 2007

Anders Zachrison
Papers

The following five papers are appended and will be referred to by their Roman numerals. All papers are printed in their originally published state with the exception of minor errata, changes in text and figure layout, and changes in the language and notation in order to maintain consistency throughout the thesis.

In the papers, the first author is the main author, responsible for the work presented, with additional support from the co-writer.


The following published papers are not included in the thesis but constitute an important part of the background of the work presented.


# Contents

1 Introduction and Background .................................................. 1
   1.1 Background  ............................................................... 1
   1.2 Grand vision ............................................................... 2
   1.3 Structure of the thesis ................................................... 3

2 Test systems ............................................................................. 5
   2.1 Pneumatic system ............................................................. 5
   2.2 Hydraulic servo system .................................................... 7
      2.2.1 Control system ......................................................... 8
      2.2.2 Faults and their implementation ................................ 8

3 Condition Monitoring ............................................................. 11
   3.1 Condition monitoring principles ........................................ 11
      3.1.1 Model-based approaches .......................................... 13
      3.1.2 Data-driven approaches ........................................... 13
      3.1.3 Online / off-line monitoring .................................... 14
   3.2 Fault modelling ............................................................... 15
      3.2.1 Fault classification ................................................. 16
      3.2.2 Fault models ........................................................... 17

4 Neural Networks ........................................................................ 19
   4.1 Learning patterns ............................................................. 19
   4.2 Self-organising maps ......................................................... 20
      4.2.1 SOM algorithm ......................................................... 21
      4.2.2 A short SOM example .............................................. 23
      4.2.3 Usage and properties of the SOM ............................. 24
      4.2.4 Training with reduced dimension ............................. 27
      4.2.5 Structure of the used SOM ..................................... 28
      4.2.6 Scaling and normalisation ...................................... 28

5 Technical Parameter Estimation Using Self-Organising Maps ......... 31
   5.1 Introduction ................................................................. 31
   5.2 SOM adaption ............................................................... 32
Introduction and Background

The work presented here concerns the use of self-organising maps in condition monitoring of technical systems, especially pneumatic systems. The techniques discussed here also lend themselves to cooperation with / post-processing the results from model-based methods.

1.1 Background

Condition monitoring of systems and detection of changes in the systems are of significant importance for an automated system, whether it is for production, transport, amusement, or any other application. Although condition monitoring is already widely used in machinery, the need for it is growing, especially as systems become increasingly autonomous and self-controlled. One of the toughest tasks concerning embedded condition monitoring is to extract the useful information and conclusions from the often large amount of measured data. The converse, drawing conclusions from a minimum of data, is also of interest. In this case, interest is at least two-fold: to reduce costs (fewer sensors) and to create redundant monitoring and analysis systems. The use of self-organising maps, SOMs, for embedded condition monitoring is of interest for the component manufacturer who lacks information about how the component is to be used by the system integrator, or in what applications and load cases.

At the same time, there is also a potential interest on the part of the system builders. Although they know how the system is designed and will be used, it is still hard to identify all possible failure modes. A component does not break at all locations or in all functions simultaneously, but rather in one, more stressed, location. Where is this location? Here, the collection of as
much data as possible from the system and then processing and structure it with the aid of SOMs allows the system integrators to create a map of the load on the system in its operating conditions. This gives the system integrators a better chance to decide where to improve the system.

Automating monitoring and analysis means not only being able to collect prodigious amounts of measured data, but also being able to interpret the data and transform it into useful information, e.g. conclusions about the state of the system. However, as will be argued in this thesis, drawing the conclusions is one thing, being able to interpret the conclusions is another, not least concerning the credibility of the conclusions drawn. This has proven to be particularly true for simple mechanical systems like pneumatics in the manufacturing industry.

1.2 Grand vision

The motivation for this work is partly the vision of “self-aware machines” given in the introduction section of the previous thesis, [1]. The idea is of a machine that has knowledge about itself and its surrounding environment and is able to react to changes in that environment. To support this vision, a number of engineering disciplines need to be merged and the control strategy “predictive simulation adaptive control”, PSAC, developed.

Such a machine needs a way to know whether it is in a new, unknown situation. This situation could be caused by changes in the operating environment, but could also be due to faults in the machine itself.

As a large part of the vision is that most of the control and supervision systems should be automatically assembled from development models etc., a supervision system capable of detecting unknown situations on its own is needed. One such system is proposed and presented in this thesis, based on self-organising maps.

How, then, does the work in this thesis fit into the larger vision as described above?

A model-based condition monitoring and monitoring of both the system itself, as well as the surrounding environment, would be better suited for both the control strategy presented in [1–3], and as an aid in the identification of faults (not only the detection of them); by using a complete and accurate model of the system and its surrounding environment, all information needed both for control and condition monitoring is available. Thus, optimal control and monitoring would be possible.

There are nevertheless reasons to look into data-driven methods. Concerning the control concept described earlier in this section, data-driven methods could be useful to detect new, unknown situations in which it might be preferable to continue operation with greater care and to proceed with caution, i.e. behave as if the situation is somewhat frightening. How should the surrounding environment (or the system fault) be modelled, if no knowledge about where and in what environment the system will be run (or how the system will break)
exists? Thus, data-driven methods are necessary, at least as a complement to model-based methods.

In the case of more traditional use of the condition monitoring system, the monitored system might be too complex to reliably model the connections between the measured variables. A model-based approach could also result in too much data, thus advocating the use of a data-driven method on top of the model-based method.

1.3 Structure of the thesis

First, in chapter 2 the two test systems used in this work are described. Following this, in chapter 3 an overview on condition monitoring techniques is presented. In chapter 4 an overview of artificial neural networks and self-organising maps is given, followed by a discussion of the use of self-organising maps in condition monitoring in chapter 5. Another example of how the self-organising map can be used is given in chapter 6 which deals with operation estimation.
In this chapter a brief overview is given of the actual systems used to test the ideas and concepts in this thesis.

2.1 Pneumatic system

A pneumatic rod-less cylinder was chosen as the first test system. The pneumatic cylinder was chosen as it is a highly non-linear system, where internal and external friction plays a significant part (further increasing the non-linear behaviour). The pneumatic system is also characterised by a large variation in dynamics, the fast pneumatics and the slow thermodynamic effects, none of which may be excluded.

The cylinder is controlled by four 3/2 valves used as on/off-valves, allowing individual control of the two chambers’ exhaust and inlet flows. In figure 2.1 a photo of the test system is shown and a schematic sketch in figure 2.2.

The stroke of the cylinder used in the appended paper [I] is 1100 mm and the diameter is 40 mm. The mass load on the piston is 10 kg. In the other appended papers, [II]- [V], the stroke is 1000 mm and the diameter is 50 mm.

A variable mass load is used, to obtain different operating conditions and also to introduce known/unknown changes to the system. The loads used are 0 kg, 10 kg, and 20 kg; where 10 kg represented the normal state case. A second set of changes to the operating system is introduced by placing the outlet valves backwards, which results in mainly two simultaneous changes: a slight change to their time characteristics, but it also introduces a larger change to the system, as there is some leakage through the valves in this configuration.
Fluid Power Applications Using Self-Organising Maps in Condition Monitoring

(a) The original test system used in paper [I].

(b) The test system as used in the later papers.

Figure 2.1: A photo of the test system.

Figure 2.2: A schematic sketch of the test system. The rod-less pneumatic cylinder is controlled by four on/off-valves, allowing independent control of both chambers’ exhaust and supply valves.
Control system

The control system consists of four sensors and a normal desktop computer. There are three pressure sensors, one in each chamber inlet and one monitoring the supply pressure. In the original system, figure 2.1a, there is also a rotational potentiometer used to measure the piston position by means of a wire mounted at the right end of the cylinder. The newer system, figure 2.1b, uses a linear resistance sensor (using a conductive plastic as resistance track). This sensor greatly enhances the resolution of measured piston position.

The controlling computer is equipped with double PII–400 MHz processors. The computer is running Linux together with a real-time extension, RTAI\footnote{Real-Time Application Interface for Linux, see www.rtai.org}, to obtain real-time performance and behaviour from the system.

2.2 Hydraulic servo system

A hydraulic position servo, see figures 2.3 and 2.4, is used as the test object. A MOOG servo valve is used to control the symmetric cylinder. Three pressure sensors are used, one measuring the system pressure and the other two the chamber pressures. A linear position sensor is mounted in the cylinder. Additionally, the servo valve also measures the spool position.

