A Study on Condition-Based Maintenance with Applications to Industrial Vehicles

Anna Wigren
Abstract

A Study on Condition-Based Maintenance with Applications to Industrial Vehicles

Anna Wigren

The company CrossControl develops display computers for control systems in industrial vehicles which operate in rough environments. Currently, the system can detect and diagnose different faults but CrossControl would also like to predict upcoming failures by using Condition-Based Maintenance (CBM). CBM is a cost effective maintenance strategy where the health condition of the system is monitored and maintenance is only performed after a degradation in performance has been observed.

This thesis work aims to investigate the possibilities of implementing CBM on CrossControl's system by studying the theory behind CBM and associated methods and by analysing real data from the system of one of CrossControl's customers.

The results presented in this thesis consist of two literature studies and one case study. The first literature study introduces different types of maintenance and gives a detailed explanation of CBM. The second literature study contains a collection of methods used to estimate the Remaining Useful Life (RUL) of the system, which is an important step in CBM. The case study considers the twistlocks of Bromma Conquip's spreader system and serves as an example of how CBM can be used in practise and exemplifies some difficulties that can be encountered when implementing CBM. Finally, a discussion of the obtained results and some suggestions for future work and ideas for how CBM can be implemented on CrossControl's system are given.
Populärvetenskaplig sammanfattning

Alla mekaniska system utsätts för slitage vid drift som med tiden medför minskad prestanda och till slut kan göra att systemet helt upphör att fungera. Ett system som går sönder kan medföra stora kostnader för ett företag, både i form av skador som måste repareras och att systemet inte kan utföra sin uppgift under reparationstiden. Haverier är därfor något som bör undvikas och detta kan uppnås helt eller delvis genom att utföra underhåll. Underhåll kan här innebära något så enkelt som att olja rörliga delar, eller mer avancerade processer som att delar av systemet byts ut. Att utföra underhåll medför också en kostnad, bl.a. i form av inköp av eventuella reservdelar och utökad arbetstid för anställda, och det är därfor viktigt att underhållet genomförs på ett effektivt sätt. Den metod som i de flesta fall har störst potential för effektivt underhåll är Condition-Based Maintenance (CBM). Idén bakom CBM är att underhåll endast genomförs då det finns tecken på att systemet är påväg att haverera.

Det här examensarbetet har utförts åt företaget CrossControl som tillverkar displaydatorer för styrsystem i industrifordon åt sina kunder. Idag kan CrossControls system diagnostisera olika typer av fel men företagets kunder efterfrågar även funktionalitet för att kunna förutsöga när de behöver utföra underhåll genom implementation av CBM. Examensarbetet syftar därför till att undersöka vilka möjligheter som finns för implementation av CBM i CrossControls system genom att studera teorin bakom CBM och genom att analysera data från ett av CrossControls kunders system.

Resultaten presenteras i form av två litteraturstudier och en case study. Den första litteraturstudien behandlar olika typer av underhåll och ger en mer detaljerad beskrivning av CBM. Den andra litteraturstudien behandlar olika metoder för att skatta den kvarvarande livslängden för ett system, vilket utgör ett viktigt steg i CBM. Case studyn har genomförts med data från CrossControls kund Bromma Conquip som tillverkar lyftok till kranar. Den visar hur man praktiskt kan gå till väga för att implementera CBM och ger exempel på ett antal svårigheter. Slutligen diskuterar resultaten och ett antal förslag för framtida arbete med CBM i CrossControls system presenteras.
## Contents

1 Introduction .................................................. 3  
  1.1 CrossControl ............................................. 3  
  1.2 Problem description ...................................... 4  
  1.3 Goal and purpose ........................................ 4  
  1.4 Delimitation ............................................. 4  
  1.5 Structure of the report .................................. 5  

2 Literature study: Maintenance Theory ....................... 5  
  2.1 Corrective Maintenance .................................. 6  
  2.2 Time-Based Maintenance .................................. 7  
  2.3 Condition-Based Maintenance ............................. 8  
    2.3.1 The data and feature extraction ..................... 9  
    2.3.2 Diagnostics ........................................ 10  
    2.3.3 Prognostics ........................................ 10  
    2.3.4 Cost optimisation .................................. 11  

3 Literature study: Algorithms for estimation of the RUL .... 13  
  3.1 Physical Models .......................................... 13  
  3.2 Models based on learning ................................ 15  
    3.2.1 Regression .......................................... 16  
    3.2.2 The Wiener process ................................ 17  
    3.2.3 The Gamma process ................................ 18  
    3.2.4 The Proportional Hazards Model .................... 19  
    3.2.5 Neural Networks .................................... 20  
    3.2.6 The Self-Organizing Map ......................... 23  
    3.2.7 The Markov process ................................ 24  
    3.2.8 The Hidden Markov Model ......................... 25  
    3.2.9 Bayesian Networks ................................ 27  
    3.2.10 Stochastic Filter-based methods ................. 29  
      3.2.10.1 The Kalman Filter ............................ 30  
      3.2.10.2 The Particle Filter .......................... 31  
    3.2.11 Fuzzy Logic systems ............................... 33  
    3.2.12 Principal Component Analysis .................... 34  

4 Case study: Bromma Conquip .................................. 35  
  4.1 Bromma Conquip .......................................... 35  
  4.2 The twistlock ........................................... 36  
  4.3 The data ................................................ 37  
  4.4 Analysis of the data .................................... 39  
  4.5 The classification ..................................... 41  
  4.6 Possibilities for prognostics and estimation of the RUL . 44  
  4.7 Main results of the case study and suggestions for future work .. 46  

5 Results and discussion ....................................... 46  
  5.1 Comments on the results presented in this thesis ........ 47  
  5.2 Recommendations for CrossControl ....................... 48  

6 Conclusion .................................................. 49  

7 References .................................................. 50  

A Additional algorithms for estimation of the RUL .......... 56  
  A.1 Statistical Process Control ............................ 56  
  A.2 Other types of Regression ............................. 56  
  A.3 Linear Discriminant Analysis ......................... 56  
  A.4 The Hidden Semi-Markov Model ....................... 57  
  A.5 Expert Systems ........................................ 57
1 Introduction

All industrial systems in operation experience wear in some sense which will cause a degradation in performance over time and which can eventually lead to a failure of the system. A complete failure of the system will cause a downtime when no actions can be performed which can be very costly for a company. In order to keep the system running for as long as possible without breaking maintenance is normally performed. Maintenance can be something as simple as adding some lubrication oil to moving parts or a more complicated process, e.g. replacing some parts of the system. Common to all types of maintenance is that it costs money since it requires extra working hours, possibly spare parts and it might also cause a downtime if maintenance can not be performed while the system is running. To be able to determine when maintenance should be performed it is also necessary to monitor the system using e.g. sensors or inspections and there is an additional cost associated with this. Since maintenance costs money it is in every company’s interest to aim to keep this cost as low as possible. Previous studies have shown that maintenance can be one of the main expenses for a company and it can account for at least 15 percent, but maybe up to as much as 50 percent, of the total cost. It has also been shown that as much as 30 percent of this cost is due to inefficient maintenance methods [1]. In Sweden for example, a study on maintenance cost in ten manufacturing companies presented in [2] concluded that 39 percent of the unavailable time was spent on maintenance, hence a lot can be gained by implementing efficient maintenance strategies.

The most common type of maintenance today is to perform maintenance periodically based on a precalculated time-interval. When the long-time behaviour of the system is known this strategy works well, but it has been shown to be too conservative i.e. it favours performing maintenance more often than is actually necessary [3]. A potentially more effective maintenance strategy is Condition-Based Maintenance (CBM), which is the topic of this thesis. CBM is a form of maintenance where the current condition of a system is analysed in order to predict when maintenance should be performed to minimise the total cost. An earlier study states that only in the US CBM could save 35 billion dollars every year which shows that companies can gain a lot from implementing CBM [4]. Despite this only about 10 percent of the manufacturing companies in Sweden have some type of CBM implemented today [1].

1.1 CrossControl

CrossControl is a company specialised in developing advanced platform solutions for control systems in industrial vehicles which operate in rough environments. The company was founded in Alfta, Sweden, in 1991 and their product range includes display computers, controllers and communication devices for which both the hardware and software are developed by CrossControl. In Figure 1 three examples of products developed by CrossControl can be seen. CrossControl aims to help their customers by designing smart, safe and productive control systems and one step for achieving this is to include efficient methods for maintenance [5].

Figure 1: A display computer (left), a controller (middle) and a communication device (right), all developed by CrossControl [5].

CrossControl has customers in many different industrial sectors including agriculture, cargo, construction, forestry, marine, mining, rail and utility. Some examples of customers include John Deere (forestry), Bromma Conquip (cargo) and Bombardier (rail). Depending on which sector a customer belongs to they have different priorities for their system; for cargo it is important that the system is time efficient and dependable, for forestry it is important to have a long and reliable uptime and for rail the safety is most important. This diversity of customers and their corresponding priorities complicate the design since the products provided by CrossControl must be able to handle all these differing requirements in an efficient way [5].
The control system provided by CrossControl is implemented based on CoDeSys (Controller Development System) and communicates through CAN-buses (Controller Area Network). CoDeSys is a programming environment for software PLC (Programmable Logic Controller) controllers developed by the German company 3S-Smart Software Solutions. It includes both the software PLC and a programming environment where the control system can be set up with inputs and outputs [6]. CrossControl’s own software includes several tools and one of these is a diagnostic platform implemented in CoDeSys called DRE (Diagnostics Runtime Engine) which facilitates the design of a diagnostic system. It can monitor the signals from the control system and take different actions if it detects an abnormality. These actions include sending warnings, setting alarms and saving data values in a database. In DRE the end-users can choose to use one of the already implemented basic diagnostic blocks or to implement their own diagnostic block. Each diagnostic block takes one or more inputs from the control system, performs diagnostics on these and then generates an output which leads either to no action or to one of the previously mentioned actions. The diagnostic blocks are designed in another tool called DABE (Diagnostic Application Builder Environment) [7, 8]. An illustration of the diagnostic system and how information flows between different parts of the system can be seen in Figure 2.

Figure 2: The diagnostic system developed by CrossControl. The arrows indicate the direction of the information flow.

1.2 Problem description

The system used by CrossControl today enables implementation of diagnostics, that is it includes a platform for implementation of methods to detect failures and take appropriate actions when a failure is detected. However, CrossControl’s customers would also like to be able to predict when an upcoming failure will occur in order to perform maintenance or repairs to prevent it. To achieve this more advanced methods for diagnostics and prognostics than those supported by the platform today are required. CrossControl would therefore like to investigate the possibility of implementing CBM on their devices. More specifically they want to gain a better understanding of how CBM works and which methods that are best suited for their system.

1.3 Goal and purpose

This thesis aims to look into the possibility of using CBM to perform diagnostics and prognostics in CrossControl’s system. Through a literature study of relevant books and articles the theory behind CBM and its associated methods for prediction will be examined to determine if it will be possible to implement CBM on CrossControl’s diagnostic platform. If time permits one or more of the methods deemed most suitable for CrossControl’s system will be chosen for test implementation in MATLAB. Data from the system of one of CrossControl’s customers will be used for the test implementation to prove that the concept works for a real system.

1.4 Delimitation

Due to the limited amount of time for this thesis work it was necessary to restrict the work somewhat. The following restrictions were initially made:

- The main focus of the work will be on the theoretical aspects, that is the literature study.
- If time permits a test implementation of one of the studied methods should be done in MATLAB, nothing will be implemented in CrossControl’s actual system.
If it can be obtained, data from one of CrossControl’s customers should be examined for possibilities to implement CBM.

During the thesis work the following restrictions were also made:

- CBM consists of several steps which could not all be covered in detail, hence the focus here will be on the step where the Remaining Useful Life (RUL) of the system is estimated.
- Due to the immense amount of different methods for estimation of the RUL only a subset of these will be covered in the literature study.
- In the literature study each method is explained in a broader sense and a short explanation of how it can be applied for estimation of the RUL is presented. However, for a detailed explanation of how to use it for estimation of the RUL the reader is referred to the referenced works.
- The case study will only look at RUL estimation for one component of the system (the twistlock).
- Only already available data will be considered for the case study, i.e. there is no possibility to choose what is measured.

1.5 Structure of the report

This report contains six chapters. The first chapter gives a motivation to why maintenance is an important concept and why CrossControl is interested in implementing CBM. It also states the goal and purpose of this thesis work. The second chapter is a literature study on maintenance theory. It explains the different types of maintenance and gives a detailed explanation of the different steps in CBM. The third chapter is a literature study of a selection of methods for estimating the remaining useful life (RUL) of a system which is one of the steps in CBM. Both chapter two and chapter three are intended as a reference for CrossControl if they decide to implement CBM. The fourth chapter is a case study where data from Bromma Conquip, one of CrossControl’s customers, is analysed to determine if it can be used for implementing CBM. The fifth chapter contains a discussion of the main results of the thesis, gives some recommendations for how to implement CBM on CrossControl’s system and provides some ideas for future work. In the sixth chapter the main results of the thesis are summarised and some conclusions are drawn.

2 Literature study: Maintenance Theory

Maintenance is performed to minimise or to completely avoid the damage caused by breakdowns (a failure of the system). According to [9] maintenance can be formally defined as ”a combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function”. When discussing maintenance it is important to first define what is considered to be a failure of the system. According to [9] there are two terms which are important to distinguish between; a failure and a fault. A failure is defined as ”the termination of the ability of an item to perform a required function” whereas a fault is defined as ”the state of an item characterized by inability to perform a required function, excluding the inability during preventive maintenance or other planned actions, or due to lack of external resources”. Hence a failure (breakdown) is an event whereas a fault is a state of the system. It is also important to be aware that a failure is not always the result of a badly functioning system, in many cases it can have natural causes. An example of a failure due to natural causes is a bearing which becomes worn out with time no matter the quality of the original product. In cases like this the degradation of the bearing can not be avoided, but the consequences of a failure may be avoided by supervising the system in some way and by performing maintenance at an appropriate time [10].

Breakdowns of a system can be expensive for a company, which is why maintenance should be performed. An unexpected breakdown normally causes a long downtime during which the system can not perform its designated task. The downtime can be due to troubleshooting, repair work and possibly also waiting for spare parts, which results in production losses. An additional cost is also introduced from the possible spare parts and from the extra work hours that may be required for repairing the system. In some cases a breakdown may also be a safety issue, for example if the breakdown results in hazardous emissions. Examples of cases where this might happen is a nuclear power plant or a chemical
plant. Hence, by repairing and maintaining a system to avoid a complete breakdown a company can save a lot of money [11].

Maintenance in itself can also be a big expense for a company, especially if it is performed too often or if it utilises expensive measuring devices. The cost for maintenance consists of two parts; the cost of performing the maintenance and the annual cost of repairs and replacements. The cost of performing maintenance decreases when the time-interval \( T \) between maintenance instances increases, i.e. the cost is lower if maintenance is performed more seldom. The annual cost for repairs and replacements on the other hand increases when the time-interval \( T \) for performing maintenance increases, i.e. if maintenance is performed regularly the cost of spare parts is low whereas if it is performed more seldom the cost will be higher due to a more severe damage. Since these two types of costs associated with maintenance behave differently it is possible to find an optimal time-interval \( T^* \) for performing maintenance which minimises the total cost [11]. This relationship is visualised in Figure 3 where the annual cost is plotted against the time-interval between maintenances. It can be seen that both performing maintenance very often and very seldom results in a high annual cost. Since maintenance can be the main expense for a company it is important to make it as cost effective as possible [1].

![Figure 3: Annual cost of repairs and replacement, cost of maintenance inspections and the total cost for maintenance plotted against the time interval for maintenance inspections [11].](image)

There are two main types of maintenance strategies; Corrective and Preventive Maintenance. Corrective Maintenance is performed after a breakdown whereas Preventive Maintenance is performed before a breakdown. Preventive Maintenance can be further divided into Time-Based and Condition-Based Maintenance, see Figure 4 [1].

![Figure 4: The different maintenance strategies.](image)

### 2.1 Corrective Maintenance

Corrective Maintenance is the simplest and the earliest form of maintenance. It is also called run-to-failure or curative maintenance. These names refer to it being a strategy where maintenance and repairs
are performed after the system has failed. There are no planned maintenance activities which result in a low or no maintenance cost before the breakdown occurs. Since no maintenance is planned the system will not be over-maintained, but when a breakdown does occur the system downtime may be long, both because the breakdown might be more severe and because the breakdown was not planned and hence spare parts may have to be ordered. This results in a high cost after the breakdown, especially if the breakdown occurs at an unfortunate point in time. There is also a risk that the component breaking causes other parts of the system to break, a so called secondary failure [12]. Corrective Maintenance is best suited for inexpensive parts of the system which are not a safety issue and which are not critical for the performance of the whole system. It is also well-suited for parts of the system which are easy to repair or which can not be repaired [9, 13].

2.2 Time-Based Maintenance

Time-Based Maintenance (TBM), also called Periodic Maintenance, is a maintenance strategy built on the assumption that the degradation process is predictable. Hence maintenance is performed periodically based on a precalculated time-interval. The time-interval can be determined from e.g the number of hours in use, the distance covered or some other statistic related to the degradation of the system. TBM aims to slow down the deterioration process by continuously performing maintenance and thereby reduce the probability of failure [14]. The length of the time-interval between points where maintenance is performed is either determined from recommendations from the Original Equipment Manufacturer (OEM) or from estimations based on failure statistics for the system [15, 3].

It is common to use failure curves as a tool to predict when the system or a part of the system may fail and from these estimates an appropriate time-interval for performing maintenance can be determined. In a failure curve the probability of failure is plotted against the age of the system, some common examples can be found in Figure 5. A high value for low ages indicates that the part is likely to break early in its life (Bathtub and Infant Mortality in Figure 5), whereas a high probability towards the end of the curve indicates that the component will break at a later point in time (Bathtub, Wear Out and Fatigue in Figure 5). A constant failure curve implies that a failure is equally likely at all points in time, hence it is not possible to predict when the failure will occur solely based on the age of the part (Random in Figure 5 and also Initial Break-in and Infant Mortality after the first part of the lifetime). The TBM strategy is based on the assumption that the degradation process is predictable, hence it is best suited for processes with a non-constant failure curve, i.e. processes which correspond to the type of failure curves on the left side in Figure 5 [3, 15].

![Failure Curves](image)

Figure 5: Six different types of failure curves.

The TBM strategy reduces the amount of catastrophic failures, but can not prevent them from happening altogether. It also gives an opportunity to plan ahead when ordering spare parts and planning for when the system will be out of operation. According to [14] it is a 12-18 percent more cost effective strategy than Corrective Maintenance. However, this strategy tends to be too conservative for many systems, i.e. too far to the left in Figure 3, which results in a high maintenance cost. When applying the TBM strategy the system will sometimes be maintained or repaired even though it operates normally.
This is especially common for systems where failures occur randomly. The cost of this strategy increases when the system becomes more complex and the demands on quality increase, since it becomes harder to predict the failure behaviour for a system which consists of many parts [3]. According to [16] only around 11 percent of all failures are age-related (follows one of the failure curves on the left side in Figure 5) and hence a TBM strategy would be inefficient in detecting almost 90 percent of all failures. The reliability of this study was questioned due to a data mix-up, but later studies like [17] which states that only 30 percent of all failures are age related and [18] which states that only about 15-20 percent of all failures are age related, confirm that TBM is an efficient maintenance strategy for only 10-30 percent of all failures. For the remaining 70-90 percent another strategy is required which can handle random failures.

2.3 Condition-Based Maintenance

Condition-Based Maintenance (CBM) provides a solution to the two main shortcomings of the Corrective and the Time-Based Maintenance strategies by aiming to eliminate breakdowns completely while keeping the maintenance interval as long as possible [1]. It can reduce or completely avoid any unnecessary maintenance, and thereby the related cost [18]. The CBM strategy is built on the assumption that a failure is a process, not an event and the failure is therefore expected to occur gradually so that it is possible to predict it. Hence the actual condition of the system is monitored in CBM to detect symptoms of an upcoming failure before it occurs and maintenance is only performed when a degradation in performance has been observed [13].