![Figure 2.3: A photo of the test system.](image)

\footnote{Real-Time Application Interface for Linux, see www.rtai.org}
2.2.1 Control system

The control algorithm for the hydraulic position servo, as well as the simulated faults are implemented in MATLAB/Simulink. For the actual control and measurement, as well as user interface, dSPACE hardware and software are used.

2.2.2 Faults and their implementation

Two symptoms of faults have been studied in the hydraulic test system in this work: an offset of the spool and a decrease in the bandwidth of the valve. A schematic sketch of the main stage of a servo valve is shown in figure 2.6. An offset in the spool position means that when the valve is commanded to the centre position, it will actually be slightly open to either of the two load ports. This then means that a command signal different than 0 is needed to properly close the valve. Such an error could be caused both by mechanical damage or electrical faults.

These two implemented faults have been applied between the output from the controller and the D/A-converter, see figure 2.5. The simulated faults and the corresponding fault levels are listed in table 2.1. The offset is calculated as
Test systems

Figure 2.6: *A schematic drawing of a valve body and main spool.*

A percentage of the maximum valve opening (control signal for the simulated fault), and for the bandwidth the percentage of the degradation of bandwidth of the valve is shown. Both single faults and double faults are simulated in the test rig. Thus, all possible 25 combinations from table 2.1 are used for the study.

Table 2.1: *The fault modes and the fault levels used.*

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset in spool</td>
<td>0% 0.5% 1% 1.5% 2%</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>0% 5% 10% 15% 20%</td>
</tr>
</tbody>
</table>

The decrease in the bandwidth of the valve is simulated by letting the command signal, for the faulty cases, pass through a low pass filter with the cut-off frequency set to \( \omega_{v,X\%} = (1 - X\%) \omega_{v,n} \), with X\% taken from table 2.1. The nominal bandwidth of the valve is here denoted \( \omega_{v,n} \), and is roughly 80-100 Hz (depending on the amplitude of the command signal).

The test system itself, neglecting the dynamics of the much (7–10 times) faster servo valve, has a bandwidth of

\[
\omega_h = \sqrt{\frac{4/3 A_p^2}{V_t M_t}} \approx 79.9 \text{ rad/s} \approx 12.8 \text{ Hz} 
\]  

(2.1)
This also assumes a centred piston (equal chamber volumes). $\beta_e$ is the effective bulk modulus, $A_p$ the piston area, $V_t$ the total chamber volume, and $M_t$ the total mass load of the system.
Condition monitoring, in one form or another, has been of great interest since the dawn of industrialisation. In the beginning it was performed by the skilled operators, using their senses to estimate the condition of the machinery. Noises, vibrations etc. made the operators aware that something was about to go awry. However, with the arrival of modern times with operators being moved farther away from the machine, and machines become controlled by control systems, their ability to monitor the condition of the machinery is severely reduced. Hence, the need to also automate this important part of the operators’ responsibilities grows. Figure 3.1 illustrates this change.

In this chapter, a broad overview on condition monitoring principles and techniques will be given, followed by, a brief review of some of the simpler fault models and basic classification will be given.

3.1 Condition monitoring principles

Quite a few different principles of condition monitoring exist. These range from static thresholding of sensor signals, via data-driven approaches to methods based on first principle models, see figure 3.2 for a short overview.

Another classification of condition monitoring principles is whether they are on- or off-line methods, see section 3.1.3.

Model-based fault detection has progressed significantly in recent decades and is based on mathematical models of the physics of the monitored system [6–8]. Knowledge based fault detection, on the other hand, uses qualitative models (cause-effect graphs etc.) and is thus suited for systems for which mathematical models are hard to develop or unachievable for the engineering community involved with the development of the system. Data-driven approaches are a
(a) Each machine had at least one operator, constantly working on the machine, thus hearing and seeing the condition of the machine.

(b) As automatisation came along, each operator got a number of machines to care of. Thus, the need grew for alarm bells and lights to attract the attention of the operator.

(c) Even later on, as the machines were further automated, the operators were moved away from the machines to separate control rooms. This requires even more monitoring capabilities.

Figure 3.1: The evolution of the machine operator. The operator has evolved from manufacturing components by directly controlling a machine, to supervising the machining process from a control room.

<table>
<thead>
<tr>
<th>First principles</th>
<th>Data-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling</td>
<td>Neural networks</td>
</tr>
<tr>
<td>Parameter estimation</td>
<td>SOM</td>
</tr>
<tr>
<td>Kalman filter</td>
<td>Case-based reasoning</td>
</tr>
<tr>
<td>Parity space (residual generation)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: An overview of some monitoring principles and their classification into model-based (first principle) and data-driven methods.
third method, suitable when the first two are not applicable. The data-driven methods could also be useful in other circumstances, due to their simplicity, adaptivity, and the lack of need for in-depth knowledge of the system, see [9] for an overview of some data-driven methods like support vector machines, k-nearest neighbour and principal component analysis. In this work, their applicability even when some limited system knowledge is available will be shown. They also have a potential use together with model-based approaches.

### 3.1.1 Model-based approaches

A few examples of model-based approaches are shown in figure 3.2.

Parameter based methods could for instance incorporate running an optimisation on a simulation model in order to find what parameter values will produce a good fit between the model and the measured data. Ramdén has an example of this in [10, 11], where the control system of a gear box is monitored. To find the value of the monitored parameters, the complex method is used in the HOPSAN simulation package, see [12].

Other model-based methods are based on structured hypothesis tests, where the test quantities can be designed in a number of different ways: residuals, observers, or the likelihood function etc. In [13], Nyberg discusses the design of a diagnostic system using structured hypothesis tests and the prediction errors as the test quantities. In this work, a systematic design process is developed capable of handling faults of different behavioural modes (where the different behavioural modes are described by different fault models, see section 3.2.2).

### 3.1.2 Data-driven approaches

Data-driven approaches are based on the common idea that instead of creating an explicit model and matching the available data to it, new data is compared to older, already processed, data. The format of the data storage, as well as the the processes used to both store and compare the data, is specific to each individual data-driven method. One such method, the self-organising (feature) map, will be discussed in great detail in chapter 4 and its use in condition monitoring in chapter 6.

The data-driven approaches can be divided into a large number of classes. Two such classes are classification-based methods and case-based reasoning, which share a great many similarities. In both classes, the intention is to match the current state of the system to already known states.

Additionally, data-driven approaches, especially some of the approaches in this thesis, could be useful as either a pre- or post-processing route in the case of model-based condition monitoring. Post-processing in particular is of interest, as large amounts of information can easily be obtained and/or the detection of new faults/states could be improved.
Classification

Mundry and Stammen discuss online condition monitoring techniques in [14, 15], where they also, in the latter, recommend specific methods for a number of components. In general, their recommended methods include neural network techniques to process data, together with additional pre-processing. In the case of pumps and motors, the suggested method is to obtain the frequency spectra of the pulsations or vibrations and use this as input to a neural network. The same idea was discussed by Ramdén in [11] (and in her previous papers, eg. [16]), although she used the backstrum and cepstrum to gain some robustness. Both the backstrum and cepstrum are used to find periodical behaviours in the spectrum, and the former has the advantage that very low amplitudes will not have a disproportionately large significance. Stammen, on the other hand, presents another idea to gain robustness in [17]. Here the idea is to calculate a transfer function of the pump, that will transform the individual pump spectra to a standard spectra for this model of pump.

Donat, et al., compare a number of other classification techniques, viz. support vector machine, probabilistic neural network, k-nearest neighbour, principal component analysis, Gaussian mixture model, and physics-based single fault isolator in [9]. They also use self-organising maps to visualise the complexity of the classification tasks, as well as to get a figure for the complexity. One of their goals is to look at data reduction techniques, and if it is possible to improve the fault classification performance by using them. In the work in this thesis, on the other hand, no data reduction techniques or feature extraction techniques are used, instead the raw data is used directly. By using some of the data reduction techniques discussed in [9], it is possible that some of the performance improvements they discovered would apply here too.

Case-based reasoning

Olson, et al., present case-based reasoning for condition monitoring in [7, 8]. The authors claim that other data mining methodologies suffer from the need for relatively large training sets and the risk of over-fitting. The alternative they offer, case-based reasoning, is considered to be free from these problems. It also gives the opportunity to learn from experience, but skipping the data training step.

3.1.3 Online / off-line monitoring

No matter what base approach (model-based or data-driven) is used, the methods can still be divided into on- and off-line methods.

The model-based method described in [10] is an example of an off-line method. The basis here is that measurements from one cycle (in this case a gear shift) is used to extract features. An optimisation routine then tries to match these to the features extracted when running a simulation of the
same process for various parameter values. Then the set of parameter values that made the best match between the features is chosen and these parameters describe certain faults. Such an optimisation can potentially take quite some time and computing resources, thus it is not certain that it is suitable for implementation in an ECU.