To decide if a failure is approaching a P-F curve, where the performance of the system is plotted against time, may be used to help visualise the degradation process, see Figure 6. The performance measure should be based on one or several parameters or features of the system which indicate whether the behaviour of the system is normal or abnormal (failure approaching) [1]. The point P on the curve is the point where a decrease in the performance is first detected (it may begin before this point). The point F on the curve is the point where a failure occurs. The distance in time between the points P and F is the P-F interval [10]. The length of the P-F interval is critical for whether it will be possible to detect an upcoming failure or not. If condition monitoring is performed, data must be collected at a time-interval equal to at most half the P-F interval to enable certain detection of a failure. CBM aims to maximize the P-F interval by moving the point P to a point as early as possible [1, 10].

The P-F curve is very appealing in theory, but unfortunately it is very hard to construct in practice since a thorough understanding of the system (failure patterns, history, recommendations, operation conditions etc) is necessary to develop the curve. Its main usage is therefore to facilitate the theoretical understanding of the degradation of a system and the CBM concept [12].

In CBM the aim is to perform maintenance at an optimal point in time based on the current condition (health) of the system. Optimality here refers to minimising the total maintenance cost for the company. CBM can be thought of as a process in three steps; data collection, diagnostics/prognostics and cost optimisation. In the first step data describing the current condition of the system is collected. This step is essential for the performance of any CBM algorithm and corresponds to monitoring the current condition of the system by utilising measurements from sensors and human observations [18]. The type of measurements or observations needed for performing CBM varies depending on the monitored system, but some examples of sensing technologies that have been used are vibration measurements, the speed of rotation for rotating parts and analysis of the lubrication oil [1]. More details on the data collection can be found in Section 2.3.1. In the second step (diagnostics/prognostics) the collected data is used
to find a model which describes the degradation of the system. The model can then be used to predict
the point in time when a failure will occur. Diagnostics can detect symptoms of an upcoming failure
whereas prognostics can give an estimate of when the failure will occur [19]. There is an immense
amount of different algorithms which can be used for performing diagnostics and prognostics. Details
on methods for estimating the failure time can be found in Section 3 and further details on diagnostics
and prognostics can be found in Section 2.3.2 and 2.3.3. In the third and last step the optimal time to
perform maintenance is found using the estimated failure time and other parameters such as the cost for
different types of maintenance. Details on the cost optimisation can be found in Section 2.3.4.

According to [20] 99 percent of all failures are preceded by signs of the impending failure, which makes
CBM suitable for many applications. In addition CBM works well even for failures that occur randomly,
as long as they can be detected in advance [13]. Compared to TBM and Corrective Maintenance the
CBM strategy gives a better availability and reliability of the system since accurate predictions of failures
reduce the downtime of the system. It also increases the useful life of the system when components are
utilised for as long as possible. Additionally, the maintenance cost is reduced due to a reduced number of
occasions when maintenance and repairs are performed, reduced overtime for the employees and a reduced
need for storing of spare parts when maintenance and repairs can be better planned [14]. According to
[9] it is possible to save as much as 20 percent of the total cost of maintenance by using CBM.

Despite the many advantages, around 30 percent of all systems do not benefit from implementing
CBM [4]. Depending on what accuracy the measurements must have the sensors and other monitoring
equipment needed for the condition monitoring can be expensive to install. This gives a high initial cost
for implementing CBM which not all companies can afford. CBM is also a quite young and intensive
research area which means that methods used may not be fully developed, especially for performing
prediction. Another potential problem with using CBM is that many of the methods might need data
which includes failures to achieve good results. The system would have to be run until failure to collect
that type of data which is not possible for all systems. It should be noted however that many methods
work well with data that either does not contain any complete failures at all or a mix of normal operation
data and failure data [14]. In the following sections the three steps of CBM are explained in more detail.

2.3.1 The data and feature extraction

The data collected from the system is vital for performing accurate CBM since the data is used both
to develop and validate the models. The performance of the CBM algorithms is strongly dependent on
both the amount of data and its accuracy. If the data does not contain enough information about the
system and its degradation it does not matter how advanced the CBM method for failure prediction is,
no reliable results can be obtained. Therefore the collection of data requires some careful consideration
in order to achieve good estimates on when to perform maintenance.

There are two different types of data which can be used for CBM; event data and condition monitoring
data. Event data is data which contains information on what happened to the system and/or what was
done to fix it. Examples of event data is data about when installation, breakdown, repair or preventive
maintenance occurred. Condition monitoring data on the other hand consists of measurements of param-
eters which in some way can be related to the condition of the system. Examples of condition monitoring
data is data from temperature measurements, vibration measurements and acoustic measurements which
is collected using different types of sensors [19].

The condition monitoring data can be further divided into three groups; value, waveform and mul-
tidimensional data. The value data consists of one single value for each measurement. Examples of the
value type is data from temperature or pressure measurements. For waveform data each measurement is
a time series. Examples include measurements of vibrations and acoustic data. Multidimensional data is
a series of data in several dimensions. Examples of this type is image data such as X-ray images or
thermographs. Depending on the type of condition monitoring data different methods must be used for
extracting significant features from the data [19].

Methods which estimate the correlation, such as Principal Component Analysis (PCA), or trending
is applicable for the value data. The waveform data can be analysed in both the time and frequency
domain. For analysis in the time domain time series models like AR and ARMA may be used. It is
also possible to extract characteristic features like the mean value, standard deviation or the root mean
square value which can then be analysed in the same manner as the value type data. If the analysis is
performed in the frequency domain the Fourier transform may be used to obtain the spectrum of the
signal. The spectrogram or wavelet analysis may also be used if analysis of the frequency behaviour over
time is desired [19].
Condition monitoring data is more commonly used than event data which results in the event data sometimes being forgotten or ignored. However, it is important to remember that the event data is just as important as the condition monitoring data. If they are analysed together it is possible to obtain better results than if just one of them is used. This requires methods that can handle both types of data and examples of such methods are the Proportional Hazards Model (PHM) and the Hidden Markov Model (HMM), see Section 3 for details on these methods [19].

2.3.2 Diagnostics

The aim of performing diagnostics is to get an early warning that the system is not functioning properly. The term diagnostics usually incorporates three actions; fault detection, fault isolation and fault identification. Fault detection, as the name suggests, is to detect that the system is not operating the way it should. Fault isolation refers to locating which part of the system that is causing the fault. Fault identification is to determine the nature of the fault. Diagnostics is a posterior action, and is hence performed after a failure starts occurring [15, 19].

Diagnostics has been applied to maintenance problems for quite some time, hence there has been an extensive amount of research on different techniques for performing it. Lately more focus has been placed on prognostics which is a newer and more powerful technique. However, diagnostics is still useful when the prognostics method fails and it can also be used to improve the prognostic method by preparing event data from the system [19].

2.3.3 Prognostics

The aim of prognostics is to predict when a failure will occur by estimating the progression of the degradation of the system. Hence prognostics is an apriori action which allows for better planning of maintenance and resources. Prognostics is more efficient than diagnostics, but can not replace diagnostics entirely since some failures can not be predicted and the failures that can be predicted can not be predicted with 100 percent certainty [19]. Prognostics is also a newer research area than diagnostics and therefore there has not been as much and thorough research done on this topic as there has been on diagnostics [21].

There are two different approaches for performing the prediction; estimation of the time left before the system fails or estimation of the probability that the system does not fail up until a certain point in time. The first approach, estimating the time left until failure, has two synonymous names; "Estimation of Time To Failure" (ETTF) or estimation of the "Remaining Useful Life" (RUL). The ETTF is used in the international standard [22], but the RUL is the more widely used term. The second approach, estimating the probability that the system has not failed at time \( t \), can be used for all types of systems since it is always useful to know the probability that the system will fail before the next scheduled inspection. However it is especially interesting for systems where a failure will have catastrophic consequences, for example in a nuclear power plant [19]. According to [23] almost all articles on prognostics deal with the estimation of the RUL and therefore the concept of RUL will be used in this thesis.

The formal definition of the RUL is "The length from the current time to the end of the useful life", and it can depend on many different parameters; the current age of the system, the type of environment in which the system operates, the current observed condition etc. Usually the RUL of a system is both random and unknown, hence it must be estimated. The RUL at a certain time \( t \) can be denoted \( X_t \) and, being a random variable, its probability density function (pdf) conditioned on observations of the condition \( Y_t \) is given by \( p(X_t|Y_t) \) [23]. Estimation of the RUL is normally equal to estimation of this pdf since it is important to not only estimate the current value but also to obtain a confidence interval which gives information about the certainty of the prediction. There are two types of uncertainties in the estimated value for the RUL; one is related to the actual prediction and one is related to the threshold value determining which performance is acceptable and which is not. Figure 7 illustrates these two types of uncertainties for a system where the monitored condition is denoted \( Y \). The condition is continuously monitored until "Initial time" and the future values of \( Y \) are then predicted. The uncertainty in the actual prediction is denoted "1st uncertainty" and corresponds to not knowing exactly when the (possibly estimated) threshold value for \( Y \) will be reached. The uncertainty in the threshold value is denoted "2nd uncertainty" and arises when the threshold level for the condition \( Y \) is not known beforehand. Whether a CBM algorithm can produce a pdf of the RUL and thereby an estimate of the confidence interval can sometimes influence the choice of algorithm since not all algorithms yield a pdf of the RUL [21, 24].

There are an immense amount of different algorithms for computing the RUL of a system. A very simple method which can be applied if the system is well-known is to use the P-F curve for the system.
By computing the current value of the performance measure it is possible to determine where on the P-F curve the system currently is. The RUL can then be computed as $t_F - t_C$ where $t_F$ is the time of failure (point F in Figure 6) and $t_C$ is the current time found by comparing the current value of the performance measure with the precalculated P-F curve. As previously mentioned this approach is very appealing in theory but very difficult to use in practise since it is normally very hard to find an accurate performance measure. In practise other methods are used to estimate the RUL of a system. A more detailed description of a subset of these methods can be found in Section 3.

2.3.4 Cost optimisation

The cost optimisation is the final step of CBM. It aims to find the optimal replacement time and/or the optimal time between inspections in order to minimise the total cost for maintenance. The optimisation is performed after the estimation of the RUL. There are many different ways to do the cost optimisation and no uniform theory exists. Therefore three different examples will be introduced in this section to give the reader an idea of how it can be done. Two of these are quite simple examples and one is a more elaborate example using a Neural Network.

There are three different types of costs associated with maintenance. Inspections of the system performed to determine the current state of degradation gives rise to an inspection cost, $C_I$. Replacing a component in the system for which the degradation process has started but before it fails gives rise to a preventive replacement cost, $C_P$. Finally, a failure of the system gives rise to a failure replacement cost, $C_F$. The failure replacement cost, $C_F$, is normally several orders of magnitudes greater than the other types of costs and the preventive replacement cost, $C_P$, is normally greater than the inspection cost, $C_I$. The maintenance process for a system consists of inspections performed with a time interval $T_I$ between them until a preventive or failure replacement occurs. The amount of time between these inspections is crucial for attaining the minimum cost, hence the optimal value on $T_I$ should be determined for the system in addition to the optimal replacement time. This is most commonly done through simulations where the interval is varied and the $T_I$ which gives the minimum cost is chosen [25].

In [26] the optimal replacement time is found by minimising the expected cost $C(t)$ per unit time over the next time interval, given a preventive replacement scheduled at time $t > t_i$. If a minimum for the expected cost is found to occur before the next monitoring point the optimal replacement time is the current time (before the failure occurs). If a minimum within the next time interval is not found then
the system is allowed to continue until the next monitoring time without performing any replacements. The expected cost per unit time to be minimised for each time interval is given by

$$C(t) = \frac{C_F P(t - t_i|Z_i) + (1 - P(t - t_i|Z_i))C_P + iT_I}{t_i + (t - t_i)(1 - P(t - t_i|Z_i)) + \int_{0}^{t-t_i} x_i p_i(x_i|Z_i) dx_i}$$

(2.2)

where \( t_i \leq t \) is the current (i-th) monitoring point, \( Y_i \) is the condition monitoring information at \( t_i \) with \( y_i \) as its observed value, \( Z_i = (y_1 \ldots y_i) \) is the history of observed condition variables up to \( t_i \), \( X_i \) is the remaining useful life at time \( t_i \) and \( p_i(x_i|Z_i) \) is the pdf of \( X_i \) conditional on \( Z_i \). Note also that \( P(t - t_i|Z_i) = P(X_i < t - t_i|Z_i) = \int_{0}^{t-t_i} p_i(x_i|Z_i) dx_i \) which is the probability of failure before \( t \) conditioned on previous observations \( Z_i \). For this optimisation the determination of \( p_i(x_i|Z_i) \) is crucial. This value is computed using one of the algorithms in Section 3.

In [27] cost optimisation for a Neural Network (see Section 3 for details on Neural Networks) is performed given a preset failure probability threshold \( \theta \). The cost optimisation is performed numerically instead of the more common way of using simulations. The condition monitoring data collected from the system is sent through a Neural Network to compute a life percentage from which an estimation of the failure time can be obtained. Based on the estimated failure time the cost optimisation can be performed. After computing the life percentage it is assumed that \( t_m \) is the actual failure time, \( \mu \) is the mean lifetime prediction error, \( \sigma \) is the standard deviation for the lifetime prediction error and \( T_I \) is the length of the inspection interval. For a specific predicted failure time \( t_m \) it is possible to compute a preventive replacement time \( t_{PR}(t_m) \) which corresponds to the inspection time when the failure probability is greater than the preset threshold \( \theta \). The total replacement cost for the actual failure time \( t_m \) is then given by

$$C_T(t_m) = C_{TP}(t_m) + C_{TF}(t_m)$$

(2.3)

where \( C_{TP} \) is the total preventive replacement cost given by

$$C_{TP}(t_m) = \int_{0}^{\infty} f_m(t_m) C_P I_P(t_{PR}(t_m) < t_m) dt_n$$

(2.4)

and \( C_{TF} \) is the total failure replacement cost given by

$$C_{TF}(t_m) = \int_{0}^{\infty} f_m(t_m) C_F I_F(t_{PR}(t_m) \geq t_m) dt_n.$$  

(2.5)

In (2.4) and (2.5) \( f_m(t_m) = \frac{1}{\sqrt{2\pi} \sigma_m} e^{-\frac{1}{2}(\frac{t_m - \mu_m}{\sigma_m})^2} \) is the pdf for \( t_m \), \( I_P(t_{PR}(t_m) < t_m) \) indicates if a replacement was done before the actual failure time \( t_m \) \( (I = 1) \) or not \( (I = 0) \), and \( I_F(t_{PR}(t_m) \geq t_m) \) indicates if a failure occurred before the actual replacement time \( t_m \) \( (I = 1) \) or not \( (I = 0) \). The total replacement time \( T_T(t_m) \) can be computed in a similar way, see [27] for details. The total replacement cost and time can then be used with a degradation model, in the form of a pdf \( f(t_m) \), for the system (a Weibull distribution in this example) to compute the actual replacement cost and the actual failure time given by

$$C_{TA}(t_m) = \int_{0}^{\infty} f(t_m) C_T(t_m) dt_m$$

(2.6)

and

$$T_{TA}(t_m) = \int_{0}^{\infty} f(t_m) T_T(t_m) dt_m.$$  

(2.7)

The expected total replacement cost per unit time can then be computed as \( C_{exp}(\theta) = \frac{C_{TPA}}{T_{TA}} \). The optimal failure probability threshold \( \theta^* \) can then be found from the minimisation problem \( \min C_{exp(\theta)} \) s.t. \( \theta > 0 \) and from the failure probability threshold the minimised cost and optimal replacement time can be computed.

From the above examples it is clear that the cost optimisation can be performed using different techniques. Since there does not seem to exist any uniform guidelines for how to perform the optimisation carefully analysed to determine how to find the optimal time for performing maintenance.
3 Literature study: Algorithms for estimation of the RUL

There are an immense amount of different algorithms for estimating the RUL of a system. Some models are based on laws of physics whereas some are based on statistics learnt from data collected from the system. Based on similarities in the tools and the type of data the algorithms use they can be divided into different groups. However, there is no standard for how to do this and each author uses its own division of the methods for RUL estimation. There are also examples of the same group of methods having several different names. An example of this are the methods which are based on analytical models derived from laws of physics. Some authors call this group model-based while others call it physical models. Methods that are learnt from data (using statistical and machine learning/artificial intelligence tools) are commonly grouped together into data-driven models due to the way they develop a model based on the observed data. Some authors use the group experience-based models for methods which use experience feedback to adjust parameters of a predefined model. Yet another popular group is knowledge-based methods which are methods requiring large databases but no model, similar to problems that would be solved by human specialists. Attempts to group methods together are further complicated by the fact that it is very common to use more than one of the algorithms in an implementation since they all have different advantages and weaknesses. By combining several methods they can complement each other thereby giving better precision and/or reduced complexity [3]. Taking this into account, some authors include a group for combination models, sometimes also called hybrid models.

To illustrate the diversity of the division of algorithms for estimating the RUL into different groups some examples of divisions into two, three or more groups are present below. [28] uses only two groups for their division; physical models and data-driven methods. [29] uses the same division but have a third group for hybrid models. [21] uses three groups; model-based methods, data-driven methods, and experience-based methods. [19] also separates the algorithms into three groups, by splitting the data-driven methods into two different groups; model-based approaches, statistical approaches and artificial intelligent approaches. [3] on the other hand employs a division into four groups; model-based, data-driven, knowledge-based, and combination models. Finally [24] uses a more detailed division into five groups; experience-based prognostics, statistical trending prognostics, artificial intelligence based prognostics, state estimator prognostics and model-based prognostics.

From the above examples it is clear that physical/model-based methods and data-driven methods (sometimes split into two groups) are groups used by all authors. Some also use a finer division into experience-based models, knowledge-based models and hybrid models. In this work only two groups will be used; physical models and models based on learning (corresponding to data-driven used by other authors). The name model-based used for methods based on analytical models is rather peculiar and confusing since all methods used for estimating the RUL are based on some kind of model (analytical, statistical or a model for how to perform learning). Hence the name physical models was chosen to describe the group of models based on laws of physics in this thesis. The group data-driven was also renamed to models based on learning since the name data-driven is somewhat misleading when physical models also might use data to estimate some of the parameters in the model. The common feature of all models in the, by other authors called, data-driven group is that the model is learnt from observations of the system, hence the name models based on learning seemed more appropriate. The further division into the groups experience-based and knowledge-based was a bit unclear and differed between authors. Therefore, since these methods clearly goes under models based on learning, these groups were discarded in this work. Combinations of models (hybrid models) will not be considered in this thesis (except for some examples of applications of the RUL estimation algorithms which may use more than one method) since the focus here is on understanding the principles behind the different algorithms.

In the following sections different methods for estimating the RUL of a system are presented, split into the two groups physical models and models based on learning.

3.1 Physical Models

The physical models are based on an analytical model of the system consisting of a set of algebraic and/or differential equations. The model is derived from laws of physics and explains the complete behaviour of the system, including its degradation. To produce an accurate model of this type specific knowledge about the failure mechanism and other theoretical relations related to the degradation process is required. The degradation process is often represented by one or more variables for which the dynamics are given by a set of parameters which needs to be determined [21, 24]. The parameters are usually real and measurable quantities which have a physical meaning in the system and must be determined from
An important and commonly used feature for physical models is the residual, defined as the difference between the current output from the model $f(x)$, where $x$ is the input, and the measured output of the system $y$. When the system operates normally (with normal noise levels and disturbances) the residual, $f(x) - y$, should be “small” and stay within a predefined interval. When the system starts degrading on the other hand the residual will start deviating outside of the interval, indicating an approaching failure.

The most common way to estimate the RUL of a system when it is described by a physical model is by using trending and a failure threshold. First the parameters of the model are estimated by matching the algebraic/differential equation which describes the system with observations collected from the system. Once the parameters have been estimated the model of the system is fully known and can be used to compute a future trend for one or more of the monitored variables or some other feature which relates to the health of the system. This trend is then compared to a failure threshold, which is commonly estimated using statistical techniques. Based on when the trend of the physical model is estimated to cross the failure threshold the RUL for the system can be estimated as the time from the current point in time until the trend crosses the failure threshold [3, 24]. In Figure 8 estimation of the RUL by trending is illustrated. The black part of the curve corresponds to estimated values for which there are observations and the grey part of the curve corresponds to predicted values.

A few examples of analytical models used for estimation of the RUL are presented below to give a flavour of the area of physical models. Since the physical model is very system dependent (usually a model will work only for a specific system) the reader should be aware that there are many more models than those presented here. Note also that these examples focus on the actual models used to describe the system, once this is known the RUL can easily be estimated as previously explained.

Physical models are often used for describing degradation in the form of structural anomalies, such as cracks and wear [24]. In [31] the Paris-Erdogan equation is considered for estimation of the RUL of bearings. The equation is empirical and deterministic and describes the propagation of any type of fatigue crack (not just for bearings). In the general case it can be formulated as

$$\frac{da}{dN} = C_0(\Delta K)^n$$

where $a$ is the length of the crack, $N$ represents the running cycles, $C_0$ and $n$ are material related constants and $\Delta K$ is the range of the stress intensity factor over one cycle. For bearings it is commonly the surface area and not the length that is important and for this case (3.1) can be formulated as $\frac{dD}{dt} = C_0(D)^n$ where $D$ is the defect area. Modifications of the Paris-Erdogan law for modelling fatigue crack growth in bearings can be found in [32] and for a boom structure on a concrete pump truck in [33]. In [34] the degradation of a gear tooth is considered by modelling the dynamics of the gear box using a set of differential equations and by modelling the crack propagation using the Paris-Erdogan equation. Some other physical models than 3.1 for estimation of RUL for different systems can be found in [35]. The data sets examined there comes from the NASA database, [36], and the RUL is estimated for milling tools, bearings, Li-ion batteries and turbo-fan engines. Different physical laws are presented and tested for each of the datasets, a more detailed description can be found in [35].
There are several benefits related to using physical models (compared with data-driven models). The most important one is perhaps that they are intuitive to use and easy to comprehend since all parameters in the model have a physical meaning. They are also easy to modify by just estimating new parameters if the operation conditions or the load on the system change. This also implies that they can be used for simulating the response of the system for different operation conditions which could be useful for testing the system. Yet another advantage is that the physical model, due to its complete description of the behaviour of the system, can detect unexpected failures that have not yet been observed. This in contrast to data-driven methods which often require an observation of a fault to be able to detect it. It is also true that when the parameters are chosen carefully so that the physical model is very accurate it outperforms any data-driven method [3, 24, 14].

The physical models have some less desirable properties as well. Most real systems are quite complex processes for which it can be very difficult or even impossible to develop an accurate analytical model. If it is possible to develop an analytic model it is usually expensive since extensive experiments and analysis must be performed. The physical model is therefore mostly used to describe very simple systems, such as bearings or gears. In some cases more complex systems can be described either by just modelling the most critical part of the system or by splitting the system into several, more simple subsystems and modelling these and their interactions instead [21, 24, 14, 3].

### 3.2 Models based on learning

Models based on learning are essentially pattern recognition algorithms and therefore they do not require an analytic expression or equation to describe the system. Instead the model is based solely on observed input \((x_i)\) and output \((y_i)\) data, also called a training data set. The learning can be either supervised or unsupervised. In the supervised case both inputs to the system and the corresponding values on the outputs are available in the training data set whereas for the unsupervised case only the inputs are available. Obviously the unsupervised case is much harder to solve, but despite not having an output value these methods can still learn relationships between features and the structure of the problem. Most of the methods covered in this section are supervised, the exceptions are the self-organising map and the principal component analysis which are unsupervised [37].

When learning a model (which in many cases equals estimation of a set of parameters) the method used aim to find the model which in some sense best describes the observed data. One way to do this for supervised methods is to measure how well the estimated model \(f(x)\) fits the true data \(y\). A common measure is to use the Mean Square Error (MSE) given by \(MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2\). However, if the MSE is computed using the training data the achieved value might be misleading for how well the estimated model performs in reality. When computing the MSE of a model a separate testing data set should always be used which contains measurements that was not used to find the model. The reason for this is that the data used normally contains both the desired signal and some noise, so if a model is found which fits the training data perfectly this means that it is also modelling the particular noise sequence in this data set. Hence if the model is used on a new data set the model might perform poorly since it was overfitted to the noise in the training data. Therefore the MSE of an estimated model should always be computed for a test data set. A potential problem is that a test data set is seldom available which means that the test MSE is hard to compute in practise. Instead a common approach is to estimate the test MSE from the training data set. This can be done in several ways, two common approaches is to either set aside a part of the training data as test data or to use crossvalidation [37].

Models based on learning are well-suited to use on systems which are unknown or too complex to develop a physical model for. They are also good at finding unknown relationships in the data which may not be apparent to the human eye. In addition they can handle many different types of data and (some of them) have the ability to transform high dimensional data to a lower dimension [3, 24].

However, most of the learning-based models require a large amount of data and can therefore be hard to use in practise. This is both because they require an efficient data collection and because a lot of data implies a lot of computations and hence a high computational cost for the model. Another complication is that the data must be rich enough to describe the system adequately, it should for example include examples of degradation and failures [24]. These models can usually only handle failures that are contained in the data used for learning, so if a previously unknown failure occurs it may not be recognised since the model has never seen it before and hence can’t know it is an indication of a failure [14]. It might also be a problem that the learnt parameters might not have the same clear physical meaning as the parameters in the physical models have, which makes it harder to interpret and explain the resulting model [21].
3.2.1 Regression

Regression is a widely used supervised statistical technique which examines and models the dependence between variables [38]. For prognostics and diagnostics it is mainly used to analyse trends of condition monitoring variables. It is based on the assumption that the health of the system can be determined using a set of monitored variables and preset threshold value which indicates a normal behaviour. Regression methods are simple and intuitive, they can be seen as the process of fitting a curve to the data points by minimising the distance from the curve to the data points [19, 23].

It is assumed that a training set of \( n \) observations of the input data \( x = (x_1, x_2, \ldots, x_n) \) \( T \), also called the independent variables, exists with a corresponding target vector \( y = (y_1, y_2, \ldots, y_n) \) \( T \), also called the dependent variable. The aim is to find a regression model \( f \) during the training process so that the target \( y \) can be estimated given a new input \( x \) using the model \( f \). Mathematically the regression model can be stated \( y \approx f(\beta, x) = \bar{E}(y|x) \) where \( y \) is the target vector (it will be a matrix if there are several target values for each input), \( f \) is the regression model, \( \beta \) are the parameters of the regression model (to be determined) and \( x \) are the inputs \([39, 40]\).

A number of different types of regression methods exist and they can be both linear and nonlinear [23, 38]. The type of regression is determined both by the shape of the regression function (linear, cubic, exponential etc) and by the type of output variable (binary, only between 0 and 1, continuous etc) [40]. The simplest form of regression is the linear regression where the output of the model \( f \) is continuous and depends linearly both on parameters and inputs according to

\[
f(\beta, x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n = \beta_0 + \sum_{i=1}^{n} \beta_i x_i \tag{3.2}
\]

where \( x_i \) are the inputs and \( \beta_i \) are the regression parameters. If \( x_0 = 1 \) is added to the input vector the regression model can be written as a matrix equation given by \( F = X^T \beta \). It is the regression parameters \( \beta \) that must be estimated to determine the regression model. For the linear regression this is done using the Least-Square (LS) algorithm [38, 39]. The LS algorithm finds the best values for \( \beta \) by minimising the squared residual (error) between the measured data and the model with respect to the parameters. The squared error \( E \) is given by

\[
E = \sum_{i=0}^{n} (y_i - \beta x_i)^2 = \|Y - X^T \beta\|^2 \tag{3.3}
\]

where the second expression represents the matrix formulation of the problem. Differentiation of (3.3) with respect to \( \beta \) gives the normal equation \( (X^T X)\beta = X^T Y \) (matrix version) from which the parameters can be estimated as \( \beta = (X^T X)^{-1}X^T Y \) [39].

A slightly more general case is linear regression where the output depends linearly on the regression parameters, but the relation to the inputs can be nonlinear. This corresponds to extracting features from the inputs which are then used for the estimation. A linear regression of this type is given by

\[
f(\beta, x) = \beta_0 + \sum_{i=1}^{n} \beta_i \Phi_i(x_i) + \epsilon \tag{3.4}
\]

where \( \Phi \) is a nonlinear function of \( x \) and \( \epsilon \) is the error. Since the model is linear with respect to the parameters it can be solved using LS in the same way as for the completely linear case. The normal equation for (3.4) gives an estimation of the parameters given by \( \beta = (\Phi^T \Phi)^{-1} \Phi^T Y \). Since linear regression is the simplest type of regression there are some shortcomings including that the relationship between the variables must be linear and that the LS estimation algorithm is sensitive to outliers. Also, overfitting is common if the model is too complex in relation to the amount of data used for the estimation. Despite these shortcomings linear regression is very often the first method to try when modelling a system for the first time because of its simplicity [39, 40].

Another popular type of regression is logistic regression which is used for classification (the output belongs to one out of a number of classes). It is most commonly used for binary classification (the output can only take two different values). The aim is here to find the probability of an event (dependent variable) given the input variables. This probability can then be used to determine which class that is most probable for the observed data [37, 40]. For the binary case (the output belongs to class "0" or class "1") the logistic regression uses a model structure which is different from the linear regression, given by

\[
f(x) = P(y = 1|x) = \frac{1}{1 + e^{-g(x)}} \tag{3.5}
\]
where \( P(y = 1|x) \) is the probability that the observed output belongs to class "1" given the input \( x \) and \( g(x) \) is a logit function. The logit function can have different shapes but a common version is \( g(x) = \sum_{i=0}^{k} \beta_i x_i \). Just like for the linear regression case it is the parameters \( \beta \) which should be estimated. For the logistic regression this is commonly done using the Maximum Likelihood method which maximises the probability (likelihood) of the observed values given the model [41]. The likelihood function for the regression model in (3.5) is

\[
L(\beta) = \prod_{i=1}^{n} \frac{e^{\beta_i \sum_{i=0}^{k} \beta_i x_i}}{1 + e^{\sum_{i=0}^{k} \beta_i x_i}}
\]

(3.6)

and the optimal values of \( \beta \) are found by iteration using for example the Newton-Raphson method [42]. Since the maximum likelihood algorithm is used for the estimation, a lot of data is required to achieve a good performance using the logistic regression [40, 41]. Note that the logistic regression can be extended to cases where the output can belong to more than two classes [37].

In [43] the RUL is estimated using a linear regression model and data from a competition held in 2008. The procedure starts with a division of the data into 6 different operation regimes. The most significant sensors are then chosen for the estimation and the data are collected into "units" for each regime. A linear regression model is fitted to the data from the chosen sensors for each operation regime. The regression models have a "health index" as their output. Testing data units are then used with the models to produce a health index. The distance \( d \) between the estimated health indices (model output) and the actual health indices are then computed and the distance is used to estimate the RUL for each test unit. The total RUL for the system is then computed as a weighted sum of the estimated RUL values, where the weights depend on the distances \( d \). Details on how to perform each of these steps are found in [43].

Another example of RUL estimation using regression can be found in [41], where logistic regression is used together with an ARMA model to predict the RUL for an elevator door. The logistic regression model is used to map the features extracted from the input data into a failure probability graded from 0 to 1. The failure probabilities are then trended using an ARMA model to estimate the RUL.

There are an immense amount of different ways to perform regression and only the most common ones have been considered here. Some special types of regression models will be considered in the following subsections. These are the Wiener process and the Gamma process. The Neural Network is also closely related to regression and can be seen as a type of generalised, nonlinear regression which will turn into a linear regression in the most simple case.

### 3.2.2 The Wiener process

The Wiener process, also called Brownian process, is a type of regression model with special properties since it includes terms for drift and diffusion. It is well suited for processes that have Gaussian noise and a bidirectionally varying degradation with respect to time [23]. The process model was originally developed to describe the movement of particles (in physics). These particles can move around back and forth with small fluctuations. When the Wiener process is applied to estimate the RUL this implies that the degradation can both get worse and improve.

Mathematically the Wiener process \( Y(t) \) is given by

\[
Y(t) = \lambda t + \sigma B(t)
\]

(3.7)

where \( \lambda \) is a drift parameter, \( \sigma > 0 \) is a diffusion coefficient and \( B(t) \) is the standard Brownian motion [23]. The RUL at a certain time \( t_i \) for a Wiener process is defined as

\[
X_{t_i} = \inf\{x_{t_i} : Y(t_i + x_{t_i}) \geq \theta \mid Y(t_i) < \theta\}
\]

(3.8)

where \( Y(t_i) \) is the process output given in (3.7) at the current time \( t_i \) and \( \theta \) is the threshold for normal operation. This means that the RUL for a Wiener process is defined as the time from the current time until the process \( Y \) crosses the threshold \( \theta \). The Wiener process have been extensively researched and there exists a closed form expression for the pdf for the RUL in the form of the inverse Gaussian distribution given by

\[
f_{x_{t_i}}(x_{t_i}) = \frac{\theta - Y(t_i)}{\sqrt{2\pi x_{t_i}^2 \sigma^2}} \exp \left( \frac{(\theta - Y(t_i) - \lambda x_{t_i})^2}{2x_{t_i}^2 \sigma^2} \right).
\]

(3.9)
where all variables are as defined in (3.7) and (3.8) [23].

The Wiener process has some interesting properties which must be considered when deciding if it is appropriate for modelling the system at hand. Firstly, the Wiener process does not degrade monotonically since the level of degradation can both get worse and improve. Hence it can not be used to model a monotonically decreasing process. Secondly, the Wiener process is time homogenous and hence the degradation is equally fast at all times. This is not true for all real processes. Thirdly, the Wiener process obeys the Markov property which means that the process is assumed to move to the next state based solely on the current state, ignoring all previous observations. This is often, but not always true for real processes. Fourthly, looking at (3.7) it is clear that the variance of the noise term is proportional to the time. This is a strong assumption which not many processes fulfill. Finally, it is assumed that the mean degradation path is either linear or can be linearised. Since many processes are nonlinear this could be a limitation [23].

In [44] the Wiener process is used for estimation of the RUL for a Lithium battery. The data used for the estimation comes from [36]. A Weiner regression model of the type in (3.7) is used where the parameter \( \lambda \) is considered to be random due to noise which makes the RUL estimation slightly complicated. The fixed parameters of the model are estimated offline using Maximum Likelihood based on an earlier implementation. The random parameter \( \lambda \) is estimated using a modified version of the Maximum Likelihood algorithm in two steps (details on this can be found in [44]). When the parameters have been determined a slightly modified version of the inverse Gaussian distribution in (3.9), which includes the mean and standard deviation of \( \lambda \), is used to estimate the pdf of the RUL.

Advantages of modelling using the Wiener process includes that the Wiener process has been studied for a long time. Its properties are therefore well-known and it has been used extensively in many different areas. The closed form expression for the pdf is also a clear advantage of modelling the RUL using the Wiener process. However, as for all types of regression the Wiener process requires quite a lot of computations which makes it unsuitable for online implementation [23].

### 3.2.3 The Gamma process

The Gamma process is based on regression, like the Wiener process, but it uses the gamma distribution instead of parameters for drift and diffusion. The Gamma process is monotonic which implies that the degradation can only evolve in one direction. Therefore it is commonly used to model degradation which constantly decreases the performance and occurs gradually. Examples of this type of degradation includes wear, corrosion and cracks [23, 45].

A Gamma process, \( \{Y(t), t \geq 0\} \), has three properties: 1) For a given time interval \( \Delta t = t_i - t_{i-1} \) the increment \( Y(t_i) - Y(t_{i-1}) \) has a gamma distribution \( \Gamma(\nu(t_i) - \nu(t_{i-1}), \alpha) \), where \( \nu(t) > 0 \) is a shape function and \( \alpha > 0 \) is a scale parameter. 2) For any set of disjoint time intervals the increments \( Y(t_i) - Y(t_{i-1}) \) are independent random variables with the distribution given in 1). 3) For almost all processes with the above properties it is also true that \( Y(0) = 0 \) [23].

The RUL for a Gamma process can be defined as

\[
X_{t_i} = \inf\{x_{t_i} : Y(t_i + x_{t_i}) \geq \theta \mid Y(t_i) < \theta\} = \{x_{t_i} : Y(t_i + x_{t_i}) \geq \theta \mid Y(t_i) < \theta\}
\]

(3.10)

where \( \theta \) is the threshold for normal operation and the last equality comes from the monotonic property of the Gamma process [23].

In [46] the Gamma process is used to estimate the RUL of a Lithium battery using the data provided from [36]. The RUL is estimated in two steps. In the first step the degradation is modelled as a combination of a Gamma process and additional Gaussian noise. The true increment is assumed to follow a gamma distribution of the form \( \Delta y_{true} \sim \Gamma(\alpha, \lambda) \) where \( \alpha \) and \( \lambda \) are parameters, and the noise increment is assumed to have a normal distribution of the form \( \epsilon(t + \Delta t) - \epsilon(t) \sim N(0, 2\sigma^2) \) where \( \sigma \) is a parameter. The parameters \( \alpha, \lambda \) and \( \sigma \) are estimated from the data using the method of moments (see [46] for details). In the second step the degradation is extrapolated forward in time using a Gamma process (no noise) with the parameter values as estimated in the first step. The RUL can then be estimated as the time from the current measurement until the extrapolated degradation curve crosses a preset threshold. More examples on how to use the Gamma process for estimation of the RUL can be found in [45].

The monotonic property makes the Gamma process suitable to use for modelling degradation in processes where wear, cracks, corrosion and fatigue are common causes of failure, since all occur gradually in small steps. Apart from the monotonic property there are some other advantages of using the Gamma process. There is a somewhat physical meaning to the involved parameters, hence the model is easy to
understand. Also, the mathematical computations necessary are quite simple. An interesting property of the Gamma process is that the sum of gamma distributed increments is also gamma distributed which can be utilised to determine the optimal inspection and maintenance intervals [23].

There are also some disadvantages. The most important one is that the Gamma process can only model strictly monotonic degradation. There are many processes which do not degrade monotonically and the Gamma process can hence not be used to describe these. Just like the Wiener process the Gamma process fulfills the Markov property, which means that it moves to the next state only based on the current state, ignoring all previous states. As mentioned for the Wiener process not all real processes fulfill this criteria. Additionally the noise involved in the Gamma process must also have the gamma distribution [23].

3.2.4 The Proportional Hazards Model

The Proportional Hazards Model (PHM) was introduced in 1978 by [47] for survival analysis and was initially mainly used in biomedicine. Since then it has been used in many fields of research, lately also for maintenance purposes. It belongs to the broader class of covariate-based hazard models and is the most commonly used of these [23, 48]. The PHM is a combination of covariates (a variable that might influence the outcome) and a time-dependent hazard function. An important feature of this model type is that it can easily handle both event and condition monitoring data which is not the case for all models.

The complete PHM model consists of two parts and is given by \( h(t|z(t)) = h_0(t)\psi(z(t), \beta) \), where \( h_0(t) \) is the baseline hazard rate function which only depends on time, and \( \psi(z, \beta) \) is a positive function which takes the effects of the covariates \( z \) into account. The \( \beta \) is a vector of regression parameters, hence the PHM in some sense belongs to the regression type of models [23, 3].

The baseline hazard rate function, \( h_0 \) accounts for the age of the system (time-dependence) and it can be either parametric or non-parametric. The are many baseline hazard rate functions to choose from, but the most commonly used is the Weibull hazard function given by \( h_0(t) = \frac{\gamma}{\eta} \left( \frac{t}{\eta} \right)^{\gamma - 1} \) where \( \gamma \) and \( \eta \) are the scale and shape parameters [19, 23].

In Cox’s model the positive function \( \psi \) is an exponential of multiplied regression parameters and covariates according to \( \psi(z, \beta) = \exp(\beta_1 z_1 + \ldots + \beta_n z_n) \) where \( n \) is the number of covariates. A covariate is a factor that is somehow involved in causing the degradation. They change stochastically and influence the lifetime of the system, which means that they are important to include when modelling the system. The type of covariates included in the model varies depending on the system but some examples of commonly used covariates are: temperature, the speed of rotation and event data such as previous maintenance times. Normally all condition monitoring data can be seen as a covariate [23, 19].