An example of an on-line method is the structured hypothesis tests with test quantities based on residuals described by Nyberg, [13].

The data-driven method, and its variants, discussed in chapter [3] and the appended papers [II–V] can be used in both an on- and off-line fashion. This is also demonstrated in the appended papers. Online results are for instance available in the appended papers [II, III], while in the appended paper [V] off-line results are discussed.

The distinction between on- and off-line methods is sometimes quite thin. Quite often, it is possible to use an off-line approach in a semi-online fashion by running the off-line algorithm at a certain interval, eg. every \( n \)th sample. This obviously requires the algorithm to not take too long to finish processing and that we do not need to supply the algorithm with a large, new batch of measurements each time.

### 3.2 Fault modelling

Especially in model-based diagnosis and condition monitoring, modelling of the possible faults is of utmost importance and thus the engineer needs expertise in how faults occur and what types of fault exist. In the case of data-driven methods, it might not be that clear that this knowledge is an advantage. However, if one knows what classes of faults one is interested in and how they manifest themself, it is a lot easier to design a data-based condition monitoring system that will be sensitive to these faults. To detect an abrupt change in a signal, one set of features might be enough, but if the fault is instead an intermittent one, it is not guaranted that you can detect it reliable at all using the same set of features.

It is also useful to consider whether the fault is in a sensor or in an actuator both for condition monitoring as well as for fault tolerant control. A fault in a sensor might after detection be corrected, while a faulty actuator can be more troublesome.

Fault models can also provide additional insights for the engineer. Can a certain fault at all be detected using the available signals? Thus, it is still of interest to look at some of the different classes of faults and how they can be modelled.

---

1. Electronic Control Unit
3.2.1 Fault classification

Faults can be classified according to many schemes. Two classification schemes will be discussed here. These two schemes complement each other. The first concerns where the fault occurs, while the second set describes its time-variant behaviour. See Blanke, et al., [6], for a more thorough treatment of fault types.

Subsystem classification

When classifying the fault according to the subsystem where the fault occurs, the following classes of faults can readily be recognised:

- process faults (also known as system or component faults)
  Changes in friction, mass, leaks, components getting stuck or loose. The leaking exhaust valves and the change mass load used in this work belongs to this class. Both the offset fault and the decreased bandwidth of the hydraulic servo valve (section 2.2.2) used in this work could belong to this class, if the actuator is considered a system itself.

- sensor faults
  Short-cut and cut-off in connectors and wirings. Also changes in gain, bias and bandwidth.

- actuator faults
  Actuators can themself be considered systems, thus all faults from process and sensors are applicable. The faults (offset of the spool and decrease in bandwidth) in the hydraulic example, section 2.2.2 in this work belongs to this class.

Time-variant behaviour classification

The time-variant behaviours of the faults are, if possible, even more important than their subsystem classification when discussing the detectability of the respective fault. Three main groups of time-variant behaviours are:

- abrupt change
- incipient fault
- intermittent fault

Abrupt changes could for instance be the cut-off of a wire or hose. Such faults are easily detected. However, it is not always certain that an abrupt fault will be that easy to detect; a change in bias or friction for instance might in some cases only be detected during transients and thus be invisible for long periods of operation. It might also happen that an abrupt change is only detectable during a certain time close to the change, due to the dynamics of the system.
Take for example the fictitious example of a speed sensor and a position sensor. If the two sensors are monitored by comparing the speed sensor to the derivate of the position sensor (3.1), an abrupt change in the position sensor will only be shown momentarily, while, on the other hand, if the monitoring is done by comparing the position sensor to the integrated speed (3.2), an abrupt change in either sensor will be detectable for a long period.

\[ r_1(t) = \frac{d}{dt}x(t) \]  
\[ r_2(t) = x(t) - \int_{t_1}^{t_2} v(t) \, dt \]  

An intermittent fault is often hard to find. It is necessary to be able to detect the fault while it is there, as it could soon go away and the system will work normally again.

### 3.2.2 Fault models

Just as there exist many ways to classify faults, the faults in the different classes can be modelled in a number of different ways. In this section, we will take a look at a few models. See for instance Blanke, et al., [6], for a further discussion of fault models, as well as model-based diagnosis.

Although no fault model is needed in the case of data-driven condition monitoring, it is still useful to consider how a certain fault should/could be modelled, especially concerning what type of restrictive model one would prefer to use, as this might give some understanding of what sensor signals it is necessary to include in the set of features for the condition monitoring task, as well as the detectability of the fault.

#### General models

The first class of models is general models. Here, the models are designed such that as few restrictions as possible are set upon them. The most general fault model of them all is to model the fault as a signal, \( f(t) \), which implies no restrictions at all on the behaviour of the fault. Thus, this fault model fits all faults and the use of this model could make detection somewhat easier, but it makes identification of the fault very hard or impossible.

#### Restrictive models

Usually, a more restrictive modelling of the possible faults in a system could make it easier to both detect the fault and draw conclusions about how it will affect the system. Examples of restricted fault models are the abrupt addition of a bias, (3.3), or an abrupt change in the gain, (3.4). The offset fault of the valve spool described in section 2.2.2 is implemented according to (3.3).
A pressure spike might cause a pressure sensor to give either a bias in future readings and/or change the gain of the sensor.

\[ y(t) = x(t) + f_1 \quad f_1 = \begin{cases} 0 & t < t_{ch} \\ \theta_1 & t \geq t_{ch} \end{cases} \quad (3.3) \]

\[ y(t) = f_2 x(t) \quad f_2 = \begin{cases} \theta_0 & t < t_{ch} \\ \theta_2 & t \geq t_{ch} \end{cases} \quad (3.4) \]

When using such fault models, the idea is often to estimate the parameters \( \theta_1 \) and \( \theta_2 \). When this can be done, the operator could get an estimate of how large the deviation from the normal state is. This is also often used to form a hypothesis test.

Other types of restrictive fault models could be the introduction of an explicit delay or a change in the dynamics of the component, such as the decrease in bandwidth in the hydraulic example. This can be modelled by letting the signal pass through a filter, for instance a low pass-filter as is done in section 2.2.2.

\[ y(t) = x(t - f_3) \quad f_3 = \begin{cases} \theta_0 & t < t_{ch} \\ \theta_1 & t \geq t_{ch} \end{cases} \quad (3.5) \]

Using more restricted fault models also allows the identification of what fault has occurred. A general fault model makes such conclusions a lot harder, not least when there are several possible faults in the system modelled by general models. On the other hand, a general model requires less information and is thus easier to use. The restricted models require more information about the fault, how it occurs, and what its effects are.
Neural networks are well suited for modelling non-linear systems in an approximate fashion. They are also suited for classification problems and information storage.

Neural networks, NN, or more correctly artificial neural networks, were originally developed to model the neurons in the brain and their synapses in order to simulate the human brain. Today, NN is generally viewed as a mathematical tool rather than a model of the brain. Some NN structures, such as the Kohonen self-organising map, are still sometimes looked upon as models of the brain. See for instance the discussion of different kinds of brain maps by Kohonen in [18], eg. the tonotopic map in the auditory cortex. Another example is found in the work done by Erwin, et al., [19], in which the visual cortex is studied.

4.1 Learning patterns

Neural networks store empirical knowledge by learning from examples. They can be classified in terms of the amount of guidance that the learning process uses. An unsupervised learning network, figure 4.1a, learns to classify input patterns without external guidance; it tries to approximate the probability distribution of the training vectors. As such, the input vector to the unsupervised learning network has to describe the complete state of the environment, both what in the common case would be called the input as well as the output. A supervised learning network, on the other hand, adjusts the weights of the neurons on the basis of the difference between the values of output units and the desired values given by a “teacher”, for a given input pattern, see figure 4.1b. An example of an unsupervised learning network is the self-organising map, SOM, or feature map.
Fluid Power Applications Using Self-Organising Maps in Condition Monitoring

(a) Unsupervised training. The input vector describes the complete state of the environment.

(b) Supervised training. The teacher produces the desired results, which are then compared to the results from the learning system, $\Sigma$.

Figure 4.1: Two classical learning patterns for neural networks.

4.2 Self-organising maps

Self-organising (feature) maps are a special kind of neural network developed by Kohonen, see for instance [20, 21]. The SOM was inspired by neurobiology, in particular by ideas derived from cortical maps in the brain. It has been shown in [19] that the SOM is able to explain the formation of computational maps in the primary visual cortex of the macaque monkey. As such, the SOM can still be viewed as one model of the brain’s information and signal routing processes.

The SOM works by approximating the probability distribution of the input vectors by its neurons’ weight vectors. As such, it accumulates knowledge during training and this knowledge is distributed in the same areas as the input vectors are. This allows the SOM implementation to keep a record of well-known regions in the input domain and distinguish these from unknown, novel inputs.

One common use for the SOM is to categorise data. Other uses are the closely related fields of data mining and encoding/decoding. Kohonen, et al., present an overview of the application of SOMs in engineering applications in [22]. In one example, used in [23, 24], each neuron (node) is associated to a model of some observation, in this example with a short-time spectrum of natural speech. This is used to create a phoneme map for a speech recognition application.

Of special interest for this work is the use of SOMs for condition monitoring. Two ways of using the SOM for condition monitoring are discussed in [25]. First, the quantisation error between the winning neuron and the feature vector could

---

1 Also known as Kohonen networks
be used; secondly, the SOM can be trained to include a forbidden area. Other examples of condition monitoring are [26, 27]. Jack studies automated fault detection in helicopter gearboxes. Lumme discusses condition monitoring by using the SOM as a categoriser, and also to some extent how to handle new conditions not known to the SOM.

Another aspect of the SOMs of especial interest for this work is their use to estimate friction and other parameters of physical systems. Two such papers are [28, 29]. In [28], Schütte describes an application where friction and other parameters are estimated for an electric drive. Later, the structure of the system is also recognised (stiff/non-stiff, backlash/no backlash), and a suitable controller is automatically derived. Naude, [29], studies the use of SOMs to capture the tendencies of the stick-slip phenomena to aid machine tool design.

A comprehensive text on neural networks in general, also covering SOMs, is [30], where both the Willshaw–von der Malsburg and the Kohonen models are discussed. However, the Kohonen model is the one that has received most attention in the literature, and which will be used in this work. A second thorough text on SOMs is [21], which is one of the first attempts to produce a complete overview of the algorithms, variations and ideas behind the SOM and the closely related learning vector quantisation, LVQ.

One interesting feature of the SOM, especially with Kohonen’s version, is the ability to perform dimensional reduction. An \( m \)-dimensional input signal is mapped onto the (usually) 2-dimensional neuron lattice. This property is used for example for encoding an \( m \)-dimensional signal, (e.g. an image), by just storing which neuron gets the hit for each signal value (in the case of a two-dimensional lattice, two coordinates need to be stored as opposites to the \( m \) values in the signal. These \( m \) values might usually also need many more bits to be encoded than the position coordinates in the lattice. One example of how this dimensional reduction property is used is the visualisation of the complexity of data, as used by Donat, \textit{et al.}, [9].

\subsection{SOM algorithm}

In this work and description of the self-organising map, Kohonen’s model is used. The Kohonen model consists of one layer of neurons, which are all stimulated by the input, see figure 4.2. The output is normally the winning (best matching) neuron; however, additional means of creating output are discussed later, in section 4.2.4 and also in Schütte, \textit{et al.}, [28] and in the appended paper [I].

The different phases in the use and training of the SOM can be divided into three parts: the competitive process, the cooperative process and the adaptive process. In the following, these processes will be considered as they are described in [30].

The implementation of the SOM is quite straightforward; however, some interesting questions arise. One such question is whether all dimensions in the
The layout of the Kohonen model as used in this work. The model consists of a two-dimensional array of neurons, which are all stimulated by the input vector.

Figure 4.2: The layout of the Kohonen model as used in this work. The model consists of a two-dimensional array of neurons, which are all stimulated by the input vector.

input vector are to be used when the neurons compete. This will be further discussed in section 4.2.5.

Competitive process

The $m$-dimensional input vector randomly selected from the input space is denoted by

$$\mathbf{x} = [x_1, x_2, \ldots, x_m]^T$$

(4.1)

The synaptic weight vector of neuron $j$ is denoted by

$$\mathbf{w}_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \quad j = 1, 2, \ldots, l$$

(4.2)

where $l$ is the number of neurons in the lattice. The $l$ neurons are often organised in an $n \times n$ two-dimensional lattice, where $l = n \cdot n$. (The lattice can be structured in a number of ways, from a traditional rectangular grid to a hexagonal grid, for example). Each component of $\mathbf{w}_j$ corresponds to the same component in $\mathbf{x}$. In the competitive process the winning neuron $i$ is determined by

$$i(\mathbf{x}) = \arg \min_j \| \mathbf{x} - \mathbf{w}_j \|, \quad j = 1, 2, \ldots, l$$

(4.3)

As the output of the SOM is normally taken solely from the winning neuron, this leads to the following observation (from [30]):

“A continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network.”

The SOM can in such a case only deliver a maximum of $l$ different values as the output, hence the discrete output space. The neurons try to approximate the
probability distribution, \( p(x) \), of the input vectors in the output space. This is done by moving the neurons in the feature space during the adaptive process.

**Cooperative process**

To determine how the input vector should affect different neurons in the lattice, a neighbourhood function is used. A typical choice of the neighbourhood function \( h_{j,i} \) is the Gaussian function

\[
h_{j,i}(x) = \exp \left( -\frac{d_{j,i}^2}{2\sigma^2} \right)
\]

which will decrease as the distance between neurons \( j \) and \( i \) increases. In the case of a two-dimensional lattice, \( d_{j,i} \) is defined by

\[
d_{j,i}^2 = \| r_j - r_i \|^2
\]

where the discrete vector \( r_j \) defines the position of neuron \( j \) in the lattice.

The neighbourhood function, \( h_{j,i} \), normally shrinks with time and this is realised by making \( \sigma \) a function of time (discrete time \( n \)).

\[
\sigma(n) = \sigma_0 \exp \left( -\frac{n}{\tau_1} \right), \quad n = 0, 1, 2, \ldots
\]

and thus letting \( h_{j,i} \) be defined as

\[
h_{j,i}(n) = \exp \left( -\frac{d_{j,i}^2}{2\sigma^2(n)} \right)
\]

**Adaptive process**

The adaption of the neurons in the SOM to the input vector is performed in the adaptive process. The training is done almost exactly as in the (in NN) standard competitive learning rule, with the exceptions that all neurons are adapted and that the adaption rate is influenced by \( h_{j,i} \). A discrete-time adaption process for the neurons is

\[
w_j(n + 1) = w_j(n) + \eta(n)h_{j,i}(n)(x - w_j(n))
\]

where \( \eta(n) \) is the time-dependent learning-rate parameter

\[
\eta(n) = \eta_0 \exp \left( -\frac{n}{\tau_2} \right)
\]

Again, \( n \) is the discrete time, \( n = 0, 1, 2, \ldots \)
4.2.2 A short SOM example

As a short example of how the neurons in a SOM organise themselves, let us take a look at a one-dimensional SOM map. This means that the neurons are situated along a line (most probably not a straight line). Let us also assume that the weight vector of each neuron consists of two values, \([\alpha, \sin\alpha] \). Initially, if no knowledge of the distribution of the training data exists, the neurons are just filled with random values, i.e. both of the two values in each neuron are chosen randomly and independently of each other. This initial distribution of the neurons in the SOM is shown in figure 4.3a. The neurons are marked by asterisks. As can be seen, no order exists among the neurons and no resemblance to our intended structure of the weight vector \([\alpha, \sin\alpha] \) can be seen.

The distribution of the neurons after 40 and 300 training iterations is shown in figures 4.3b and 4.3c, respectively. In the first of these two figures, it can be seen that the basic order of the neurons has been established, although they still have not found a form resembling the desired distribution \((\alpha, \sin\alpha)\). In the latter figure, on the other hand, the sine-shape is strong and only some minor adjustments are left to do.

After further training of the SOM, the neurons have adapted themselves to the training data, see figure 4.3d. Here it is also possible to see that the neurons are adapted as a group, the 1-dimensional structure follows the sine curve smoothly. Even though this example does slightly abuse the intentions behind the SOM, it still illustrates how the SOM works and adapts to the training data.

4.2.3 Usage and properties of the SOM

Classification

As stated in the beginning of this section, SOMs are suitable for classification. The normal procedure is (see Haykin, [30]):

1. Train the SOM.

2. Match each neuron to the training data.

3. Store and use this classification.

Thus, the SOM is first trained as normal, using the training data (which at this point still can be unclassified). For the second step, the training data needs to be classified. Each neuron is compared to the training data, and its classification is decided by a majority vote. Then this classification is stored for each neuron. A small example of a classification map is shown in figure 4.4.
Figure 4.3: Illustration of the training of a SOM. Here, a 1-dimensional SOM is trained using a sine-period as training data. The solid, smooth line represents the training data (the sine-curve) and the asterisks the neurons in the SOM. The neurons sit in a 1-dimensional structure, thus the line connections between neighbouring neurons. For higher order (order > 2) neurons, it is not possible to visually compare the lattice to the training set.