Based on the PHM a reliability function can be constructed according to \( R(t\{Z(t): 0 \leq \tau \leq t\}) = e^{-\int_0^t h(\tau, Z(\tau))d\tau} \). Once known, the reliability can be used to estimate the RUL of the system [49]. The RUL for a PHM model is formally given by

\[
X_t = \{x_t: T - t | T > t, z(t)\}
\]

(3.11)

where \( T \) is the total lifetime and \( z(t) \) is observations of all covariates up to time \( t \) [23]. The pdf of the RUL can then be formulated as

\[
f_{X_t}(x_t|z(t)) = \frac{f(t + x_t|z(t))}{R(t|z(t))} = h(t + x_t|z(t))\frac{R(t + x_t|z(t))}{R(t|z(t))}
\]

(3.12)

[23].

When learning the PHM it is common to use maximum likelihood estimation to find the parameters using event and condition monitoring data. A likelihood function can be constructed according to

\[
l(\gamma, \eta, \beta|D) = \prod_{j \in D_F} h(t_j|z(t_j)) \prod_{k \in D_F \cup \Omega_C} R(t_k|\{z(\tau): 0 \leq \tau \leq t_k\})
\]

(3.13)

where \( D \) is the available data, \( \Omega_F \) is the failure set and \( \Omega_C \) is the survival set. The expression in(3.13) can be transformed to a loglikelihood and the parameters \( \gamma, \eta \) and \( \beta \) can then be estimated using maximum likelihood. To estimate the RUL the future degradation features \( \{\tilde{z}(\nu): t \leq \nu \leq \infty\} \) must first be predicted. This can be done using a time series method, such as AR or ARMA, or more generally by assuming that the features are given by some nonlinear function \( g \) according to \( z(t) = g_i(t) + \epsilon \), where \( \epsilon \) is normally distributed noise with zero mean. The RUL can then be approximated as

\[
RUL \approx \int_t^{\infty} e^{-\int_t^\tau h(\nu, \tilde{z}(\nu))d\nu} d\tau
\]

(3.14)
In [49] the RUL of bearings was estimated with a PHM using two covariates based on features from the data. The first feature is the Root Mean Square (RMS) of the signal, which corresponds to the energy of the signal, and the second feature was the kurtosis, which corresponds to the peakedness of the signal. To perform online prediction of the RUL the future degradation features were first predicted using a second order polynomial and then (3.14) was applied to estimate the RUL of the bearings.

A clear advantage of the PHM is that it can handle both event and condition monitoring data. It is important to use both types of data since the time of failure may be related to both the events happening to the system and the condition of the system. Another advantage is that the effect of a specific covariate on the hazard function can be evaluated using this method, since it is easy to relate the PHM to time-varying covariates [19, 23].

There are also a few disadvantages with the PHM models. Firstly, a second process, usually a Markov chain, must sometimes be used to model the covariate process when these evolve stochastically which gives heavier computations. Secondly, there is a proportionality between the different covariates which arise from the definition of the PHM when the different covariates share the same baseline function. Thirdly, large amounts of both failure and condition monitoring data is necessary since the parameters of the baseline function and the regression parameters $\beta$ must normally be estimated at the same time due to a multiplicative effect [23].

### 3.2.5 Neural Networks

A Neural Network is a general and flexible model consisting of many simple elements called neurons which are connected in a complex network where each connection is represented by a weight. Each neuron takes a number of inputs and performs some calculation which then produces an output [25]. There are three different types of layers; the input layer, the hidden layer and the output layer. Each of these layers have a set of neurons in them [3, 19, 25]. If the input to the neuron comes from an external data source the neuron belongs to the input layer. If the input data comes from another neuron and its outputs are connected to other neurons it belongs to the hidden layer. The neuron or neurons which give the final output of the network belong to the output layer. The Neural Network learn the underlying function by adjusting the weights between neurons to match the observed inputs and outputs [3, 19, 25]. In Figure 9 (left) an example of a simple Neural Network with one input layer, one hidden layer, and one output layer can be seen. The circles represent the nodes and the arrows show the connections (weights) between the different nodes. In the example to the left in Figure 9 the network has 3 inputs, 4 hidden nodes and 2 outputs.

![Neural Network Diagram](image.png)

Figure 9: The three types of layers in a Neural Network (left) and an example of the simplest type of neuron, a perceptron, with inputs $x_i$ and weights $w_i$ (right).

The simplest type of neuron is the perceptron neuron. In this type of neuron the inputs are multiplied with a set of weights and are then added to determine the output. If the sum is greater than some threshold $\theta$ the output 1 is given, if not the output 0 is given. Figure 9 (right) shows an example of the perceptron neuron. Mathematically the output is determined by

$$\text{output} = \begin{cases} 1 & \sum_{i=1}^{N} w_i x_i > \theta \\ 0 & \sum_{i=1}^{N} w_i x_i < \theta \end{cases}$$

(3.15)
where \( N \) is the number of inputs, \( x_i \) is input \( i \) and \( w_i \) is the weight for input \( i \). During the training both the threshold and the weights are determined. This neuron has the limitation that it produces a binary output. This works for some systems but for many real systems it is desirable to have more than two possible values for the output. The binary output problem can be avoided by adding additional thresholds, but this gives a more complicated learning process since the best possible set of thresholds must then be learnt [25].

To avoid these problems completely a more advanced neuron which uses an activation function can be used. The activation function, \( \sigma(z) \), is applied to the weighted sum and threshold according to

\[
\text{output} = \sigma\left( \sum_{i=1}^{N} w_i x_i + \theta \right)
\]

where all variables are the same as in (3.15). A commonly used activation function is the sigmoid function \( \sigma(z) = \frac{1}{1+e^{-z}} \). The sigmoid function can take all values between 0 and 1. If negative values on the output is desired an alternative is to use the hyperbolic tangent \( \sigma(z) = \frac{e^z-e^{-z}}{e^z+e^{-z}} \) as the activation function. These are just two examples of activation functions, there are several others which all have their own areas of usage [25].

A frequently used Neural Network is the Feed-Forward Neural Network and as the name indicates it is only possible to go ”forward” in this network. The Neural Network in Figure 9 is a simple Feed-Forward Neural Network and the output of this network is constructed as follows. The inputs \( x_1, x_2, \ldots x_N \) are first combined to form the activation

\[
a_j = \sum_{i=1}^{N} w_{ji}^{(1)} x_i + w_{j0}^{(1)} , \quad j = 1 \ldots M
\]

where \( w_{ji}^{(1)} \) are the weights for the inputs and \( w_{j0}^{(1)} \) are the biases (which correspond to the threshold for the perceptron). The index \( i \) indicates which input neuron the weights are applied to and the index \( j \) indicates which neuron in the hidden layer that the activation belongs to. The superscript (1) refers to the first layer of the network after the input layer (the input layer is normally labelled (0)). These activations are then used to form the output of the hidden layer given by

\[
z_j = h(a_j)
\]

where \( h(\cdot) \) is a nonlinear activation function, for example the sigmoid function mentioned earlier. An output activation can then be formed from the hidden layer outputs \( z_j \) in (3.18) according to

\[
a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}
\]

where index \( k \) indicates which neuron in the output layer the activation belongs to. Note that the superscript in now (2) since the second layer of the network (the output layer) is considered. The final output of neuron \( k \) is

\[
y_k = \sigma(a_k).
\]

During the learning phase the weights and biases of each neuron in the Neural Network are updated to make the output of the network as close to the observed output as possible for a given input. Before the learning process starts the Neural Network is initialized with random weights and biases [25]. The update of these weights are then chosen to minimize the error function given by

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2
\]

where \( y \) is the vector of outputs given \( x_n \), which is the vector of inputs, and \( w \), which are the weights. \( t_n \) is the vector of target outputs and \( N \) is the number of training points [39]. The optimization of this error function with respect to the weights can be performed using several methods, two examples are backpropagation and Levenberg-Marquardt [50]. The backpropagation algorithm propagates the
output error from the output layer backwards in the network using gradient descent [25, 29] whereas the Levenberg-Marquardt algorithm instead uses a mix between gradient descent and the Gauss-Newton algorithm [51]. The backpropagation algorithm is performed in two main steps. In the first step the derivatives of the error with respect to the weights are computed. This step is where the error is propagated backwards through the network from the output. In the second step the computed derivatives are used to update the weights. Details on how to perform these two steps can be found in [39]. Details on the Levenberg-Marquardt algorithm can be found in [52]. There are a few difficulties regarding this optimisation. Firstly, since the Neural Network is nonlinear, the optimisation problem will be a non-convex problem so the "optimal" value found for the weights can be just a local minima [39]. The optimal value found for the weights can depend on the initialisation of the weights and hence, depending on the problem, the training should be performed more than once so that the set of weights which results in the smallest error can be chosen. This is important since the performance of the Neural Network depends on the chosen weights [29]. Secondly, over-parameterisation is common, so it is important to use a few of the training sets for validation of the model to determine when the training should be stopped [50].

Neural Networks can be used for estimation of the RUL. The information collected about the system, such as the age and condition monitoring variables, is used as input to the network. For the output from the network there are several approaches. The most commonly used approach is to have one or more variables indicating the condition of the system as the output of the Neural Network. The output is compared to a threshold to determine how much time is left until failure. This means that a threshold must be determined which can sometimes be difficult in practise. An example of another approach can be found in [53] where a Feed-Forward Neural Network is used to estimate the distribution of the RUL for bearings. A vector of frequency amplitudes is used as input to the Neural Network and the output describes how long the bearing has been operating since the first defect occurred. The output is then further processed to determine the distribution of the RUL. Yet another approach for estimating the RUL for rotating equipment can be found in [51, 50]. The method uses the age of the equipment and two condition variables for the current and previous point in time as inputs to the Neural Network. The previous time point is used to include the rate of change of the variables. The output from the Neural Network is the life percentage $P_i$ which is defined as $P_i = \frac{t_i}{T_f}$ where $t_i$ is the current age and $T_f$ is the failure time. Hence the failure time is required for this approach which can be a problem since systems are usually not run until failure. When the Neural Network is trained the RUL can be estimated as $RUL = \frac{1}{P_i}$. In [54] a way to determine the distribution of the RUL, which is required for accurate cost optimisation in CBM, is also developed.

There are many different types of Neural Networks, apart from the ones mentioned earlier the Neural Network can also be combined with different tools which give rise to new methods such as: 1) Dynamic Wavelet Neural Network (DWNW) which utilizes wavelets to extract features [3]. 2) Polynomial Neural Network (PNN), which is used for generic fault detection, isolation and estimation. It can utilise the high order of the data and cross-coupled nonlinearities [3]. 3) Cascade Correlation Neural Network (CCNN), which can be used online since it does not require an initial structure or a preset number of nodes [19].

The Neural Network is well-suited for problems where a lot of data is available, but where the underlying system might be difficult to explain or might even be unknown [24]. There are several reasons for this. The Neural Network is a universal approximator, hence it can approximate any continuous function in $\mathbb{R}^p$ arbitrarily good (if a large enough number of layers is used). This is due to the complex layer structure of the Neural Network which enables it to model complex nonlinear functions with several inputs and outputs only based on the information in the data [3, 19]. The Neural Network can also fuse inputs together efficiently which gives a faster computation [3]. Additionally the Neural Network is an adaptive network since it learns from examples and like all data-driven methods it can capture relations in the data which are unknown to the user. They are also robust and can handle noisy data [24].

Despite its desirable properties there are some less appealing properties of the Neural Network. Like many data-driven methods it is a typical black-box model which means that the weights for the different neurons do not have a physical meaning, they are just parameters which makes it hard to understand the model of the system from the Neural Network model [3, 19]. If a big Neural Network is required it may become hard to train since the choice of the number of hidden layers and the number of nodes for each layer becomes more difficult to select [3]. A big Neural Network may also require heavy computations during the learning phase which can be a problem. When the learning is completed however, the Neural Network is fast [14]. A severe limitation of the Neural Network is that since it is not a statistical model it is hard to estimate the uncertainty in the prediction of the RUL, the method normally only gives a single value on the RUL. But as mentioned earlier several solutions to this problem have been presented.
3.2.6 The Self-Organizing Map

The Self-Organizing Map (SOM), also called Kohonen Network, is a method for visualising and analysing nonlinear high-dimensional data by transforming it into a lower dimensional space \[55, 56\]. The aim of this method is to acquire an understanding of the unknown process which has generated the observed data series \[14\]. It is a special type of unsupervised Neural Network which only has one two-dimensional layer of nodes, either in a square formation or in a hexagonal formation, see Figure 10 \[3, 19, 56\]. Every node has a specific topological position in the network given by coordinates \((a, b)\) and a vector of weights or models, \(w_i\), which is updated during the learning process to optimally match the input data. Note that there are no connections between the nodes in the SOM unlike the case for the Neural Network \[56, 57\].

![Figure 10: A SOM with a square formation (left) and a SOM with a hexagonal formation (right) \[58, 59\].](image)

The SOM learning is performed iteratively and starts by the initialisation of the weights of the nodes. The weights, \(w_i(t) = (w_{i1}(t), \ldots, w_{in}(t))^T\), where \(t = 1, 2, \ldots\) is the iteration index, can be randomly initialised, but the learning will be faster if they are initialised as vectors from a subspace spanned by the two largest principal components of the inputs. After initialisation a vector of inputs, \(x(t) = (x_1(t), \ldots, x_n(t))^T\), from the training data are selected. The selected input vector is compared with all weights of the SOM and the weight which most closely resembles the input vector is chosen as the best matching unit (BMU). The BMU, with weight vector \(w_B\), is found using some distance measure, for example the Euclidian distance given by

\[
d_i = \sqrt{\sum_{j=1}^{n} (x_j(t) - w_{ij}(t))^2}
\] (3.22)

where \(d_i\) is the Euclidian distance from the input \(x(t) = (x_1(t), \ldots, x_n(t))^T\) to weight vector \(w_i(t) = (w_{i1}(t), \ldots, w_{in}(t))^T\). After the BMU has been found the weight of the BMU and all other nodes in its neighbourhood will be updated. The neighbourhood of a node is defined as all nodes within a certain radius from the node. The radius of the neighbourhood decreases with each iteration so that in the end it will only include the BMU. Hence the number of weights that are updated in each iteration will be big in the first few iterations but will then gradually decrease to one. A commonly used function for the decreasing radius is given by

\[
s(t) = \sigma_0 e^{-\frac{t}{\lambda}}
\] (3.23)

where \(\sigma_0\) is the initial radius and \(\lambda\) is a time-constant \[57\]. For nodes in the neighbourhood of the BMU the weights are updated according to

\[
w_i(t+1) = w_i(t) + \alpha(t) h_{B,i}(t) (x(t) - w_i(t))
\] (3.24)

where \(\alpha(t)\) is the learning rate which decreases over time, for example with exponential decay as \(\alpha(t) = \alpha_0 e^{-\frac{t}{\gamma}}\) where \(\alpha_0\) is the initial learning rate and \(\gamma\) is a time-constant. The function \(h_{B,i}\) is the neighbourhood function which makes sure that weights of nodes located close to the BMU are adjusted more than the weights of nodes further away. This process makes weights which are close to each other more similar compared to those further away, since they share a larger amount of similar input data. The neighbourhood function depends both on the distance from the BMU and the number of iterations and can for example have the form of a Gaussian according to

\[
h_{B,i}(t) = e^{-\frac{||r_i - r_B||^2}{\sigma(t)}}
\] (3.25)
where \( r_i \) is the position of the node, \( r_B \) is the position of the BMU and \( \sigma(t) \) is given in (3.23). When all weights have been updated a new input vector is selected and the same steps are performed again [55, 56, 57].

In [60] an example on how to use the SOM for estimation of the RUL for a set of bearings can be found. The SOM is used together with a Feed-Forward Neural Network and the SOM is used for the feature extraction. The ability to handle nonlinear and highly deviating data makes the SOM a good classifier in this case. After the input has been normalised the SOM algorithm is applied. When the training is completed the resulting weights are grouped together into clusters depending on their relative distances. The idea is to train the SOM only using data from normal operation conditions which means that the clusters in the SOM all indicate normal states. An indication of an upcoming failure will be defined as when the input to the SOM is too far away from any of the weight vectors in the SOM. This is done using the Minimum Quantisation Error (MQE) which is defined as

\[
MQE = \| D - w_B \| \tag{3.26}
\]

where \( D \) is the input vector and \( w_B \) is the weight vector of the BMU for the input. If the MQE is higher than a preset threshold an indication of an upcoming failure (or an outlier) has been found. When the data has been classified it can be used to estimate the RUL using the Feed-Forward Neural Network. For this implementation of the SOM it is important to choose the size of the SOM depending on the amount of input data, if the SOM is too big compared to the amount of data the whole SOM can not be identified. The choice of neighbour function is also important for an efficient implementation.

There are several benefits from implementing the SOM. The SOM can be used with many types of input data. As long as it is possible to define the distance between two inputs it is possible to implement the SOM [55]. The output of a trained SOM is a feature map of the input and hence the SOM is useful for classification of an input and for finding and visualising structures in the system’s behaviour [3, 57]. Also, the SOM is well-suited for nonlinear and deviating data and it gives a low dimensional and discretised representation based on the input data [14, 3].

A big disadvantage of the SOM is the high computational cost which degrades the performance of the method due to the fact that it must check all neurons. Also, it requires a lot of data for the training process [14].

### 3.2.7 The Markov process

A Markov model is a multistage deterioration model which utilizes the concept of Markov chains. The system is assumed to pass through a set of states, such as \{"good", "ok", "still working", "failure"\}, during its deterioration. The first state is the state of "normal operation" and the last state is the "failure" state. A transition between two different states \( i \) and \( j \) occurs with a certain probability \( P_{ij} \). In Figure 11 a Markov process for the four states \{"good", "ok", "still working", "failure"\} is visualised. \( P_{ij} \) are the transition probabilities. Throughout its lifetime the system passes through all states unless a random failure occurs which moves the system immediately to the "failure" state (not illustrated in Figure 11). A random failure is normally modelled using a Poisson distribution for which the failure rate varies with the current deterioration state of the system [61].

![Figure 11: An example of a Markov process with four states. \( P_{ij} \) are the transition probabilities.](image)

There are two important assumptions for the Markov process. Firstly the process is a Markov Chain and hence it is memoryless, which means that the future state depends only on the current state. This can be expressed mathematically as

\[
P(X_{t+1}|X_t...X_1) = P(X_{t+1}|X_t) \tag{3.27}
\]
where $X_{t+1}$ is the next state and $X_t$ is the current state. Secondly it should be possible to find the system state immediately from observed condition monitoring data [23, 39].

Mathematically the degradation process, $Y_n$ evolves in a finite state space denoted $\Phi = \{0, 1...N\}$, where 0 is the “normal operation” state and N is the “failure” state. To describe the transition between states a transition probability matrix is defined as

$$P = \begin{pmatrix} \bar{P} & P_0 \\ 0 & 1 \end{pmatrix}$$

(3.28)

where $\bar{P}$ is the transition matrix for transient states $\Phi \setminus \{N\}$, $P_0 = (I - \bar{P})e$ where I is the identity matrix and $e = (1...1)^T$ [23]. Each element in the matrix is given by $P_{ij} = p(X(n) = j | X(n-1) = i)$ where $i, j = 1...N$. The degradation process will be assumed to be monotonic (the system can not repair itself) so all values below the diagonal in the matrix $P$ will be zero since they correspond to transitions from a more deteriorated state to a less deteriorated one which is not allowed. Note also that the 1 in the bottom right corner corresponds to the probability $P_{NN} = p(X(n) = N | X(n-1) = N) = 1$, hence if the process reaches the failed state it stays there [62].

The RUL at time $n$ can be defined as

$$X_n = \inf\{x_n : Y_{n+x_n} = N | Y_n \neq N\}.$$ (3.29)

The distribution and expectation of the RUL is given by

$$\Pr(X_n = k) = \alpha_n \bar{P}^{k-1}(I - \bar{P})e, \quad \mathbb{E}(X_n) = \alpha_n(I - \bar{P})^{-1}e$$

(3.30)

where $\alpha_n = (\alpha_n(0), \alpha_n(1), ..., \alpha_n(N-1))$ is the distribution of the current degradation state. An example is $\alpha_n(i) = \Pr(Y_n = i), i \in \Phi \setminus \{N\}$ [23].