Figure 4.4: A small example of how the neurons in the lattice can be grouped after classification. In this example, two classes (faulty and normal state) are used and the 5 neurons in the lower left corner were deemed to belong to the faulty class.
An example of classification

In the previous section, the classical classification process using SOMs was described. In this section, we take a look at a simple example, illustrating the outcome of the process. The example is inspired by a larger example of classification of different animals, given by Kohonen in [18] (section 3.14.1).

The task is to train a SOM to recognise different vehicles due to their configuration and if they require a licence to drive. In the example, four different types of vehicles are used, namely: MC, car, bicycle, and snowmobile. In table 4.1 the vehicles and their recorded attributes are shown. A SOM, with 4-by-4 neurons in the lattice, is trained using these attributes as the feature, thus the feature vector is 6-dimensional. In figure 4.5 the SOM-lattice is shown with labels on the best matching neuron for each of the different vehicle types.

<table>
<thead>
<tr>
<th></th>
<th>MC</th>
<th>Car</th>
<th>Bicycle</th>
<th>Snowmobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Wheels</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4 Wheels</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Engine</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Handlebar</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Steering wheel</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>License req.</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: The vehicles and their properties (features) as used in the example of classification.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>Snowmobile</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>Bicycle</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Car</td>
<td>-</td>
<td>-</td>
<td>MC</td>
</tr>
</tbody>
</table>

Figure 4.5: The SOM-lattice with labels on the neurons matching each type of vehicle.

Parallelisation properties

The structure of the SOM algorithm inherently lends itself for parallelisation. During the competition between the neurons, the lattice can be divided into two halves, and a winning neuron is determined in each. After that, the two winners can be compared to find the global winner. These two halves can further be divided into quarters, and so on. Such a scheme shows significant similarities with the efficient method of calculating Fourier transforms, the Fast
Fourier Transform (FFT). Also, both the cooperative process and the adaptive
process are only dependent on the individual neuron, and are thus also easily
parallelisable.

One way to use this parallelisation property is to implement the SOM algo-
rithm in hardware, for instance on an FPGA. Some papers discussing hard-
ware implementations using an FPGA, and the performance/cost to do this
are [31–35]. Such an implementation could become extremely fast, making it
suitable also for real-time control/monitoring applications requiring a combi-
nation of high band-width and large feature vectors.

4.2.4 Training with reduced dimension
A question arises here: does the intended use of the SOM affect how the winner
is chosen? If the SOM is to be used to predict friction, should the friction
component of the neuron actually be used when looking for the winner during
training? Obviously, it cannot be used during prediction/estimation. To avoid
neurons with, in our case, almost identical first four components and a large
difference only in the last component (here friction), it is advantageous to train
the SOM as it is going to be used, using only the first four components for the
competition.

The input vector can thus be divided into two parts: an independent part
and a dependent part (called active and non-active components in [28]. This
gives the following notation for the input vector and the modified competitive
process:

$$x = \left[ x^{\text{ind}}, x^{\text{dep}} \right]^T$$

$$i(x) = \arg\min_j \| x^{\text{ind}} - w_j^{\text{ind}} \|, \quad j = 1, 2, \ldots, l$$

If the SOM is used as a “look-up” table to estimate/predict certain parameters
(for instance friction in appended paper [I] or other physical parameters in [28]),
this modified approach is better suited, as it will produce a more unique winning
neuron.

From an engineering point of view, a large variation in $x^{\text{dep}}$ for similar $x^{\text{ind}}$
during training, indicates that there is a shortage of information here, as the
SOM cannot resolve the different states. To solve this, more information is
needed in the feature vector, thus either more measurements are needed, or the
process needs to be complemented by a model. Thus, by keeping track of the
standard deviation of the dependent part of the neurons’ weight vectors during
training, it is possible to understand whether the dimension of the feature
vector is sufficient.

In the case of condition monitoring by grouping similar states (or features)
together, i.e. classification, this scheme of using $x^{\text{ind}}$ for the matching could
still be valuable, see sections 6.3 and 6.5 or appended papers [II] to [V].

\footnote{Field Programmable Gate Array, a reconfigurable electronic device}
However, if the SOM should be in the more classical classification scheme, it is advantageous to use the full input vector. Here the useful output from the SOM could be the winning neuron’s position in the lattice. This position could be used in two ways: either to group different fault modes together as areas in the lattice or to create a trajectory of the winning neuron in the lattice. An example of an application where such a trajectory is used for monitoring can be found in [25]. This trajectory is suited for systems working in a repetitive cycle.

Here, in the classification task, support vector machines (SVMs) might give better results, either alone or in combination with SOMs. One way to combine them would be to use the SOM to derive a suitable set of features to be fed to the SVM. This way of combining them is a most interesting object for future studies. The SVM is a supervised learning method, currently regarded as state of the art in classification algorithms. It works in the opposite way to the SOM: it maps the input data into a high-dimensional space, linearly separating different classes of input data. As such, the training is quite costly. It is equivalent to solving a linearly constrained Quadratic Programming problem, in which the number of variables equals the number of data points. Support vector machines are dealt with exhaustively in the different articles in [36].

4.2.5 Structure of the used SOM

In this work, input spaces of different dimensions are used according to what is to be investigated. For instance, in the case of learning the friction response of the pneumatic cylinder shown in figure 2.1a (and studied in chapter 5), a 5-dimensional input space is used. The 5 dimensions consist of

\[ [x_p(n), \dot{x}_p(n), p_A(n), p_B(n), F_{fric}(n)] \]  

(4.12)

In order to restrain one of the input dimensions from becoming too dominant, the actual values are scaled such that the “absolute normal values” are within the interval [0, 1] (outliers and extremes can still be outside this interval. The actual form of the input vector in (4.12) is thus

\[ [x_p(n), \dot{x}_p(n), \frac{p_A(n)}{7 \cdot 10^5}, \frac{p_B(n)}{7 \cdot 10^5}, \frac{F_{fric}(n)}{300}] \]  

(4.13)

A similar approach to the scaling is used in the other papers with their corresponding feature vectors.

4.2.6 Scaling and normalisation

A perhaps better scaling would be to translate the features used to build the input vector to a common mean, for instance 0 (with no loss of generality), and then scale the features to a common variance.
In the literature it has sometimes been suggested that $\mathbf{x}$ should be normalised (to unit length) before use. As stated by e.g. Kohonen in [21], this is not necessary in principle, although there may be numerical advantages such as improved numerical accuracy due to the input vectors then having the same dynamic range. In an application where the absolute values of different features are of interest, such as this one (the friction estimation), normalisation might not be desirable. The proposed normalisation scheme in the previous paragraph should basically give the same advantages while still making the features easy to interpret.

As a result, the two-dimensional lattice will be trained to represent the $m$-dimensional input space with the training vectors. Afterwards two possible uses of the lattice exist: either to use the lattice as an estimator by feeding it with an $(m-n)$-dimensional input vector and have it return the remaining dimension(s) from the winning neuron. Alternatively, the indexes of the winning neuron in the lattice itself could be used to group states together in order to, for instance, draw conclusions about the condition of the system. This latter use would benefit from the lattice having been trained with both training vectors from a good condition and from faulty conditions. Both these approaches and their application to condition monitoring will be discussed in greater depth in chapter [6].
5

Technical Parameter Estimation Using Self-Organising Maps

Estimation of parameters is an important part of both the modelling and monitoring phases of the design and use of technical systems. The same applies to mapping of parameters and/or model structures. This is an area where the SOM has some potential use.

5.1 Introduction

For estimation of parameters in a system, several methods exist. One way is to use SOMs. A few papers dealing with this problem are [28, 29] and appended paper [I]. In [28], Schütte describes an application where friction and other parameters are estimated for an electric drive. Later, the structure of the system is also recognised (stiff/non-stiff, backlash/no backlash), and a suitable controller is automatically derived. Naude, [29], studies the use of SOMs to capture the tendencies of the stick-slip phenomena to aid machine tool design.
5.2 SOM adaption

The technique used to allow a mapping from input states to estimated parameters in both the work done by Schütte, [28], and in appended paper [I] is to append the estimated parameters at the end of the training vector, see section 4.2.4. The augmented feature vector is then divided into two parts: an independent part and a dependent part. This extra information, such as estimated parameters, are put into the dependent part and not used in the competitive process. As stated in section 4.2.4, the modified competitive process is used and the augmented parameters are seen as the output of the matching between the input vector, $x^{ind}$, and the neurons in the SOM-lattice.

5.3 Friction estimation

In order to be able to try and map the friction force, $F_{fric}$, against piston position, $x_p$, velocity, $\dot{x}_p$ and chamber pressures, $p_A, p_B$, a way to estimate the friction force is needed. As a first approach, the simplest off-line estimation is to low-pass (with a null-phase) filter the position and pressure signals, and then high-pass filter the position signal in order to derive the velocity and acceleration signals. An approximation of the friction force is then

$$F_{fric} = A_p (p_A - p_B) - m_p \ddot{x}_p$$

(5.1)

This assumes a horizontal system, with no external force load. Due to the use of HP-filters, problems occur when using noisy signals. One case in which the estimate will differ significantly, is when the piston reaches one end of the cylinder, in this case the estimate of $F_{fric}$ will include the force from the cylinder end dampers and seat. This explains the increase (spike) in the estimated force at the middle of the cycles, starting from $\sim 12s$ in figure 5.1 (it can also be seen that the piston reaches the end of the cylinder at the same time). Additional spikes (mainly negative ones) can be seen once every fifth/sixth cycle; these come from the measurement card (the last spikes will be of no concern to us, as the method used to estimate/illustrate friction in this work will attenuate these, effectively working as a low-pass filter).

5.4 Results

5.4.1 Input sequence

Most of the results are based on an open-loop sequence, where the valves are open for 1 s and closed for 1 s. These square waves result in the triangular piston movement shown in figure 5.1. During the sequence, the piston drifts towards the end of the cylinder and after approximately half the sequence the piston spends a small part of the cycle pressed against the end. The estimated friction
force during this sequence is also shown in figure 5.1. The results presented here are from a SOM trained with the reduced set of dimensions, i.e. the friction component of the weight vector, \( w^{deP} \), was not used to determine the winning neuron.

![Graph of cylinder position and estimated friction force over time](image)

**Figure 5.1:** The sequence used in this work. At the top is the cylinder position. The lower figure shows the estimated friction force used to train the SOM.

### 5.4.2 Friction estimation

To study how well the trained SOM works as a friction estimator, a second test run of the measured sequence is performed. The estimate from the SOM is calculated using \( x_p, \dot{x}_p, p_A, p_B \) from this new measurement. In figure 5.2, the estimates (according to SOM) from four cycles of the sequence are compared to the SOM estimate. A close-up of the second cycle is shown in figure 5.3.

During the short periods when the piston is stuck at the end, the estimated friction force increases. This is due to not estimating the actual friction force in this case, but rather getting the complete cylinder force as the estimated force instead. One occasion where this happens is used before the sample time 550, where there is a relatively large spike.
Figure 5.2: The friction estimate during four cycles compared to a validation sequence. The solid line is the estimate from the SOM and the dash-dotted line is the validation data. Note how the estimate suddenly rises when the piston reaches the far end of the cylinder.

Figure 5.3: A close-up of the second cycle in figure 5.2.
Figure 5.4: The trajectory created by the winning neuron from an SOM trained using an augmented feature vector. The first four parameters in the input vector, \( x \), are used to find the winner. The trajectory is superimposed on a graph showing the amount of training each neuron has received. Note the small deviation caused by striking the cylinder end.

In figures 5.4 and 5.5, the trajectory created by the winning neuron on the SOM-lattice for the test sequence can be seen. In the first figure, the SOM is trained using a reduced set of features during the matching, while the second figure is trained using the complete feature vector. The difference in the deviation from the normal path, when the piston hits the cylinder end is obvious (this point is denoted “Max friction” in the respective graphs, see appended paper [I] for an explanation of the choice of the label). These results pave the way for the next chapter, about condition monitoring using SOMs.
Figure 5.5: The trajectory created by the winning neuron from an SOM is more suited to condition monitoring. All five parameters in the input vector, \( x \), are here used to find the winner. Here the deviation becomes much larger when the piston reaches the cylinder end (following a path to neuron (15,5) instead of going through neuron (13,11)).
6 Condition Monitoring Using Self-Organising Maps

Self-organising maps have great potential as a method of data-driven condition monitoring. This applies both for the cases of unknown or known faults, as well as the combined case. In this field, both the categorisation property and the ability to recognise well-known and new sets of features are used.

6.1 Feature vectors

The choice of what to use as features in the input vector to the SOM is of utmost importance, not least when it comes to using the SOM for classification purposes, see section 6.3.2 and figure 6.2 for an example of a potential problem.

The desired state is to have features corresponding to certain faults, and only to these, thus always creating unique combinations. This allows for perfect matching and classification. In appended papers [III, IV], classification is used to detect and identify known faults. However, as direct sensor signals (cylinder pressures and piston position) are used as features, feature sets exist that equally well belong to different classes (normal/faulty). Other types of derived features in a pneumatic system were developed by Fritz and Murrenhof in [37]. Krogerus, et al., [38] uses the wavelet transform to derive features as a pre-processing step for the water hydraulic lift system in a forklift.

Donat, et al., discuss data reduction techniques in [9], in order to minimise the uncorrelated information and memory requirements. In their work, they
use the visualisation property of the SOM to illustrate the complexity of the classification task, by the clustering in the SOM-lattice (one cluster for each class, as compared to a large number of small clusters spread all over the lattice for each class).

6.2 Quantisation error – unknown faults

When using a self-organising map only to detect deviation from the normal state of the system, the SOM is trained using data only from the normal state. In this case, the quantisation error, q.e.,

\[ q.e. = \| x - w_i \| \] (6.1)

is regarded as the best measure for fault detection. This measure represents the misfit of the test vector, \( x \), to the best matching neuron, \( w_i \), in the SOM. Kasslin, et al., presents an example in [25], in which a power transformer is analysed in this way. Appendix paper [II] includes an analysis of the mean \( q.e. \) during a working cycle, and the influence of the choice of the feature vector on the detectability of the introduced faults.

6.2.1 Results with only unknown faults

Training and validation data

In the experiments conducted here, the measurements from the following test conditions are used:

- 0 kg mass load
- 10 kg mass load
- 20 kg mass load
- 10 kg mass load and leaking exhaust valves

An example of a test run from the second test condition is shown in figure [III].

6.2.2 Results from the hydraulic servo

To illustrate the use of additional domain knowledge when using a data-driven method, the following experiment was performed. On a hydraulic position servo, two faults were simulated, see section [II]. The two faults were an offset of the valve spool and a decrease in the bandwidth of the valve. In the experiment tests were conducted with no fault and 4 fault levels of each of the two faults, thus resulting in 25 test cases. Only the no fault data were used to train the SOMs. In appended paper [V] two variants of condition monitoring using SOMs were investigated.
Figure 6.1: An example of the training data for the SOMs. This is one of the 10 kg mass load tests. The top graph shows the position of the piston when running a specific open-loop control sequence. The lower graph shows the two chamber pressures: \( p_A \) as a solid line and \( p_B \) as a dash-dotted line.

In one of the two variants, an ARX-model is estimated from the commanded control signal, sent to the valve, to the load pressure. When using the estimated ARX-parameters as the feature vector, the results in table 6.1 were achieved. The same results, normalised with the no fault result, is shown in 6.2. Here it is easier to see the large response to both faults.

6.3 Classification – known faults

In classical use of the SOM for classification, each neuron is classified, and that class is used as the result if/when this neuron is the winning neuron. In figures 5.4 and 5.5, another way of using the SOM for condition monitoring is shown. Here a path is constructed by looking at the winning neuron in consecutive samples, and deviations from this path indicate a new system state. Both ways of using the winner, either by constructing a path from consecutive samples or by directly relating the position to a fault mode, could be described as a classification process. In [25], Kasslin, et al., uses the trajectory in the SOM-lattice to analyse a power transformer and the pattern normal usage creates. A fault moves the trajectory to a new area of the lattice, thus indicating that a certain type of fault has occurred.
Table 6.1: The quantisation error for the ARX parameters of the hydraulic position servo.

<table>
<thead>
<tr>
<th>Offset [%]</th>
<th>Decrease in bandwidth [%]</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0038</td>
<td>0.4799</td>
<td>0.6529</td>
<td>0.5860</td>
<td>0.0355</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.0522</td>
<td>0.1468</td>
<td>0.1177</td>
<td>0.0446</td>
<td>0.0037</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.0715</td>
<td>0.1264</td>
<td>0.1193</td>
<td>0.0631</td>
<td>0.0139</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>0.0335</td>
<td>0.1138</td>
<td>0.1235</td>
<td>0.0358</td>
<td>0.0108</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>0.0310</td>
<td>0.3895</td>
<td>0.1179</td>
<td>0.0375</td>
<td>0.0062</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: The normalised quantisation error for the ARX parameters for the hydraulic position servo. Normalised with the q.e. for the normal case.