Learning for the Markov process consists of determining the number/type of states and estimating the probabilities in the transition matrix. Determining the division into states for the process is commonly done using expert knowledge about the process. The transition probabilities can be estimated using collected data. A simple way to do this is to approximate the probability as $P_{ij} \approx P_{ij} = \frac{\Pi_{ij}}{\Pi_i}$ where $\Pi_{ij}$ is the total number of transitions from state $i$ to state $j$ in the data set and $\Pi_i$ is the total number of data points in state $i$ [62].

The Markov process is normally not used to model degradation processes for RUL estimation on its own. Instead it forms the basis for other methods, such as the Hidden Markov Model (see Section 3.2.8 for details). It is also used in combination with other methods, an example of this can be found in [62]. There the RUL of a periodically monitored degradation process is estimated where the operation conditions are expected to be dynamic and evolve as a Markov process according to (3.27). Data from a turbofan engine used in the 2008 PHM Conference Challenge was used for validation of the method. The RUL was estimated by simulation using a path averaging method. Given the state $X_{n_i}$ at the current point in time $n_i$ a sequence of future states $\{X_n : n_i < n \leq N\}$ are simulated according to the evolution of the operational conditions. This is done $M$ times, where $M$ is a large number. The RUL is then estimated for each of the $M$ generated paths as the time until the first entry into the failure zone, details on this estimation can be found in [62]. The total estimated RUL of the process is then obtained by averaging over all paths according to the "law of large numbers".

An advantage of using the Markov process to model a system is the division of the system degradation into meaningful states such as {"Good", "OK", "Needs maintenance"} which make the model intuitive and easy to interpret. Another advantage is the strong mathematical basis of the Markov theory [23].

Disadvantages includes the assumption of a memoryless process since this is a quite strong requirement which is not fulfilled by all processes. Additionally there is currently no way to test whether the process fulfils this assumption or not. Also the Markov process assumes that the time in each state is exponentially distributed which is not always true [23].

### 3.2.8 The Hidden Markov Model

The Hidden Markov Model (HMM) was introduced in the early 1970s and it has been used a lot since then. This is due to its solid theoretical base and many applications in very diverse areas. The HMM consists of two different processes, one hidden Markov chain, which represents the true state sequence and can not be observed, and one observable process which is observed via condition monitoring data [3, 19, 23, 63]. The hidden process can be estimated from the observed process. The trained HMM can then be used to determine if an observation represents a normal or a faulty operational state [19, 64].
The degrading, hidden process $Z_n$ is a Markov chain which evolves in a finite state space $\Phi = \{1, 2, \ldots, N\}$, where 1 is the "normal operation" state and $N$ is the "failure" state. The relation between the hidden process and the observable process $Y_n$ is expressed using a probability measure $P(Y_n | Z_n = i), i \in \Phi \setminus N$. Figure 12 illustrates the HMM, the grey circles represent the observed process $Y$ and the white circles represent the hidden process $Z$. The HMM is fully specified by three parameters, provided that the number of states $N$ and the number of observations $V$ are known. The first parameter is the initial state distribution $\pi(i) = P(Z_1 = i)$, the second parameter is the state transition probabilities $A = (a_{ij}) = P(Z_n = j | Z_{n-1} = i)$ and the third parameter is the observation probabilities for each state $B = b_j(Y_n) = P(Y_n = y_n | Z_n = j)$. In the most basic HMM the parameters are assumed to be time invariant and stationary, and the model can then be written $HMM(\pi, A, B)$. The observations can be both continuous and discrete. If the observations are discrete they are represented as mass functions, and if they are continuous they are given by a parametric model family [63, 64].

It is possible to have HMMs of different orders. The simplest case is an HMM of first order which assumes that only the current state influences the system behaviour, Figure 12 is an example of a first order HMM. An HMM of second order assumes that the current state and the state before that can influence the behaviour and so on. Figure 13 is an example of a second order HMM [63, 64].

There are two important assumptions for the HMM: 1) Given the $n-1$:th hidden state the $n$:th hidden state is independent of all previous states and observations, that is $P(Z_n | Z_{n-1}, Y_{n-1}, \ldots, Z_1, Y_1) = P(Z_n | Z_{n-1})$. 2) Given the $n$:th hidden state the $n$:th observation is independent of all previous states and observations, that is $P(Y_n | Z_n, Y_{n-1}, \ldots, Y_1, Z_1) = P(Y_n | Z_n)$ [63].

The RUL for the HMM process at time $n$ is defined as

$$X_n = \inf\{x_n : Z_{n+x_n} = N \mid Z_n \neq N, \ Y_j, \ 0 \leq j \leq n\}. \quad (3.31)$$

Learning an HMM implies estimating the model parameters $\pi, A$ and $B$. This is normally done using the expectation-maximization (EM) algorithm, which finds the maximum likelihood or the maximum a-posteriori estimates of the parameters. One HMM is created for each state, so to determine which state the system is in all new observations are run through all trained HMMs. The state assigned to the new observation is the state which corresponds to the HMM which gives the highest log-likelihood for the observation [63].

In [21] a Mixture of Gaussians HMM (MoG-HMM) is utilised to estimate the RUL for bearings using data from the NASA collection of datasets [36]. Features are extracted from the data using wavelets and...
the MoG-HMM parameters can then be learnt offline from these features. The obtained model is then used online for estimation of the RUL from new input data. The MoG-HMM works like any HMM, the only difference is that it can handle continuous observations by modelling the observation probabilities as a mixture of Gaussians. This gives rise to two additional temporal parameters which have to be estimated, the mean and variance of the duration in each state. The parameters $\pi$, $A$ and $B$ are learnt using the Baum-Welch algorithm and the temporal parameters are learnt using the Viterbi algorithm (details on these algorithms can be found in [39]). The learnt model is used to first estimate the current state for the process. Then the path from the current state to the final (failure) state is found using the transition matrix $A$. The duration parameters for each state can then be used with the found path to estimate the mean and variance of the time until failure, that is the RUL. A more detailed description of how to estimate the RUL can be found in [21].

The HMM has some desirable properties. It is an intuitive model with a simple interpretation, since each HMM represents a state transition where each state corresponds to a certain level of degradation [63]. The HMM is also easy to implement and has a solid theoretical base. Additionally there exist efficient methods for determining the likelihoods for the parameters during the training process [64].

The HMM also has some limitations. Just like the Markov process it assumes the Markov property which is not fulfilled by all real systems. The successive observations of the system are assumed to be independent, which is also something that can not always be fulfilled in practise [3]. Also, the HMM requires a lot of memory for the learning phase, which makes it unsuitable for online prediction [23]. According to [63] there is not much research done on the appropriate choice of the type of HMM and the number of states for an HMM. It is also difficult to obtain a closed form expression for the RUL using the HMM since only the mean and the variance can currently be estimated [23].

3.2.9 Bayesian Networks

The Bayesian Network is a probabilistic model in the form of a directed acyclic graph (DAG). The DAG is a combination of a structure (network) and a set of parameters. The structure consists of nodes connected by edges (directed lines or arrows) [3]. Each node represents a random variable and each edge represents a dependence between the random variables associated with the connected nodes. Hence the Bayesian Network is very efficient for evaluating how different variables in a system are connected in terms of probabilities. An example of a Bayesian Network for analysing traffic flow can be seen in Figure 14. It can be seen that according to this graph traffic jams are dependent on rush hour, bad weather and accidents. For a specific edge in the network the node where the arrow starts is the parent node and the node where the arrow ends is the child. If an edge between two random variables is absent the two variables are independent. Looking at Figure 14 it can be seen that there is no connection between rush hour and siren, hence according to this graph these are independent. Each edge in the graph is associated with a set of parameters which contain the conditional dependencies between the associated nodes [65, 66].

![Diagram of a simple Bayesian Network for analysing traffic flow.](image)

Mathematically the Bayesian Network describes a joint probability distribution for a vector of random variables $X = (X_1,...X_N)$, and can be expressed as $B(G,\Theta)$ [3, 65, 66]. The structure $G$ is the DAG, which can also be expressed as $G(V,E)$ where $V$ is a vector of nodes representing the random variables $X$ and $E$ is the set of edges between the nodes. The edge $E_{ij} = V_j \rightarrow V_i$ symbolises the dependence between the random variables $X_j$ and $X_i$. Node $V_j$ is the parent node and node $V_i$ is the child node. The set of parent variables to a random variable $X_i$ is denoted $Z_i = \{X_1,...X_{L_i}\}$ and is represented by the nodes $D_i = \{V_1^i,...V_{L_i}^i\}$. The parameters $\Theta$ in $B(G,\Theta)$ contains the local conditional probabilities.
\( p(x_i | z_i) \) for \( X \). These are estimated from observations or expert knowledge and are normally presented in the form of a table \[65, 66\].

The Bayesian Network represents a joint probability distribution for the variables involved. A probability distribution can be factorized into a product of conditional distributions using the product rule according to \( p(x_1 \ldots x_N) = p(x_N | x_1 \ldots x_{N-1}) \ldots p(x_2 | x_1)p(x_1) \) \[39, 67\]. For a specific node in the network the distribution will be conditioned on the parent node. For the network in Figure 15 the probability distribution is given by \( p(a, b, c) = p(c|a, b)p(b|a)p(a) \). The node \( a \) is parent to both node \( b \) and node \( c \), and node \( b \) is also parent to node \( c \). The conditional probabilities \( p(c|a, b) \) and \( p(b|a) \) corresponds to the edges connecting the nodes \[39\].

![Diagram of a simple Bayesian Network with three nodes](image)

Figure 15: An example of a simple Bayesian Network with only three nodes.

Learning the Bayesian Network can be fairly complex and the learning is therefore often divided into two parts: learning the structure (edges and directions) and learning the parameters. The parameter learning can only be performed if the structure is known \[66\]. In some cases, when graphical limitations such as independence between variables exist, the complexity can be reduced \[65\]. The Maximum Likelihood algorithm can be used both to learn the structure and the parameters, but it requires a lot of data to give accurate results \[39\]. There exist several other methods for learning a Bayesian Network, some of these are K2, PC and Adding Arrows. Details on these methods can be found in \[65\]. The majority of the Bayesian Network methods learn the network generally, which means that they aim to accurately approximate the joint probability \( p(x_1, \ldots, x_N) \). There are also target-oriented methods which learn the structure for a specific purpose, such as classification. They aim to be more effective for their designated purpose than the general methods by not approximating the whole domain \[66\].

The naive Bayes model is one of the most simple and well-known target-based Bayesian classifiers. The structure is pre-set, so it does not require structure learning. The node representing the class variable is predetermined as the parent of all nodes representing attribute variables. The simplicity of the method is desirable, but it uses crude assumptions about the independence between involved variables. A more advanced and accurate method is the Targeted Bayesian Network (TBN). Instead of learning the whole joint probability for the system this method learns only the targeted variables to try to reduce the complexity of the computations. It aims to extract the most influential set of variables with respect to the targeted variable and then tries to find the relation between these variables \[66\].

Another type of Bayesian Network is the Dynamic Bayesian Network (DBN). This network extends the Bayesian Network by adding a temporal dimension. This is done by allowing updates of the network as time progresses or addition of states for which to perform the prediction. This can be thought of as modelling several separate Bayesian Networks which are used for different points in time with connections in terms of probabilities between them. The DBN is similar to the HMM but being a Bayesian Network it models each state as a set of random variables using conditioned probabilities instead of just representing each state using one random variable like the HMM does. Three different types of distributions are needed to describe the DBN; the prior distribution \( P(X_0) \) which includes the probabilities for all states \( X \) at the initial time slice \( t = 0 \) (each state is now given by the Bayesian Network at that time point), the transition distribution \( P(X_{t+1} | X_t) \) which gives the probability of each state at times \( t = 1 \ldots T \), and the observation distribution \( P(Y_t | X_t) \) which gives the probability of an observation \( Y_t \) with respect to all states (this is assuming that it is not possible to observe the states \( X \)). In Figure 16 a DBN with two states \( X_t \) and \( X_{t+1} \), each given by a Bayesian Network, is visualised. Note that it is possible to add states to the network both before and after these two states \[68\].

In \[69\] a DBN is used together with a Particle Filter to predict the RUL of a drill. The features extracted from observations (for example the thrust and torque) are used for the observation nodes and the RUL, which is here considered to be the number of additional drills that can be performed before failure, are here the hidden states. The RUL estimation is performed in three steps. First data is collected and classified into four health states. Then the DBN model is constructed based on these and is trained using the data to determine characteristics such as the prior, transition and observation
distributions. Prediction of the future health state is then performed and the RUL is estimated based on $P(X_t | Y_{1:T})$. The particle filter is used to find the point of failure by computing $P(X_{t+k} | Y_{1:T})$ where $k$ is some future point in time when the drill reaches the failure health state. The prediction with the Particle Filter is similar to the prediction that can be done using the DBN but the Particle Filter is more efficient when the number of points in time and features is big [69]. Details on the Particle Filter can be found in Section 3.2.10.2.

There are several advantages of using the Bayesian Network. It gives a compact and simple visual display of the dependencies of the system which makes it an intuitive method. While being intuitive, it still has a thorough mathematical foundation. It is a flexible method since the structure of the network can be manually constructed if the dependencies between variables are known, but it is also possible to automatically construct it from observed data [65, 66]. A common argument against the use of probability based methods is that they require an absurd amount of numbers to fully specify the probability distribution. The Bayesian Network is then very useful since it needs only a fraction of these to specify the distribution since it has built-in independence assumptions. It should be noted however that the training of the Bayesian Network is usually a NP-hard problem which means that many computations will be necessary for a big network [67].

### 3.2.10 Stochastic Filter-based methods

Filters commonly aim to solve the problem of recursively computing the distribution of some desired state (variable) $x$ given a series of observations $y$ related to that state (variable). To do this a model of the system is required. This model is normally given as a state space model consisting of two equations; one which describes the evolution of the state and one which describes the observations. Expressed in probabilities the first equation relates to the prior distribution of the state $p(x)$ whereas the second equation relates to the data likelihood $p(y|x)$. By using Bayes theorem given by

$$
p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(y|x)p(x)}{p(y)}
$$

the state distribution can be updated to the posterior belief $p(x|y)$ based on the measurement $y$ and the prior distribution of $x$. By doing this repeatedly the belief of the state distribution is recursively updated based on the measurements [70]. These filters, including for example the Kalman Filter, the Benes Filter and the Particle Filter, are normally used for trending some desired variable (state). For the specific application of estimating the RUL of a system this means that some feature of the system is tracked using the filter and the RUL is then estimated as the time until the tracked process is predicted to cross some threshold value. Finding this threshold is one of the main difficulties associated with these filters [23]. In the following subsections the Kalman Filter and the Particle Filter will be examined in more detail.
3.2.10.1 The Kalman Filter

The Kalman Filter is a linear, recursive filter which can be used to estimate the current or future state of a non-stationary process given observations and the previous states. For a linear system with Gaussian noise it gives the optimal estimate of the state and for a nonlinear system it gives the optimal linear estimate. Generally the Kalman Filter is mostly used for tracking since it provides an easy way to update the state estimate when a new observation of the system becomes available. For RUL estimation the Kalman filter is used for two things; tracking of condition monitoring variables and combining different estimates of the RUL when several models are used together [71].

Before implementing the Kalman Filter a linear state space model which describes the behaviour of the system must be constructed. The state space model consists of two equations, one describing the unobserved non-stationary state process $x_t$ and one describing the observations $y_t$. These processes are given by

$$
\begin{align*}
    x_t &= Ax_{t-1} + v_t \\
    y_t &= Bx_t + w_t
\end{align*}
$$

(3.33)

where $v_t$ is Gaussian process noise with variance $Q$, $w_t$ is Gaussian observation noise with variance $R$, $A$ is the state transition matrix and $B$ is the observation matrix. Note that both $A$ and $B$ can vary with time. The recursive Kalman Filtering can then be done in two steps (prediction and update) to find the next state estimate. The prediction step is based on knowledge from points in time prior to the current time. It consists of the equations

$$
\begin{align*}
    x_t^- &= Ax_{t-1} \\
    P_t^- &= AP_{t-1}A^T + Q
\end{align*}
$$

(3.34)

where $x_t^-$ is the prior estimated state vector at time $t$ given observations up to time $t - 1$, $P_{t-1}$ is the error covariance at time $t - 1$ and $P_t^-$ is the prior error covariance at time $t$ given measurements up to time $t - 1$ [23, 72, 73, 74, 75]. The second step updates the estimated state and error covariance using the new observation of the system $y_t$. The equations describing the update are given by

$$
\begin{align*}
    K_t &= P_t^- B^T (BP_t^- B^T + R)^{-1} \\
    x_t &= x_t^- + K_t (y_t - Bx_t^-) \\
    P_t &= (I - K_t B) P_t^-
\end{align*}
$$

(3.35)

where $K_t$ is the Kalman gain, $x_t$ and $P_t$ are the a posteriori estimates (using the observation at time $t$) of the state and error covariance respectively and $I$ is the identity matrix. The Kalman gain $K$ can be interpreted as a relative weighting between the earlier state estimates and the observations [23, 72, 73, 75].

There are two different ways of using the Kalman Filter for estimating the RUL. The first approach is to use the Kalman Filter to track the behaviour of one or more of the condition monitoring variables. The RUL in each time step can be computed by interpolating the trend of the behaviour of the condition monitoring variable forward in time until the value equals some failure threshold $\theta$. The RUL is then estimated as the time when the condition monitoring variable crosses the failure threshold minus the current time. In Figure 8 in Section 3.1 this way of estimating the RUL is illustrated (trending can be used both for physical and learning based methods). Two examples of this approach can be found in [71] and [74]. In [71] the Kalman Filter is used to track modal frequencies in a vibrating steel band with a notch. The states are the modal frequency, its velocity and its acceleration, and the state transition matrix is assumed to be Newton’s equations of motion for the modal frequencies. In [74] the Kalman Filter is used to estimate the RUL for an aircraft power generator by tracking the behaviour of the generator transfer characteristic dynamics $k_{hr}$. Just like in [71] the states are chosen to be position, rate of change (velocity), and acceleration of $k_{hr}$, and the state transition matrix encodes Newton’s equations of motion. The RUL for each state estimate can then be computed as the time $n$ steps ahead, $t_n$, from the current time $t_0$ when $|f(t_n)| < \epsilon$ where $f(t_n) = x_{t_0} + \dot{x}_{t_0} t_n + \frac{\ddot{x}_{t_0}}{2} t_n^2 - \theta$, $\theta$ is the failure threshold and $\epsilon$ is some small number.

The Kalman Filter can also be used to weigh together RUL estimates from several different models to produce a combined RUL estimate. This method is called ”Ensemble Kalman Filter” and works well when there are a lot of computational power available. An advantage is that the Kalman Filter is applied to the estimated RUL which is linear instead of the degradation variables which are nonlinear [72]. In [75] several Neural Networks are used as models for producing a set of RUL estimates for data used in
the 2008 PHM Challenge. There are two main shortcomings of these Neural Networks, the first is that
the estimates produced contain random errors and the second is that they do not take into account the
predictions from earlier points in time. A Kalman Filter was therefore used to both merge the different
estimates and to make sure that the estimated RUL was based on previous estimates as well as the
current observation of the RUL. This approach will also make the estimate more robust. In [73] an
extension to the ensemble Kalman Filter in [75] is presented which can be used on nonlinear degradation
systems by approximating the nonlinear system as a piecewise linear system. In [72] the same method is
used for RUL prediction for a turbine blade in a power plant. Ensemble models increase the robustness
of the RUL estimation provided that the model ensemble is diverse enough. This is due to the fact that
a single model will always be limited in reality, hence the idea is that if an ensemble of models are used
at least one of the models should provide a good estimate.

Limitations of the Kalman Filter includes that, like for all filter based methods, a state space model of
the system is required and if such a model is difficult to obtain the Kalman Filter can not be used. Another
problem is that it might be difficult to relate the condition monitoring variables to the unobserved
condition $x$. Also, it can be difficult to choose the threshold that the unobserved condition must stay
below when it is not possible to observe the condition. The perhaps most severe limitation however is
that the Kalman Filter assumes that the process is linear with Gaussian noise. When this is indeed the
case the Kalman Filter is optimal, but if the process is actually nonlinear with non-Gaussian noise it
will perform poorly [23]. Other methods must then be used such as the extended Kalman Filter which
linearises the nonlinear model, or the Particle Filter which is introduced in Section 3.2.10.2.