<table>
<thead>
<tr>
<th>Offset [%]</th>
<th>Decrease in bandwidth [%]</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>125.3</td>
<td>170.5</td>
<td>153.1</td>
<td>9.3</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>13.6</td>
<td>38.3</td>
<td>30.7</td>
<td>11.6</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>18.7</td>
<td>33.0</td>
<td>31.2</td>
<td>16.5</td>
<td>3.6</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>8.8</td>
<td>29.7</td>
<td>32.3</td>
<td>9.4</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>8.1</td>
<td>101.7</td>
<td>30.8</td>
<td>9.8</td>
<td>1.6</td>
<td></td>
</tr>
</tbody>
</table>

Although the SOM algorithm is good at the classification task, the classification performance could be further improved by also using “learning vector quantisation”, LVQ, see [21]. The LVQ tries to move the decision borders to a more optimal location by moving neurons either towards or away from the input vector. A problem with the LVQ, is that the quantisation error becomes affected as a useful measure. In normal classification tasks, this is not a problem. However, as will be argued in section 6.5, even in the case of classification, the quantisation error is still a useful measure for condition monitoring. Another approach would be to apply another classification algorithm, eg. the support vector machines, SVM, see [36].

A question that arises when looking at monitoring using classification is how to handle new, previously unknown states. While the SOM can quite easily be trained with the no fault condition and different loads, working points etc., it is impossible to train the SOM for all possible future faults. One way to handle this is to send an alarm to the operator when a novel condition, that is too far away from all assembled knowledge, appears. Once the operator or service technician has determined what caused the alarm, these unknown input vectors can be labelled, and once a sufficiently large number of these new vectors exists, a retraining of the SOM can be ordered, see [27] for a further discussion by Lumme of a similar scheme. The newly trained SOM will then have a larger knowledge base than the previous one.
It is not uncommon in the case of known faults (supervised/classification based SOMs) that traditional neural networks perform equally well or better than SOMs; an example can be found in [39], by Kuravsky and Baranov. One reason to still use SOMs is their ability to handle unknown faults.

Krogerius presents a combination of the wavelet transform for feature extraction and SOMs for classification in [38]. In this work, two SOMs are used, one to distinguish between normal and faulty state and one to identify the fault.

### 6.3.1 Classification procedure

In section 4.2.3 the normal procedure for classification using SOMs is described. In this work, another procedure has been used, which results in a gradual classification. The principle used is the same one as used by Schütte, [28], for parameter estimation and in chapter 6. Prior to the training, the classification is augmented to the training vectors as part of $x^{dep}$ in eq. (4.10). As a result the classification is part of the adaptive process, the classification of each neuron undergoes the same adaption as the rest of the features. The result of this is that, in a classification scheme where 0 denotes fault-free state and 1 denotes a faulty state, it is possible to have neurons whose classification feature is between 0 and 1, for instance 0.3. Such a gradual classification has the potential to track and classify a system with a slowly increasing fault, as compared to the classical classification which enforces a distinct border between the two cases.

Such a classification, for example as 0.3 in the example above, means that similar states can occur in both the fault-free and the faulty states. Some reasons why this might occur are discussed in section 6.3.3 and some potential problems associated with this are discussed in section 6.3.2.

See the discussion about learning vector quantisation in appended paper [III], especially figure 3 in that paper, for a more in-depth discussion of this property and why LVQ is not suited for this kind of classification process.

### 6.3.2 Problems associated to classification of known faults

As was already hinted in section 6.1 one of the main tasks when using classification algorithms is to make sure that the system will be able separate all the classes/states. If the system is not able to do this, the problem illustrated in figure 6.2 will occur, i.e., two (or more) classes/states will share the same set of features in a overlapping region. Then the question arises: to what class does this input vector belong? (C.f. to what circle does the black dot in figure belong?). In this work, this is handled by giving this input vector a classification that lies between the two classes, see section 6.3.1.

That this is not only a potential problem, but rather a real problem, can be seen for instance in appended paper [III] and in figure 6.3. In the result section, that discusses the classification (leak indicator), there are a number of occasions where the faulty system resembles the normal state more than the faulty state according to the classification procedure.
Figure 6.2: A fictitious example showing a problem for fault classification algorithms. Similar measurements (features) are found in both classes; the decision areas thus overlap.

6.3.3 Results with known faults

Here the same pneumatic system has been used and results from three states are used. These are the normal state, the known fault (leaking exhaust valve), and an unknown state (an increase in the mass load). In figure 6.3 the classification results can be seen. As can be seen, there are occasions where the classification states that the system response is more similar to the normal state even though the leak is present (for instance the large dips around sample 850 in figure 6.3). However, the overall result is quite good as the leakage is classified correctly almost all of the time.

What could be the reason behind some of the leak indicator’s dips in the third part of figure 6.3? One example of a state in which it is impossible to distinguish between all three of the states in figure 6.3 would be as follows: the piston is moved to one end of the cylinder, such that the faulty (leaking) exhaust valve is fully open and the inlet valve to the other chamber is also fully open. In this situation, the piston will be resting against the cylinder end with system pressure in one chamber and atmosphere pressure in the other. In this case, neither the leaking exhaust valve nor a variation in the mass load will result in any difference compared to the normal state.

6.4 Accumulated excitation

It would be useful to have an easy-to-get measure of the confidence of the knowledge each neuron has. However, such a measure does not readily exist. One way to try to find such a measure is to define the accumulated level of excitation, $H_j(N)$, that each neuron has received.

$$H_j(N) = \sum_{n=1}^{N} \eta(n) h_{j,i}(n),$$  \hspace{1cm} (6.2)
Condition Monitoring Using Self-Organising Maps

Figure 6.3: The classification (leak indicator) from the SOM. A 0 denotes the normal state and 1 indicates leakages in both exhaust valves. The first two parts correspond to the normal state of the system. The third part has leaking exhaust valves and the final part has an increased mass load ($m = 20$ kg).

The matching neuron in (4.3) or (4.11). This is basically the sum of the training factor, $\eta(n)h_{j,i}(n)$, each neuron has received during the adaptive phase, see eq. (4.8). As $H_j(N)$ increases, the neuron accumulates more and more information and becomes more certain that the stored information is well founded.

As discussed in the appended papers, this is not a definitive confidence measure of the neuron; rather, it is also for this measure necessary to take into account the overall structure of the SOM lattice. However, it is still useful as an additional aid when interpreting the results (see parts of section 6.5), and it has been shown that in some cases it is possible to use this measure for both monitoring and raising alarms, see appended paper [IV].

The author has been unable to find the use of the accumulated training on an individual neuron level in any previous work.

6.5 Combined measures

Using multiple measures, it should be possible to further improve the condition monitoring performance and the interpretation of the monitoring results.

6.5.1 Quantisation error and classification

The first of the combined measures can be derived by still using the quantisation error with a SOM trained for classification. A prerequisite for this is that no further training, such as the learning vector quantisation, has been conducted, as this will destroy the q.e. measure. In table 6.3 the conclusions that can be drawn for different combinations of q.e. and classification are summarised. One question here is: what is a low or high q.e.? In appended paper [IV] it is
shown that for an SOM trained with a training sequence consisting of a non-
equal proportion between the different classes, the distinction between low and
high has to be adjusted. A class that has supplied far fewer training data than
another is likely to be represented by far fewer neurons, thus getting a much
coarser resolution in the feature space.

In general, a denser area of the lattice, which results in a smaller q.e., is
the result of a large number of training vectors matching this state. Thus, the
denser areas of the lattice correspond to the more well-known states among
the training data, while areas with a lower density correspond to lesser-known
system states.

<table>
<thead>
<tr>
<th>Table 6.3: Combinations of q.e. and fault classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low q.e.</td>
</tr>
<tr>
<td>Faulty state</td>
</tr>
<tr>
<td>Normal state</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

6.5.2 Accumulated excitation as an aid

The accumulated excitation level can also be combined with either the classi-
fication or the q.e. This is summarised in tables 6.4 and 6.5 respectively. The
same remarks apply here as in section 6.5.1 in order to interpret what is a low
or high level of excitation, one has to know how the structure of the SOM-lattice
looks like.

A low level of accumulated excitation, as compared to the rest of the lat-
tice, indicates that the corresponding system states have rarely been visited.
However, this conclusion also has to take into account the distribution of the
training vectors, just as in section 6.5.1.

Another combination that is worth a special remark, is the combination of
a high level of excitation and a high q.e. in table 6.5. This means that an area
which should have a lower level of excitation (the high q.e. indicates that only
a small share of the training vectors could have been close to these neurons)
actually has received a high level of excitation. This is a warning that it is quite
likely that the SOM is over-trained. Here, some caution has to exercised as one
has to remember that the q.e. will be larger in the faulty regions, assuming that
only a small portion of the training data comes from the faulty state. This is
further discussed in appended paper [IV].

Over-training (or over-learning) is always a risk when working with neural-
network algorithms. When alternating between learning and test phases, the
performance of the network is first increased through each iteration but sud-
denly the performance is decreased slowly. Kohonen, [18], states that it is
necessary to stop the training after some “optimal” number of iterations, typ-
ically 50-200 times the total number of codebook vectors. However, he also
states that such a stopping rule can only be found by experience.

Similar studies concerning both over-training per se, and also to help understand what a high and low level of accumulated excitation is for this region of the SOM-lattice, can be conducted by comparing the accumulated excitation and the mean distance between each neuron and its neighbours. A discussion of q.e. and mean distance can be found in appended paper [IV] and diagrams exemplifying the mean distance can be found in figure 5 in that paper. Similar reasoning can be done when studying the accumulated training and the mean distance between a neuron and its neighbours. In table 6.5, the term q.e. could be interchanged with “mean distance” and the interpretation would still be the same. In this case, the overall condition of the SOM-lattice can be studied, without having an explicit set of input signals to feed to the SOM and calculate the q.e. with.

Table 6.4: Conditions for combinations of fault classification and excitation level

<table>
<thead>
<tr>
<th>Normal state</th>
<th>Faulty state</th>
</tr>
</thead>
<tbody>
<tr>
<td>High excitation</td>
<td>Normal</td>
</tr>
<tr>
<td>Low excitation</td>
<td>Ambiguous, resembling normal</td>
</tr>
</tbody>
</table>

Table 6.5: Conditions for combinations of q.e. and excitation level

<table>
<thead>
<tr>
<th>Low q.e.</th>
<th>High q.e.</th>
</tr>
</thead>
<tbody>
<tr>
<td>High excitation</td>
<td>Normal</td>
</tr>
<tr>
<td>Low excitation</td>
<td>New, non-unique</td>
</tr>
</tbody>
</table>
Discussion and Concluding Remarks

The focus of the thesis has been condition monitoring with the aid of self-organising maps. For a number of occasions, data-driven approaches to condition monitoring is appealing. The self-organising (feature) maps are one such method that has proven useful both in this work and in the work of numerous others.

To illustrate how SOMs could be used, they were first used in this thesis to estimate and illustrate friction in a pneumatic cylinder. That the SOM is able to assemble and store such information is shown in figure 5.2 and the close-up in figure 5.3 which shows that the SOM has created an accurate description of the non-linear system that is the pneumatic system.

That using a SOM only for parameter estimation is not a computationally effective solution is quite clear; on the other hand, the computational effort is always constant with an SOM and it could also deliver other benefits. Further, a parallel implementation either in software or in hardware (eg. an FPGA) would at least reduce the time needed for the SOM.

Training the SOM by using either a full or a reduced set of dimensions of the input vector is also considered. In the case of parameter estimation it is recommended that a reduced set of dimensions is used during training. The same is true in the classification procedure proposed and described in the thesis, which uses the classification as an extra set of features. It thus resembles the parameter/operation estimation in chapter 4 and it is not a strict classification, but rather a gradual one.

From figure 6.3 it the conclusion can be drawn that it is possible to distinguish between the normal and faulty states using direct measurements as features.
The results are quite good, with almost no sample that is uncertain of its state. The values in the classification between normal and faulty show that we have similar situations (system states) occurring both in the normal case as well as in the faulty case. This latter result can give the engineer additional insight into the system’s behaviour. It could for instance provide knowledge about the detectability of different fault classes and fault types (models), as described in section 3.2.

Condition monitoring is also not only about detecting that something is different or has happened; rather, one also needs a confidence measure or some other way to reason about how certain the drawn conclusion is. Thus, condition monitoring is not illustrated by a single light bulb; at least a 2-dimensional output is needed. In this work, a number of different ways to create this 2-dimensional output have been investigated, all centred around using combined measures such as combinations of the quantisation error, the classification, and different measures derived from the accumulated excitation (training) levels of the neurons.

That the same principles used to study the confidence of the conclusion could also be used to study the overall condition of the SOM-lattice in advance is discussed towards the end of chapter 6. It is especially emphasised in conjunction with over-training (table 6.5) and the interpretation of the words high and low together with both q.e. and the accumulated training. Thus, it is argued that this use of the SOM has some inherent properties that are valuable when the over-training phenomenon is studied.

That self-organising maps fit as a condition monitoring principle in the vision described in chapter 1.4 is in particular supported in appended paper [V]. In that paper, it is discussed how a relatively large part of the design process could be automatised.

Model-based approaches, as opposed to data-driven ones, certainly have their advantages and are normally to be preferred, that is, if a suitable model can be developed. However, not all systems lend themselves to the model-based approaches and, thus, data-driven approaches are necessary. The marriage between model-based and data-driven approaches opens up new possibilities. Here, the estimated parameters and results from the model-based methods could be used, possibly also with their inputs, as inputs to a classification scheme, such as the self-organising map. Using such a scheme, it is quite likely that unknown faults will form detectable patterns in the SOM, and thus be detectable even though they are not part of the set of known faults in the model.

Of interest for further investigation are further studies of the combined use of SOMs and model-based condition monitoring.
The papers appended here concern condition monitoring of fluid power systems, primarily pneumatic systems. The order of the papers is not strictly chronological, but rather organised according to a logical line of development. The first paper, [I], introduces the som and also mentions quite a few of the ideas further investigated in the following papers. Papers [II] to [IV] concern condition monitoring of a pneumatic system and transitions from dealing only with unknown faults [II], over known faults [III], to also deal with combinations of known and unknown faults [IV]. In paper [V], further, general system knowledge is introduced in a preprocessing step.

**Paper I**

**Self-Organising Maps for Illustration of Friction in a Pneumatic Cylinder**

Here neural networks, in the form of self-organising maps, are introduced. First, the friction in a pneumatic rod-less cylinder is estimated from measurements. A self-organising map is trained to replicate these estimates based on the system states. Later, the SOM is used to illustrate friction as a function of the states. Other possible uses for the trained SOM, such as condition monitoring, are briefly discussed together with explicit training schemes for the SOM to make it more suitable for different uses.
Paper II

Detection of System Changes for a Pneumatic Cylinder Using Self-Organizing Maps

Here the main focus is on detecting an unknown fault or change to the system. The discussed change is either leaking exhaust valves or a changed mass load. Two principles for detection are studied, the measure normally considered to be best for detection of unknown faults, the quantisation error, and a new measure based on the difference in accumulated training/excitation level in the neuron and a reference neuron.

Paper III

Self-Organising Maps for Monitoring Pneumatic Systems

This paper is primarily concerned with the application of detecting known faults, although unknown faults are also discussed. The detection of known faults is performed using a gradual classification: the class of each neuron is determined using the normal training procedure of the SOM, thus a gradual classification is achieved. To make this work, the splitting of the feature vector into an independent, $x^{ind}$, and a dependent, $x^{dep}$, part is necessary. The suitability of different sets of sensor signals (features) is also discussed.

Paper IV

Condition Monitoring of Pneumatic Systems Using Self-Organising Maps

Here, the detection of both known and unknown faults is further discussed, in particular how the interpretation could be improved by using multiple measures. Also, a case with multiple faults, both known and unknown, occurring simultaneously is used to prove the capabilities of the SOM and the need for either knowledge of how the SOM-algorithm works or a good support system in order to be able to draw definite conclusions from the outcome.

There is also a discussion on how to use the accumulated training/excitation levels to raise alarms using a CUSUM-test.
Paper V

Self-Organising Maps for Change Detection in Hydraulic Systems

In this paper, the detection of unknown faults in a hydraulic position servo is discussed from the point of view of the valve manufacturer. Two faults are simulated in a test rig and two approaches to detection are discussed. One of the approaches is the same as in previous papers, direct use of measured signals, and in the second approach, an ARX-model is estimated and its parameters are used as the features. The performance of these approaches, both on single and multiple faults of different sizes, is evaluated in the paper.
References