3.2.10.2 The Particle Filter The particle filter can handle nonlinear processes with non-Gaussian
noise and is hence more general than the Kalman filter [76]. Just like the Kalman Filter it assumes that
the system can be described using a state space model with one equation for the state evolution and
one equation for the observations. The particle filter approximates the pdf of the state using a set of
particles, hence the name. For each particle there is a corresponding weight equalling the probability
density of that particular particle. The particles are generated using Monte Carlo sampling and the filter
then uses Bayes theorem 3.32 and the state space description to propagate the particles forward in time
or to predict the RUL.

When implementing the particle filter the state of the system $x_t$ is normally assumed to follow a first
order Markov process given by

$$ x_t = f(x_{t-1}) + v_t $$

(3.36)

where $f$ is a nonlinear function of the previous state $x_{t-1}$ and $v_t$ is noise, not necessarily Gaussian. The
measurement process is assumed to follow

$$ y_t = h(x_t) + w_t $$

(3.37)

where $h$ is a nonlinear function and $w_t$ is noise, not necessarily Gaussian. The equations for the state
evolution and the measurement can also be written in a probabilistic form and are then given by $x_t \sim
p(x_t|x_{t-1})$ and $y_t \sim p(y_t|x_t)$ respectively [76, 77].

The particle filter aims to approximate the state pdf $p(x_t|y_{1:t})$ at time $t$ using a set of $N$ particles
$\{x_i\}_{i=1}^N$ and their corresponding weights $\{w_i\}_{i=1}^N$. The initial particles are generated from an initial
distribution $p(x_0)$ and are then propagated forward in time in two steps; prediction and update. In the
prediction step a priori state estimate $p(x_t|y_{t-1})$ is generated from the previous estimate $p(x_{t-1}|y_{t-1})$
using the state evolution process in 3.36. In the update step the weight of each particle is updated based
on how well the prior estimate $p(x_t|y_{t-1})$ corresponds to the new measurement $p(y_t|x_t)$. Recursively the
unnormalised weights are given by

$$ \tilde{w}_i^t = w_{i-1}^t p(y_t|x_t^i) $$

(3.38)

where $w_{i-1}^t$ is the normalised weight from the previous point in time and $p(y_t|x_t^i)$ is the measurement
likelihood for the prior particle $x_t^i$. The weights are then normalised according to $w_i^t = \frac{\tilde{w}_i^t}{\sum_{i=1}^{N} \tilde{w}_i^t}$ [76, 77].

The above algorithm is the most basic implementation of the Particle Filter and it is not recommended
to use in practise since it has the severe shortcoming that within a few iterations only one or a few
particles will have significant weights which results in estimates with a very high variance. To remedy this
shortcoming resampling with replacement can be used. New samples are then drawn in every iteration
according to the prior distribution of the state in the same iteration [78, 79]. This type of Particle Filter
is called the Bootstrap Particle Filter and is illustrated in Figure 17. When using resampling the weights are updated according to the measurement likelihood only, hence the new unnormalised weights are given by
\[
\tilde{w}_i^t = p(y_t|x_i^t).
\]
These weights are then normalised as for the basic Particle Filter. New particles, which will form the posterior distribution, are then drawn according to the cumulative distribution of the normalised weights. This means that the probability of a particle being resampled equals its current weight. Hence particles which agree with the measurement will have a higher chance of being resampled. This can be seen in Figure 17 where the particles (blue circles) which correspond well with the data likelihood (red curve) clearly get a higher weight (symbolised by the size of the particle). This is the simplest form of resampling and if necessary more advanced methods for resampling, such as systematic or residual resampling, can be used. For details on these see [79].

\[
\tilde{w}_i^t = p(y_t|x_i^t).
\]

(3.39)

When the state distribution is unknown it is common to use a concept called importance sampling. Samples are then drawn from an importance distribution \(q(x_t|x_{t-1}, y_t)\) and the weight update (unnormalised) is modified to
\[
w_i^t = w_{i-1}^t \frac{p(y_t|x_i^t)p(x_i^t|x_{i-1}^t)}{q(x_t|x_{t-1}^t, y_t)}.
\]

(3.40)

where \(p(y_t|x_i^t)\) and \(p(x_i^t|x_{i-1}^t)\) are the state and measurement probabilities [76, 77].

It is also possible to use adaptive resampling meaning that resampling is only done when some criteria on the weights falls below some threshold set by the user. A common measure for when to perform resampling is given by the number of effective samples
\[
N_{\text{eff}} = \frac{1}{\sum_{i=1}^N (w_i)^2},
\]

hence if \(N_{\text{eff}}\) is too low resampling is performed at time \(t\) using 3.39 for updating the weights, otherwise 3.38 is used to update the weights [81].

The Particle Filter described above can be used to for example track an important feature, but to get estimates of the RUL of a system the state distribution \(p(x_{t+p}|y_1:t)\) at the future point in time \(t+p\) is required. The estimation of this distribution is more involved and details on a few different ways to do this can be found in [81].

When performing CBM the state variable \(x_t\) in 3.36 corresponds to some feature describing the degradation of the system. For an accurate prediction of the RUL both a good estimate of the current state of the system and a good model of the degradation process are required [76]. Once a model of the system is known the Particle Filter can be implemented according to the basic or bootstrap algorithm, or the more involved version described in [81].

In [76] the Particle Filter was used to estimate the RUL of turbine blades using the methods described in [81]. The degradation consists of cracks in the blade which are assumed to grow according to the state evolution model \(N = P + f(L^N) + w(N)\) where \(N\) is the number of stress cycles, \(L\) is the length of the crack, \(P\) is a random variable with known mean, \(f\) is a polynomial function and \(w(N)\) is i.i.d white noise. The measurement model for the process is given by \(y(N) = L(N) + v(N)\) where \(v(N)\) is assumed to be Gaussian. Using the involved method to compute a predicted value at a future point
in time the probability of failure can be estimated based on when the predicted value crosses a failure threshold. From this probability a confidence interval of the estimated time to failure (RUL) is then computed. Another example where the Particle Filter is used for estimation of the RUL can be found in [77] where the RUL of Lithium-ion batteries is estimated. Here the state evolution model is estimated from a set of parameters using relevance vector machines (RVM) assuming an exponential growth. The model obtained from the RVM is then used to propagate the particles forward in the Particle Filter used for estimation of the RUL. Resampling is used in each iteration to prevent degeneracy of the weights. The Particle Filter was used for tracking and the obtained estimates were then projected forward in time to get an estimate of the RUL.

The main advantage of the Particle Filter is that it can handle nonlinear models and non-Gaussian noise. A problem with using the Particle Filter however is that it, just like the Kalman Filter, requires a model of the system. This is a restriction since assumptions and approximations must be made. Depending on the amount of particles required to approximate the distributions accurately the Particle Filter may also be computationally expensive which is a problem if it is used for online prediction [29].

### 3.2.11 Fuzzy Logic systems

Fuzzy logic is a type of many-valued logic where the variables can take all values between 0 (definitely false) and 1 (definitely true) [82, 83]. The Fuzzy Logic system aims to sort the input data into different "fuzzy sets" which are then combined with rules in the form of "IF...THEN..." statements to produce an output [84]. It is built on the assumption that the input data can be classified as belonging to all fuzzy sets to different degrees, on a scale from 0 (not at all in the set) to 1 (definitely in the set). The graded scale makes the boundary between the sets smooth and overlapping, also called "fuzzy" [3, 83, 84]. An example of fuzzy sets are the states hot, warm, cool and freezing. It is not entirely clear what temperatures that would belong to these vaguely defined sets since a certain temperature may for example be considered hot by one person but only warm by another person. Hence that particular temperature would belong to both hot and warm to some degree and the resulting sets will be fuzzy [82]. The idea of fuzzy logic came from observations of the reasoning of the human mind. Humans almost always make decisions based on vague, uncertain or incomplete information and yet they succeed in creating acceptable and definite results. This is also the case for the Fuzzy Logic system [82, 85].

The fuzzy logic algorithm consists of three different steps. In the first step the exact input \( x \in X \), where \( X \) is the input domain, is mapped to one or more of the fuzzy sets \( A_i \) to some degree using a membership function \( \mu_i(x) : X \rightarrow [0, 1] \). The membership functions can be any function which gives an output in the range \([0, 1]\), but they are normally simple functions such as triangular, trapezoidal or Gaussian. Note that the total membership for input \( x \) should always sum to one [82, 83]. In Figure 18 an example of triangular membership functions are visualised for the temperature example presented above where the fuzzy sets are freezing, cool, warm and hot. In the second step the part of the input \( x \) in a fuzzy set \( A_i \) is combined with rules of the type "IF \( x \) is in \( A_i \) THEN \( y \) is in \( B_i \)" to determine the degree to which output \( y \) belongs in the fuzzy set \( B_i \). In the third and last step the inverse to the process in step one is performed to extract the exact output \( y \in Y \), where \( Y \) is the output range.

![Figure 18: An example of triangular membership functions for the temperature example where the fuzzy sets are freezing, cool, warm and hot.](image)

Fuzzy logic is often used as a helping tool for other algorithms such as Kalman filtering, Expert Systems and Neural Networks, but it can also be used as the main estimation algorithm [3]. In [86] the RUL for a nuclear system is estimated using "fuzzy similarity analysis" where they compare the current trajectory of the system to different reference trajectory patterns for which the RUL is known. For the reference trajectories the whole lifespan of the system is assumed to be recorded, from startup to failure. All reference trajectories are saved in a matrix \( R_{N \times k} \) where \( N \) is the number of trajectories and \( k \) is...
the maximum number of entries for each trajectory. Each element \( r_{ij} \) of \( R \) has length \( n \). The pointwise difference between the current measurement and all reference trajectories in \( R \) is saved in a matrix \( \delta_{N \times k} \) where each entry is given by \( \delta_{ij} = f(t) - r_{ij} \) where \( f(t) \) contains the last \( n \) measurements from the currently observed system. The fuzzy logic algorithm is then used to determine whether a reference trajectory is similar to the current trajectory using the condition that the difference \( \delta \) is “approximately zero”. The pointwise difference \( \delta_{ij} \) is mapped using a bell-shaped membership function, \( \mu_{ij} = e^{-\frac{\alpha^2}{\beta^2}} \) where \( \alpha \) and \( \beta \) are design parameters, to determine how much the entry belongs to “approximately zero”. A high value on \( \mu \) corresponds to a good match to “approximately zero”. After the mapping a distance score \( d_{ij} = 1 - \mu_{ij} \) is calculated. The minimum distance score for each trajectory is then used to compute a weight \( w_i \) for that trajectory which will then be used in the calculation of the RUL which is given by 
\[
RUL = \sum_{i=1}^{N} w_i RUL_i.
\]
The weights are given by given by \( w_i = (1 - d^*_i) e^{-\frac{1}{2}d^*_i} \) where \( d^*_i \) is the minimum distance score for trajectory \( i \) and \( \beta \) is the same as in the membership function. The RUL for one of the reference trajectories, \( RUL_i \), is simply determined to be the failure time of that trajectory minus the last time point for the part of the trajectory that was most similar to the currently measured trajectory.

A big advantage of the Fuzzy Logic system is that it is inherently robust, meaning that it can handle noisy, ambiguous, distorted and incomplete input data. After the system has been defined it is easy to modify it by just adding/changing the rules for the decision making. Also, the mathematics involved are very straightforward which makes it easy to construct the system. Another advantage is that despite being very simple the method can be used to model nonlinear systems [82, 84].

There are also some disadvantages of the Fuzzy Logic system. Firstly there is no systematic approach for how to choose the fuzzy sets and how to define the rules. Secondly, the fuzzy logic is based on making decisions from vague data which makes it unsuitable for applications where a very high accuracy is requested. Thirdly the model can become very complex when the number of rules and/or fuzzy sets increase or if the number of inputs is large [82].

### 3.2.12 Principal Component Analysis

The Principal Component Analysis (PCA) is a multivariate, unsupervised analysis technique [14, 19]. It extracts the most significant features from the data by decorrelating it through an orthogonal transformation into principal components. Only the most significant (highest variance) components are used to describe the data, which can significantly reduce the dimension of the data. The ability to reduce the formation into principal components. Only the most significant features from the data by decorrelating it through an orthogonal transformation into principal components. Only the most significant (highest variance) components are used to describe the data, which can significantly reduce the dimension of the data. The ability to reduce the dimension of the data makes PCA suitable to use with other methods to improve the computation speed. PCA can also be used to remove the noise in a signal since the most significant principal components normally corresponds to the interesting information whereas the noise is normally related to less significant components [3, 14, 87].

Mathematically the input data \( X \) is an \( n \times m \) matrix with \( n \) representing the number of measurements and \( m \) representing the number of features. For PCA to work properly the input data must be normalised to mean zero. This can either be done before computing the correlation matrix \( R \) by subtracting the mean of each feature \( \bar{x}_j \) from all the measurements \( x_{ij} \) for that feature, or when computing the correlation matrix. Each element of the correlation matrix \( R \) is given by
\[
R_{ij} = \text{Cor}(i,j) = \frac{n - 1}{\sqrt{\sum_{k=1}^{n}(x_{i(k)} - \mu_i)^2 \sum_{k=1}^{m}(x_{j(k)} - \mu_j)^2}} \text{Cov}(i,j)
\]
where
\[
\text{Cov}(i,j) = \frac{1}{n - 1} (x_{i} - \mu_i)(x_{j} - \mu_j) \quad i, j = 1, 2...m
\]
and \( \mu_i \) and \( \mu_j \) are the mean values of the \( i \):th and \( j \):th rows of the data matrix \( X \). Note that it is sometimes also necessary to scale the normalised inputs to have standard deviation one in order for PCA to give a good result. Next, the \( m \) eigenvalues \( \lambda_i \) and eigenvectors \( v_i \) of \( R \) are calculated. The relation between the eigenvalues and the eigenvectors is given by \( Rv_i = \lambda_i v_i \) where \( R \) is the correlation matrix. The eigenvalues can either be computed directly from this relation, or by using the singular value decomposition. The eigenvalues are then sorted so that the first eigenvalue in the resulting sequence is the largest and the last one is the smallest. This is because the biggest eigenvalue corresponds to the biggest variance in the data and hence it contains most of the information. The smallest eigenvalues are usually ignored. A matrix \( V \) of eigenvectors is created, which contains the chosen principal components as its columns. New, uncorrelated samples of the input \( X \) are generated using
\[
X_{new} = v^T X.
\]
This gives the original input data in terms of the most significant principal components [14, 88, 89]. There are several different versions of PCA such as dynamic PCA which can capture dynamic relations between the variables, and kernel PCA which can handle nonlinear relations [3].

For RUL estimation PCA is mostly used in combination with other methods, such as Neural Networks or support vector machines. PCA performs the feature extraction from noisy data or reduces the number of features used for RUL estimation. In [87] PCA is used together with Support Vector Data Description (SSVD) to estimate the remaining useful life of bearings. The SSVD estimates the RUL by fitting the data to a hypersphere whose radius is used as a health indicator for the RUL estimation. The PCA is here used to reduce the dimension of the input from seventeen features to three while still keeping 97 percent of the information from the original input. Another example of an application of PCA for RUL estimation can be found in [89]. They use PCA together with a Neural Network to estimate the RUL of bearings. The focus there is on finding the optimal number of features to use for the estimation of the RUL.

The biggest advantage of PCA is the ability to reduce the dimension of the data without loss of relevant information. Another advantage is that it facilitates the interpretation of the data by accentuating similarities and differences between features in the data [88]. It can also be used to find the optimal number of features to use for the estimation to avoid effects of overfitting [89].

### 4 Case study: Bromma Conquip

To learn more about the practical aspects of implementing CBM a case study was conducted where data from one of CrossControls customers, Bromma Conquip, was examined and processed. The case study also aimed to investigate which of all methods for estimation of the RUL in Section 3 that might be of relevance to CrossControl. Details on Bromma Conquip, the analysis of the obtained data and possibilities for estimation of the RUL can be found in the following sections.

#### 4.1 Bromma Conquip

Bromma Conquip is the largest manufacturer of crane spreaders, see Figure 19, in the world with around 50 percent of the global market. The company was founded in 1960 and has since then produced more than 14000 spreaders out of which 9000 are currently in operation. Apart from being the largest manufacturer they are also the technical leaders in the spreader industry having developed several new types of spreaders in the past. In 2007 they released their new control system Bromma SCS\(^3\) which can perform diagnostics and some basic prognostics. In the future Bromma Conquip intend to develop and implement more advanced prognostics. Bromma Conquip aims to deliver high performance spreaders to their customers and therefore the reliability and quality of the spreaders are very important. The current diagnostic system enables detection and classification of faults for better and faster maintenance. This is an important aspect in making Bromma Conquip’s spreaders superior to those of the rivalling spreader manufacturers [90].

![Figure 19](image.png)

Figure 19: One of Bromma Conquip’s spreaders (orange) [91].

The diagnostic system used today in Bromma Conquip’s spreaders consists of diagnostic blocks programmed by Bromma Conquip on the platform provided by CrossControl (DABE/DRE). The diagnostic system sends error messages when it detects an error and can also collect data from the spreader system. The computations are performed in the spreader and the data can either be sent from the spreader
using a modem (expensive) or be saved in a database for later collection. Presently there are no formal definitions of errors or failures. These are instead implemented in the system in the form of limit values based on experience (expert knowledge). Bromma Conquip wants to further develop their diagnostic system in two ways; they want to expand their analysis of statistics to learn more about the system, and they want to implement more advanced prognostics to be able to foresee errors and failures and hence give their customers a product with even higher reliability [92].

Currently there is no analytic model of the system but according to [92] it is a fairly simple system which can easily be separated into different parts. The most important subsystem is the twistlock which is the mechanism that locks the container to the spreader during lifts. It is crucial that this part works efficiently and reliably for several reasons. If a container is dropped during a lift it is a safety hazard to people working and it can also damage the load, machines on the ground or other containers. Having a ship in a port costs very much and therefore it is important to unload as quickly and smoothly as possible which requires that the twistlocks can lock and unlock easily and reliably. Finding out early that a twistlock is not working properly would enable better planning of maintenance and more efficient loading/unloading of ships. In this case study the possibility of implementing prognostics for the twistlock on Bromma Conquip’s spreaders will be examined using data from test runs with one of Bromma Conquip’s spreaders in Oslo.

4.2 The twistlock

The twistlock was invented in the 1950s in Spokane by the transport engineer Keith Tantlinger. It revolutionised the handling and stacking of containers and made the world trade more efficient, both by making it possible to stack containers on top of each other and by being a very simple construction which did not need much maintenance. They are used on cranes, on container ships, on trucks and on railway container trains [93].

One twistlock is placed in each corner of the container and consists of two parts; the actual lock and the corner casting. The corner casting is also called the “female” and this is the part fastened on the container. It consists of an oval hole with no moving parts and can be seen to the right in Figure 20. The lock is also called the “male” and it is placed on cranes and transport bases or between stacked containers. An example of the “male” part of the twistlock can be seen to the left in Figure 20. The top part of the lock can be rotated 90 degrees after being inserted through the hole in the corner casting which locks the container into place on the crane or the transport basis. The difference between the locked and the unlocked position for the twistlock can be seen in Figure 21. Each twistlock can be in the locked position, the unlocked position or in between these two positions when the container is in the process of being locked or unlocked [92, 93].

![Figure 20: The male part of the twistlock (left) and the female part of the twistlock (right) [94].](image)

![Figure 21: The twistlock in the unlocked position (left) and the locked position (right) [93].](image)

It is possible to have twistlocks both temporarily installed, e.g. when containers are stored on land, or permanently installed, e.g. on ships or cranes. The twistlock requires very little maintenance and the
degradation of the twistlock is mainly due to corrosion. This affects the male part more than the female part which is convenient since the male part is the part more easily and frequently inspected since it is placed on vehicles and cranes[93]. It is possible to disassemble the twistlock to perform maintenance or to change parts. Possible maintenance actions include cleaning the twistlock, greasing it or replacing parts or the whole lock if it is worn out [95].

4.3 The data

Operational data from one of Bromma Conquip’s spreaders in Oslo will be processed and examined. The data is collected in a csv file which contains all signals sent between the spreader and the crane. Only a small subset of all signals contains information about the twistlocks and this information must somehow be extracted to enable analysis. To be able to extract the relevant information the structure of the file must first be understood, preferably by opening the file. The size of the data file (7.5 GB) prevents it from being opened in its original form using the built-in import functions in programs like Excel or MATLAB. Therefore the file was first opened using the program 010 Editor [96]. This is a program which displays the file without formatting and hence it can open quite large files. When the structure of the data file is known it can be loaded row by row into programs like MATLAB saving only the rows containing information about the twistlocks.

The first 15 lines of the data file are presented in Figure 22 as they are displayed in the 010 Editor. The first eight rows are the header lines and the data starts from row nine in the file. From the header it is clear that the unit for the time stamp is seconds and that the data is saved as hexadecimal numbers. The last header row, row eight, contains the names of all the data columns. This is the only header row necessary when importing the file into a program where the data can be analysed. Hence rows 1-7 are deleted and a new version of the data file is saved. In Figure 23 a small part of the new data file can be seen as it is displayed in Excel, formatted to facilitate the interpretation.

![Figure 22: The first 15 rows of the csv file as displayed in 010 Editor.](image)

The data columns that will be used for analysing the twistlocks are Time, which is the timestamp for each measurement, id which is a unique number identifying each signal, and one or more of the Data columns depending on how much information the examined signals include. As can be seen in Figure 23 the signal with id 614 uses all eight data columns to store information whereas the signal with id 713 only uses one.

The id of the signal is used to filter out the signals relating to the twistlocks (the datafile contains all signals for the whole spreader system), hence the id of signals passed to and from the twistlocks must be determined. The id of the desired signals can be found using the model of the whole system in CoDeSys. The different parts of the spreader-crane system communicate using CAN-buses where the crane is the “master” and each motor or sensor on the spreader is a “slave” (also called a “node”). In between the “master” and the “slave” is a manager (in this case Bromma Conquip’s diagnostic system). Figure 24 shows a sketch of the system.

Each node has a unique ID from which it can be identified. In CoDeSys the four twistlocks have node ID 4, 5, 6 and 7. Signals going to or from the node are labelled with an identifying number in CoDeSys which ends with the node ID. This number is the id of the signal in the data file. Signals from the node to the master are labelled 180+node ID, 280+node ID, 380+node ID and so on, whereas signals going from the master to the node are labelled 200+node ID, 300+node ID, 400+node ID and so on. From the CoDeSys model it can be determined that the signals labelled 184, 185, 186 and 187 contain the data
sent from the twistlock to the crane (master) and signals 202 and 203 contains the signals sent from the crane to the twistlock with locking/unlocking commands. When the id of the relevant signals are known a MATLAB script is implemented which loops through the data file and saves only the relevant signals. This reduces the size of the data file from 7.5 GB to 6.2 MB which can easily be loaded into MATLAB. The script for looping through the data file can be found in Appendix B.

From the data columns in the data file the useful information must be extracted. All information about each signal is stored as a CAN-message which consists of an ID for the signal and 64 bits of data, see Figure 25. The model of the system in CoDeSys includes an explanation of what information is contained in the CAN-message from each node or from the master. The signals with id 184-187 only have data stored in the Data0 column which corresponds to the first 8 bits of the data part of the CAN-message. These 8 bits are stored as a hexadecimal number in the data file. The number can be converted to a binary number which give a sequence of 8 numbers having value 1 or 0. What each of these 8 bits represents can again be found in the CoDeSys model for each twistlock. For signals 184-187 the first 4 bits are an ID, the 5th bit corresponds to the variable landed, the 6th bit corresponds to the variable unlocked, the 7th bit is not used and the 8th bit corresponds to the variable locked, see Figure 25.

It is the three variables landed, unlocked and locked together with the timestamp Time that will be used for analysis of the system and possibly also for prognostics. The extraction of the values of these three variables is implemented in the MATLAB script for reading the data file and saving the relevant data, see Appendix B.

The variables landed, unlocked and locked can take either value 1 or value 0. landed has value 1 when the spreader has landed on the container and the container is still on the ground. It has value 0 when
Figure 25: The structure of the CAN-message, which consists of an ID and a 64-bit data part.

the spreader is in the air (with or without the container). unlocked has value 1 when the twistlock is in the unlocked position and value 0 otherwise. locked has value 1 when the twistlock is in the locked position and value 0 otherwise. The combination of the values of these three variables determines where in its lift cycle the spreader is at every point in time, more details on this in Section 4.4.

The signals with id 202 and 203 contain commands from the crane for locking/unlocking the twistlocks. There are just two signals since each signal controls two of the twistlocks simultaneously. Signal 202 controls node 4 and 5 whereas signal 203 controls node 6 and 7. These signals use three of the data blocks in the CAN-message; Data0, Data1 and Data2. However, only the column Data0 contains useful information, Data1 and Data2 are only used for alarms. The locking/unlocking command from the crane is stored in Data0 (the first 8 bits of the CAN-message) as a hexadecimal number. For 202 and 203 there are only three possible values that Data0 can take (0, 1 or 2) and therefore it is not necessary to convert from hexadecimal to binary. Data0 = 0 means that the motor is not doing anything, both for signal 202 and for signal 203. For signal 202 Data0 = 1 means unlocking and Data0 = 2 means locking. For signal 203 the opposite is true, Data0 = 2 means locking and Data0 = 1 means unlocking.

Apart from the CAN-messages the nodes use to communicate with the crane (184-187) and the messages from the crane (202-203) there are also two types of special messages for each node which can be used to determine if something goes wrong. The first type of message is the heartbeat message which has id=700+node ID. Each node sends this message with a certain period to let the master know it is alive. It can contain four different messages; 00=initialising, 04=stopped, 05=operational and 7f=pre-operational [97]. The second type of message is the emergency message (EMCY) which has id=800+node ID. These messages are sent when something in the CAN-bus or the node fails and a list of possible EMCY messages can be found in [98]. The idea is that the heartbeat and EMCY messages, if present in the data, could be used to help identify possible failures. When the data was scanned for these signals all signals with id 704-707 where indicating a normal operation and no signals with id 804-807 where found so no emergency messages were sent. Therefore these two types of signals will not be considered further in the analysis presented here. Note however that they might contain useful information and should be considered if another dataset or system is examined.

4.4 Analysis of the data

To gain a better understanding of the process of lifting containers the entire dataset was plotted to observe the characteristic behaviour and to find possible failures. In Figure 26 a typical lift can be seen for the twistlock with node ID 4. Both signal 184 and 202 are plotted to visualise how the control signal from the crane affects the locking/unlocking process. Note that all three variables landed, locked and unlocked normally have value 0 or 1 but in the figures in this section landed have value 0.2 or 1.2 and unlocked have value 0.4 or 1.4. This is only for plotting purposes to give a cleaner and more intelligible figure where the variables do not cover each other. The discussion in the text will assume that all variables take value 0 or 1 when referring to figures, hence if unlocked has value 1.4 in a figure it should be thought of as having value 1.
Four states were chosen to characterize the lifting process for the spreader. These are: *Idle*, *Locking*, *Lifting* and *Unlocking* and they can all be observed in Figure 26. In the *Idle* state the spreader is in the air and unlocked, and the variables have values \( \text{landed} = 0 \), \( \text{locked} = 0 \) and \( \text{unlocked} = 1 \). During the lift in Figure 26 the spreader is *Idle* up until \( t \sim 338 \text{ s} \) and after \( t \sim 366 \text{ s} \). In the *Lifting* state the spreader is in the air and locked (lifting a container), and the three variables have values \( \text{landed} = 0 \), \( \text{locked} = 1 \) and \( \text{unlocked} = 0 \). In Figure 26 this occurs between \( t \sim 344 \text{ s} \) and \( t \sim 360 \text{ s} \).

The state *Locking* is the state when the spreader has landed on top of the container and the twistlocks are locking. This state is a bit more complicated than the previous two. It starts when \( \text{landed} \) goes from 0 to 1 and ends when \( \text{landed} \) goes from 1 to 0 provided that \( \text{unlocked} \) goes from 1 to 0 and that \( \text{locked} \) goes from 0 to 1 during the time when \( \text{landed} = 1 \). Signal 202 should have value 2 when the spreader is in state *Locking*. This occurs between \( t \sim 338 \text{ s} \) and \( t \sim 344 \text{ s} \) in Figure 26. The state *Unlocking* is the state when the spreader is locked on top of the container on the ground and the twistlocks are unlocking. It is defined in the same way as the *Locking* state with the difference that \( \text{unlocked} \) goes from 0 to 1 and that \( \text{locked} \) goes from 1 to 0 during the time when \( \text{landed} = 1 \). Signal 202 should have value 1 in this state. In Figure 26 this occurs between \( t \sim 360 \text{ s} \) and \( t \sim 366 \text{ s} \).

For normal lifts the spreader always go through the states in the order that can be observed in Figure 26: *Idle*→*Locking*→*Lifting*→*Unlocking*→*Idle*. To illustrate this signals 184 (in the form of extracted values for *Locking*, *Lifting* and *Unlocking*) and 202 are plotted in Figure 27 for a series of lifts. It is clear that all of these lifts follow this order of transitions with signal 202 having value 2 in state *Locking* and value 1 in state *Unlocking*. The main differences between the lifts is that the time spent in the *Idle* and *Lifting* states varies between the lifts. This is to be expected since the distance different containers are lifted may vary and hence also the time it takes to lift them from one place to the next. The time spent in states *Locking* and *Unlocking* on the other hand seems to be fairly constant in comparison.

Apart from the normal behaviour the data also contain sections with abnormal behaviour. The most abundant abnormal feature is oscillations between 1 and 0 in the \( \text{landed} \) variable. An example of this behaviour for a lift for signal 187 can be seen in Figure 28 where the value for \( \text{landed} \) jumps between 0 and 1 more than 50 times during a lift when the container is in the air and \( \text{landed} \) should have value 0 constantly. Most of these oscillations last only for a very short time but they can become dangerous if they last for a long enough time so that the system thinks the container has landed. Therefore this behaviour can be considered a fault, but not the most severe type of fault since the spreader can still operate normally. Note also that in Figure 28 the signal 203 is also plotted and that it has value 1 when in state *Locking* and value 2 when in state *Unlocking* in contrast to signal 202 in the two previous examples.

A few sections where the behaviour of the spreader differs from a normal lift in other ways than the oscillations can also be found in the data. An example of this for node 6 (signal 186 and 203) can be seen in Figure 29. The spreader starts from *Idle* and moves to *Locking* and then to *Lifting* as usual. When the spreader tries to unlock at \( t \sim 3520 \text{ s} \) \( \text{landed} \) changes value to 1 and \( \text{locked} \) changes value to 0 as
Figure 27: A sequence of typical lift cycles for node 4. In the upper figure locked, unlocked and landed extracted from signal 184 are plotted and in the lower figure signal 202 is plotted.

Figure 28: A lift for node 7 when oscillations in landed are present. In the upper figure locked, unlocked and landed extracted from signal 187 are plotted and in the lower figure signal 203 is plotted.

usual, but unlocked does not change its value to 1 as it should. This indicates that the twistlock leaves the locked position but never reaches the unlocked position. The system stays like this until $t \sim 3610s$ when locked goes to 1 for a short time and then returns to 0. Examining signal 203 at the same time it can be seen that it goes to 1 during the Locking state as it should and it takes value 2 shortly after landing at $t \sim 3520s$. Hence the crane tells the twistlock to unlock. Another command to unlock arrives after a short time but has no effect. The crane then sends commands both to lock (1) and unlock (2) repeatedly until $t \sim 3600s$ with no result. At $t \sim 3610s$ a locking command succeeds and the system can then unlock a short time after that and move to the Idle state as it should. This is an example where the twistlock for some reason does not work as it should. The system was able to handle this fault on its own, but it could be an indication of a future failure and a diagnostic system would in this situation notify the driver that something unexpected occurred. Two more examples of abnormal behaviour with an explanation of what happens can be found in Appendix C.

4.5 The classification

Many of the algorithms used for diagnostics and prognostics are supervised and hence need classified data. The dataset from Bromma Conquip is not classified, therefore a classification scheme will be suggested in this section. The classification can then be used to determine how long the spreader stays
Figure 29: A lift for node 6 when the lock can not reach the unlocked position. In the upper figure locked, unlocked and landed extracted from signal 186 are plotted and in the lower figure signal 203 is plotted.

in each state which could enable offline diagnostics and prognostics for the system.

The idea is to classify the data using the four previously mentioned states for the spreader (Idle, Locking, Lifting and Unlocking) and two additional fault states. One of the fault states will handle the oscillations, since this is the most frequently occurring fault in the dataset. This state will be called Oscillation and can occur during any of the four states indicating normal behaviour. The second fault state will be called Other fault and will handle all unknown behaviours such as the problem with not being able to unlock the spreader that could be observed in Figure 29.

The classification algorithm was implemented in MATLAB using the knowledge gathered from analysing the data. The code for the implementation can be found in Appendix B. For the classification each state is given an identifying number, see table 1.

<table>
<thead>
<tr>
<th>State</th>
<th>Identifying number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
</tr>
<tr>
<td>Locking</td>
<td>1</td>
</tr>
<tr>
<td>Unlocking</td>
<td>2</td>
</tr>
<tr>
<td>Lifting</td>
<td>3</td>
</tr>
<tr>
<td>Oscillation</td>
<td>4</td>
</tr>
<tr>
<td>Other fault</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 30 and Figure 31 show that the classification algorithm works well for identifying the states of a normal lift and also for identifying oscillations. In Figure 31 it also detects the state Other fault which is the case when the twistlock can not reach the unlocked position on the first try (the same case as discussed in Section 4.4 when referring to Figure 29). However, some cases where it does not identify
Figure 30: A classified single lift for node 7 when an oscillation is present. In the upper figure locked, unlocked and landed extracted from signal 187 are plotted and in the lower figure the corresponding classification is plotted.

Figure 31: A sequence of classified lifts for node 7. Note the identified fault (classification 5) at the right end. In the upper figure locked, unlocked and landed extracted from signal 187 are plotted and in the lower figure the corresponding classification is plotted.

the state correctly have been observed. An example of this for node 6 can be seen in Figure 32. The system starts in state Lifting (3). When landed goes to 1 the system prepares to unlock. However, an oscillation in landed occurs after locked has gone to 0 but before unlocked has gone to 1. This can be considered to be a special case which the classification algorithm is not yet designed to handle.

Apart from the apparent fault in the classification in Figure 32 there are also examples of cases where it is not certain how the classification should be defined. In Figure 33 signal 187 shows a similar behaviour to that of signal 186 in Figure 32, but here the oscillation occurs before landed changes from 1 to 0. The possible confusion arises from whether the time for which landed=1 before the oscillation should be defined as an oscillation as it is here or if it should be defined as the start of the Unlocking state with only the sudden drop being the oscillation. How this is defined is important because it affects the duration of the unlocking process which is suggested to be used as a feature for estimating the RUL in Section 4.6. The second approach, defining just the sudden drop as the oscillation, seems most reasonable but this definition could create other problems. An example of such a problem is the case when there are several oscillations occurring closely spaced before the system can unlock (or lock) properly (see for example Figure 28) and in this case how does one define when the locking/unlocking begins?
Further work with the algorithm is necessary to make it work for all possible cases (if this is even possible). In its current form the algorithm can identify most common behaviours and the known “faults”. It could be used as an offline diagnostic algorithm if fed with data from the system in the correct format. Since the aim of the work is not to develop a diagnostic but a prognostic tool no further work was done to improve this algorithm and it will be used in its current form only as a classifier.

4.6 Possibilities for prognostics and estimation of the RUL

There are two big problems related to using the data set in its current form for estimation of the RUL for the system. The most severe of these is the insufficient amount of data in time. The time stamps for the data series are measured in seconds and the whole data file contains measurements for around 117900 seconds which corresponds to approximately 33 hours. There is also a silent period starting at $t = 21000$ seconds and lasting until $t = 70000$ seconds which corresponds to 13.5 hours. Hence the data represent only around 20 hours of operation, during which around 280 lifts are performed. A twistlock can be expected to have a lifetime much longer than 20 hours/280 lifts. Its degradation is mainly due to corrosion and this degradation process requires more than a few hours to be detected. Hence it can be concluded that the present dataset is not sufficient for performing estimation of the RUL of the system.
However, the data was analysed further anyway to support conclusions about further work for RUL estimation for Bromma Conquip’s spreader.

The second problem related to using this data is that the variables _landed, locked_ and _unlocked_ are binary and hence they are difficult to use directly for estimation of the RUL using the prognostics mentioned in this thesis. Most algorithms for estimation of the RUL require data which can take more than two values for the in- and outputs. Since the spreader data contains three binary variables and the continuous variable _Time_ these can be combined to estimate the time spent in the states _Locking_ and _Unlocking_ and this estimated time can then be used as a feature instead. The reason for using the time in these two states is that the time spent in these states are fairly constant provided that the locking and unlocking runs smoothly. The time spent in the states _Idle_ and _Lifting_ is more varied which is reasonable since those times depend on the distance the containers are lifted and how often lifts are performed.

The computation of the time for locking and unlocking is fairly simple if the classified data is used. The time spent in state _Locking_ is computed by looping through the data. When a data point classified as 1 is encountered a timer is started which runs as long as data points are classified as 1 or 4 (oscillation). The same is true for the time spent in the state _Unlocking_ with the only difference being that the timer runs as long as data points are classified as 2 or 4. The oscillations are included in the computation since they sometimes occur during the locking or unlocking process. A possible problem with this computation is when the spreader has landed (_landed_=1) and an oscillation down to 0 occurs before the variables _unlocked_ and _locked_ change values. As discussed in Section 4.5, that first landing will be classified as an oscillation and hence the estimated time for locking/unlocking will be shorter than it really is (see Figure 33). Another possible problem is of course if the classification is erroneous. The script for computing these times can be found in Appendix B where a calculation of the time in the _Lifting_ state is also included.

In Figure 34 a plot of the estimated times in the _Locking_ and _Unlocking_ states can be seen for node 4. In these plots all estimated times are included except for two values for _Locking_ and six values for _Unlocking_ which for some reason was much longer (20-300s) than the others. Note also that the empty gap corresponds to the silent period present in the data. From Figure 34 it is clear that the duration for state _Locking_ is approximately 5-7 seconds whereas the duration for state _Unlocking_ is approximately 5 seconds for most lockings/unlockings. There is a quite big spread on the values with some lockings taking up to 20s (excluding the max values) and unlocking taking up to 30 seconds (excluding the max values). Note that this spread is asymmetric (only upwards in Figure 34) and this is thought to be, at least partly, due to problems with the classification of the states which in turn affects the estimation of the duration for locking/unlocking.

![Figure 34: Estimated duration for locking and unlocking for each lift plotted against time when the action occurred. The empty gap corresponds to the silent period present in the dataset.](image)

To better visualise the spread in the times for locking/unlocking the estimated times were sorted and then plotted in Figure 35. In total 278 lifts were performed based on the number of estimated times for locking/unlocking for node 4. A majority of these lifts had a duration for state _Locking_ between 5 and 7 seconds (232 of 278 lifts corresponding to 83 percent) and a duration for state _Unlocking_ between
4 and 6 seconds (218 of 278 lifts corresponding to 78 percent). This result indicates that the time to
lock/unlock should stay within a certain interval (approximately 5-7 and 4-6 seconds for locking and
unlocking respectively) when the system operates normally. If this is true it could be possible to use
the time to lock/unlock as a performance measure to compute the RUL for the twistlock using one
of the methods in Section 3. This result should be further investigated using data from the spreader
during a longer time span where it might be possible to detect a degradation in performance. This
could be done for example by collecting measurements during a few lifts every day/week/month for an
appropriate time-span (based on how long the spreader needs to be in operation before a degradation
can be observed). Preferably these measurements should be without oscillations in the landed variable,
since the oscillations are thought to influence the classification (see Section 4.5) and hence the spread in
the time to lock/unlock which can be observed in Figure 34. If this spread could be reduced a higher
accuracy for the time interval for locking/unlocking could be achieved which will be important for a
future estimation of the RUL.

4.7 Main results of the case study and suggestions for future work

When analysing the data from Bromma Conquip it was concluded that during each lift the spreader goes
through four distinctive states: Idle, Locking, Lifting and Unlocking. It is also possible to distinguish
oscillations in the variable landed as a common but not very severe fault. Apart from this there are a few
examples of other, perhaps more severe, faults such as problems unlocking after landing. Based on the
four states for each lift and the two types of faults a classification algorithm was designed. Apart from
some issues due to oscillations in the landed variable the classification algorithm can correctly classify
the state of the spreader and can hence be used as an offline diagnostic algorithm. The data obtained
from Bromma Conquip was not sufficient for estimation of the RUL of the spreader since it only covers
around 280 lifts during 20 hours of operation which is a too short time-span for observing a degradation
in the performance. Another issue was that the measured variables (landed, locked, unlocked) are binary
and hence difficult to use for prediction of the RUL. A possible solution is to combine these variables
with the variable Time and the classification algorithm to determine the time spent in the states Locking
and Unlocking respectively. These estimates of the time can then be used for estimation of the RUL.
The results indicate that this way of estimating the time in each state works quite well for a majority of
the observed lifts. Therefore it should be further investigated if/how this can be used for estimation of
the RUL for a series of measurements covering a much larger time-span.

5 Results and discussion

The aim of this thesis work was to investigate the possibilities for implementing CBM on CrossControl’s
system through a literature study of different concepts related to CBM and an implementation of one
or more of the studied methods using data from one of CrossControl's customers. The result is one
literature study on the maintenance concept in general with focus on CBM, and one literature study
on different algorithms for estimation of the RUL of a system (which is one of the steps of CBM). A
case study of the twistlock system from CrossControl's customer Bromma Conquip was also performed
to investigate the possibilities of implementing CBM for this system using data from test runs with a
real spreader. Due to an insufficient amount of data no method for CBM could be implemented, but a
thorough analysis of the data was performed and some ideas for future work are presented.

In the following subsections a more detailed description and discussion of the main results can be
found along with recommendations to CrossControl for future work with implementing CBM.

5.1 Comments on the results presented in this thesis

The literature study on maintenance theory (Section 2) is intended as an introduction to the maintenance
concept. It explains the three types of maintenance and their respective strengths and weaknesses.
Depending on the system and its failure pattern one of the maintenance types might be better suited.
Therefore the choice of maintenance strategy is the first thing that should be considered, especially since
studies have shown that not all companies will benefit from implementing a more advanced maintenance
strategy such as CBM. For very simple systems or systems which cannot be repaired the Corrective
Maintenance is the best choice. For systems where the probability of failures increases with the age of
the system a Time-Based Maintenance is more appropriate. For more complicated systems or systems
where the failure probability is constant with respect to time Condition-Based Maintenance will work
best. Condition-Based Maintenance is the main focus of this thesis and therefore it is examined in more
detail. The three steps data collection, prognostics and cost optimisation are explained in detail and a
whole section is dedicated to methods for estimation the RUL of a system.

The literature study on methods for estimation of the RUL (section 3) is intended as a collection of
methods to choose from when learning about and implementing CBM. It considers the most frequently
used methods for RUL estimation in detail and explains their advantages and shortcomings. It also gives
one or more examples on how each method can be used for estimation of the RUL. The different methods
for RUL estimation are split into two groups; physical models and models based on learning. The physical
models are based on algebraic and/or differential equations (laws of physics) and hence require a thorough
knowledge of the system. If it is possible to find an accurate model of this type it will outperform any
other model, hence if one of CrossControl's customers uses a system that can be described using laws
of physics, this will be the preferred type of model. The problem however is that this type of model is
normally very hard and/or expensive to develop for a big system. It is mostly used to model smaller
subparts of a system, such as a bearings or batteries. The methods based on learning on the other hand
require very little or no knowledge about the system. Instead they learn a model based on observed
data from the system and are either supervised (both the input and output are known) or unsupervised
(only the input is known). This group of methods is very diverse, it includes both linear and nonlinear
models and depending on the choice of method many different limiting assumptions can be made about
for example the noise in the data (Gaussian, non-Gaussian, gamma distributed etc) or the behaviour
of the system (monotone degradation, Markov process etc). Hence when choosing which learning-based
model to use it is important to first analyse which type of assumptions that can be made about the
system at hand. As an example, if the system seems to be accurately described by a linear model with
Gaussian noise the Kalman Filter is a good choice for modelling the system whereas if the system is
degrading monotonically with gamma distributed noise regression using the gamma process is a better
choice. However, there are two important limitations for application of learning-based algorithms for
estimating the RUL. Firstly, these methods can be computationally expensive compared to the physical
models. Depending on the possibilities for computational power in a system this can severely limit the
applicability of these methods. For CrossControl this could be an issue since the computational power
of the display computers is already well-used, hence one might have to consider letting the customers
compute the RUL estimation on their system and then provide CrossControl's system with the result
(details on this in Section 5.2). Secondly, these methods require a sufficient amount of data which is rich
enough to properly describe the degradation of the system in order to give accurate estimations of the
RUL. Hence if learning-based methods will be utilised the system must first be thoroughly analysed to
determine which data that should be collected/measured. If data is already collected from the system it
is of course also possible to analyse if it can be used for RUL estimation, as was done in the case study in
Section 4. The case study is also a good example on why it is important to collect a sufficient amount of
data for the RUL estimation; if the data does not contain any indication of degradation a learning-based
algorithm will be futile (except perhaps for learning what is a normal behaviour).

The case study (Section 4) considers the possibilities of implementing CBM for the system of one of CrossControl’s customers Bromma Conquip. It was originally intended to include both an analysis of the available data and a test implementation of one or more of the methods for RUL estimation based on the characteristics of the data. However, due to a limited amount of time for the thesis work and too little available data (just 20 hours of operation) only a thorough analysis of the data was performed. Despite not being able to implement any method for RUL estimation the analysis of the available data contributed to understanding the process of implementing CBM and in particular some difficulties that can be encountered. Firstly, it is very difficult to give any type of directions as to what type of data that should be collected from the system. In this case (the twistlock) the raw data used can be considered to be event data since the variables locked, unlocked and landed as well as the commands sent from the crane (signal 202 and 203) all describe something that happens at a certain time. These are then used to compute the time spent in the states Locked and Unlocked which in turn can be considered to be condition monitoring data. Since many of the methods used for RUL estimation primarily uses condition monitoring data this transformation from event to condition monitoring data can be useful. But the particular transformation performed here is specific for this system and can hence not be used in general. Secondly, the preprocessing of the data is important and might be more involved than first anticipated. In particular it is important to understand how the data was collected and saved. In this case the relevant signals had to first be extracted from the original data file and then transformed from a hexadecimal form to a binary form. The binary variables (locked, unlocked and landed) were then analysed together to identify the different states this system could be in during the lifting process and also which faults that might occur. Based on the characteristics of each state the time in each state (Locked and Unlocked in particular) could then be extracted. The time in each state is intended to be used for estimating the RUL of the system when a sufficient amount of data can be obtained. The preprocessing of the data is evidently very system dependent, the analysis performed here can not be used on another system with different characteristics. A more detailed discussion of the results of the case study can be found in Section 4.6 and 4.7.

5.2 Recommendations for CrossControl

There are two difficulties associated with implementing CBM on CrossControl’s system which must be considered when formulating the recommendations. Firstly, CrossControl’s system have a limited amount of computational power which might restrict the choice of method for implementing CBM. Secondly, during the early phase of the thesis work it became clear that CrossControl would like to have a quite general implementation of CBM on their system, which can easily be tuned to fit each customer’s needs. Ideally a solution somewhat like the existing one for diagnostics with a few predefined blocks to choose from and a possibility to implement their own blocks was desired. When more was learnt about CBM and its associated methods it became apparent that this criteria is hard to meet in practise since CrossControl have customers in many different industrial sectors which means that the system of one customer will be quite different from the systems of the other customers.

The first difficulty implies that it can be troublesome to use learning-based methods for the RUL estimation since these normally require some computational power. A physical model could in this case work better. However, considering the second difficulty it is clear that it might be necessary to implement one physical model for each customer (provided it is even possible to develop a physical model) which is a clear disadvantage. To handle this second difficulty a learning-based model might work better since they are more general. One alternative way to handle both the difficulty related to the limited amount of computational power and the difficulty related to the many different systems of the customers is to let the customers perform the RUL estimation in their own system (which might have more computational power) and then implement only the cost optimisation on CrossControl’s system. CrossControl would then have to provide their customers with information on how to perform the RUL estimation and perhaps even help them to do this. Another possible way to handle the two difficulties is to find some common feature in all customer’s systems which can be used for RUL estimation. One such feature could be the time it takes the system to perform some crucial action, which is expected to become longer while the system degrades. For Bromma Conquip’s twistlocks this time could correspond to the time it takes to lock or unlock the containers whereas for John Deere, another of CrossControl’s customers, the time to cut a tree trunk (which is also expected to grow as the cutting blade degrades) could be used. Adjustments of a model to fit each specific system would still have to be made but it could greatly facilitate the design process if the same feature could be used for all customers. If a
design similar to the existing blocks for diagnostics is desired another idea is to implement one or more of the physical models (e.g. describing some common type of wear process) as well as one or more of the simpler learning-based models (e.g. linear regression) as the predefined blocks and then allow the customers to implement more advanced models if that seems to be required. The further work for a CBM implementation on CrossControl’s system can be divided into three parts; a customer survey, more research on CBM methods and a more involved analysis of how CBM can be applied to Bromma Conquip’s spreaders. The customer survey should both investigate the interest among all CrossControl’s customers for implementing CBM and, if they are interested, check what type of data they are currently collecting from their system. For example, do they collect any data that can be used to compute the time it takes to perform some important action (like locking/unlocking twist locks in Bromma Conquip’s case)? It is also important to have a thorough understanding of each customers system to determine which of the methods for estimating the RUL that could be appropriate to use.

A more thorough analysis of methods for estimating the RUL is also necessary. In the literature study in Section 3 a selection of methods are presented, but these are just a subset of all available methods. If one or more of them is to be used for implementation it might be necessary to do some more involved research on that/those particular methods since the literature study does not cover the methods in detail. In particular, more research on available methods for the cost optimisation is required. Due to the time limitation of the thesis work it was not possible to cover these in detail, so if CBM is to be implemented it will be important to first learn more on how to do this optimisation. Particularly since optimisation normally requires some computational power from the system and this is limited in CrossControl’s case.

A further analysis of Bromma Conquip’s system would aid the understanding of how to implement CBM and would also show if the time for locking/unlocking is a good feature to use for computing the RUL of the system. First, data from a spreader which covers a longer time span must be collected, so that a degradation in performance can be identified from the data. Ideally the time span should be of a length comparable to the mean lifetime of a twistlock. If very many lifts can be performed before a degradation can be observed it might be possible to use only a subset of all lifts for the analysis. With a longer data set it will be possible to further test and develop the model for classification of the lifts and to validate if the method to estimate the time it takes to lock/unlock the twistlock is accurate. An idea is also to use data from another spreader to see if the oscillations are spreader specific or if there is a problem with the sensor for measuring if the spreader has landed on all spreaders. When the classification algorithm and the estimation of the time to lock/unlock give satisfying results one could move on to trying to estimate the RUL of the system. As a first implementation it is normally a good idea to try the simplest methods first. Therefore a recommendation would be to try to model the RUL of the system using for example linear regression or a gamma process depending on the characteristics observed. If the system can be described using a state space model the Kalman Filter might also be a good first approach. When an estimation of the RUL can be computed one can move on to implementing the cost optimisation.

6 Conclusion

The aim of this thesis was to investigate whether it will be possible to implement CBM on CrossControl’s system by studying relevant theory on CBM and by performing a case study with data from one of CrossControl’s customers.

The result of the thesis work is a literature study on maintenance theory, a literature study on how to estimate the RUL of a system and a case study on the twistlocks in Bromma Conquip’s spreader system, which together forms an introduction for how to implement CBM. The literature study on maintenance theory introduces the different types of maintenance and explains CBM in detail. The literature study on estimation of the RUL contains a collection of the most commonly used methods for RUL estimation, which is one of the steps for performing CBM. It aims to explain how each method works, how the method can be used for RUL estimation and some of its advantages and shortcomings. The case study on the twistlocks in Bromma Conquip’s spreaders provides an example of issues that must be handled while implementing CBM in practice. Due to a limited amount of time for the thesis work and too little data it was not possible to implement any of the RUL estimation algorithms, but a thorough analysis of the available data was performed and some ideas for a future implementation are given. In conclusion, the hope is that this thesis can provide CrossControl with the necessary information and ideas for continued research and implementation of CBM on their system.
7 References


[86] Interview with Fredrik Löwenhielm, Bromma Conquip, 2016-04-20.


[95] Peck & Hale, "Peck & Hale LLC Maritime Product Catalog".


A Additional algorithms for estimation of the RUL

In this Appendix a brief introduction to some other methods for estimating the RUL of a system are presented. These methods are in no sense less useful than those presented in Section 3 but due to the limited time for this thesis work only some methods could be examined in detail. Note also that there of course are many more methods for estimating the RUL of a system than those presented here and in Section 3.

A.1 Statistical Process Control

Statistical Process Control is a very simple method used to both diagnose and predict failures. It originates from control theory. It measures the deviation of the signal from a preset reference signal. The reference signal represents what the output signal during normal operation should be. If the output signal deviates more than a preset threshold level from the reference signal a failure may be approaching [19].

A.2 Other types of Regression

Random Coefficient Regression is a type of regression which uses the condition monitoring data to find a path for the data and then infer a lifetime distribution based on this. This method will give a pdf for the RUL, but this pdf will only be on closed form for special cases. A more common approach is to use simulations to arrive at an approximation of this pdf [23].

Partial least squares is another type of regression which aims to find a linear regression relation between projections of the inputs \( X \) and the outputs \( Y \). Mathematically it is given by the following formulas

\[
Y = UC^T + F
\]  

(A.1)

where \( U \) is the score matrix for \( Y \), \( C \) is the loading matrix for \( Y \) and \( F \) is a residual matrix. The score matrix \( U \) has a relation to the score matrix \( T \) for \( X \) given by

\[
U = TB + R
\]  

(A.2)

where \( B \) is a matrix of regression coefficients and \( R \) is a residual matrix. The PLS model is then given by

\[
Y = TC^T + G
\]  

(A.3)

where \( G \) is the total residual matrix [99].

Other commonly used types of regression include Gaussian Process Regression, Ridge Regression and LASSO.

A.3 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method which can be used to classify data. It assumes that the classes have a Gaussian distribution with some mean \( \hat{\mu}_k \) and covariance \( \hat{\Sigma} \) and aims to separate the classes using a linear decision boundary. For a certain test input \( x \) it will classify it as belonging to the class \( k \) for which the discriminant function

\[
\hat{\delta}_k(x) = x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k
\]  

(A.4)

is largest. The mean \( \hat{\mu}_k \), the covariance \( \hat{\Sigma} \) and the prior probability \( \hat{\pi}_k \) for class \( k \) are all estimated from data. If the decision boundary is thought to be nonlinear it is possible to instead use the related method called Quadratic Discriminant Analysis (QDA). This method assumed that each class \( k \) have its own covariance matrix \( \hat{\Sigma}_k \) which results in a slightly more complicated discriminant function \( \hat{\delta}_k \) and a quadratic decision boundary [37].
A.4 The Hidden Semi-Markov Model

The Hidden Semi-Markov Model (HSMM) has the same structure as an HMM with an additional temporal component which makes it possible for each state to send out a sequence of observations. This is done by replacing the duration probability density with a properly chosen function which is close to the real durational distribution. Hence the transition probability is no longer fulfilling the Markov criteria, the unobservable state process is a semi-Markov chain [3, 23].

The HSMM has both the flexibility of semi-Markov models with respect to temporal structures, and the flexibility of HMMs with respect to approximating a complicated probability density [3, 23].

The literature on this topic is very limited and more research is needed. The HSMM, much like the HMM, does not give a pdf for the RUL, it merely gives the mean and variance. It is a more powerful tool for estimating the RUL than the HMM, but it gives a more complex estimation of the parameters [23].

A.5 Expert Systems

Expert systems, also called rule-based or case-based systems, are designed to simulate the reasoning of the human mind by utilizing knowledge from human experts to form rules which are then used to find a solution. This method works very well for problems which would normally be solved by human experts/specialists. It transforms the knowledge into rules on the form IF... THEN ... . These rules can then be combined to produce an output. This technique is fairly old, dating from the 1960s and have traditionally been used for diagnosis and prognosis [3, 19].

A possible problem with Expert Systems is that depending on the type of knowledge it may be hard to transform it into rules. When all rules are set the system is not adaptive, so if new information surfaces it may be hard to incorporate it into the model. If the system requires a large amount of rules to be fully described a combinatorial explosion can occur [3]. A possible problem is also that it may be hard to decide how credible the knowledge is when it comes from humans/experts [19].

A.6 Independent Component Analysis

Independent Component Analysis (ICA) is an extension of Principal Component Analysis. It aims to split a signal into independent components which are non-Gaussian. Just like the Principal Component Analysis this is an unsupervised method [19].

A.7 Clusters

This method groups signals together into different fault categories based on statistical similarities in the features, hence it is unsupervised. It is done by aiming to minimize the variance within a group and maximize it between different groups. This creates heterogeneous groups with homogeneous contents [19].

A distance measure is required to measure how similar two signals are. Two examples of distance measures that may be used are Euclidian distance and Bayesian distance. It is also possible to base the division into clusters on the correlation between signals or by using the nearest neighbour method [19].

A.8 Swarm algorithms

This group of methods are, as the name indicates, inspired by the behaviour of insects. The method creates an intelligent multi-agent system. It is built on the idea that a very simple component/element can handle difficult tasks when working in a group. The system usually lacks central control (no one supervises the system). Examples of algorithms are Ant Colony Optimisation (ACO) which solves discrete optimisation problems by finding the shortest way, and the Bee Colony Algorithm (BCA) which is an optimisation algorithm built on the dancing of bees [14].

A.9 Learning Vector Quantisation

Learning Vector Quantisation (LVQ) use class information to move the so called Voronoi vectors, which are vectors that can characterise a class. It is a supervised learning technique which aims to improve the classifier in the decision regions. Mathematically an input vector $x$ is chosen from the input space randomly. If the labels for the input vector $x$ corresponds to the labels for the Voronoi vector $w$, the
Voronoi vector will be altered to be closer to $x$. If this is not the case it is altered to move away from $x$ [3].

B MATLAB code

B.1 ReadFileGet1.m

This MATLAB code reads a csv file on the format displayed in Figure 23 and extracts a signal with a certain $id$. In the code included below the signal 185 is filtered out and the relevant data is saved in the matrix $A_{185}$. This code works for the signals 184-187 where the hexadecimal data saved as $Data0$ must be transformed to binary data to extract the relevant information.

```matlab
% Choose and open file
[fileName, pathName] = uigetfile('*csv', 'Select the csv file');
fileID = fopen(fileName, 'rt');

% Initialize vectors for saving data
vSize = 300000; % Size of the vectors where data is saved
Time185 = zeros(vSize, 1);% Time
Landed185 = zeros(vSize, 1);% Landed
Locked185 = zeros(vSize, 1);% Locked
Unlocked185 = zeros(vSize, 1);% Unlocked

% Extract header
header = fgetl(fileID);

% Index for counting number of values saved for signal 185
i185 = 0;

% Loop through the file and extract only signal 185
while ~feof(fileID)
    nLine = fgetl(fileID);
    [Time, channel, id, flag, DLC, Data0, Data1, Data2, Data3, Data4, Data5, Data6, Data7, counter] = strread(nLine, '%s %s %s %s %s %s %s %s %s %s %s %s %s %s', 'delimiter', ',');
    if strcmp(id, '185')
        i185 = i185 + 1;
        Data0185 = hexToBinaryVector(Data0); % Transform hexadecimal data to binary
        Landed185(i185) = Data0185(8);
        Locked185(i185) = Data0185(6);
        Unlocked185(i185) = Data0185(5);
        Time185(i185) = str2double(Time);
    end
end

% Save all extracted data in the matrix A185
A185 = zeros(i185, 4);
A185(:, 1) = Time185(1:i185, 1);
A185(:, 2) = Landed185(1:i185);
A185(:, 3) = Locked185(1:i185);
A185(:, 4) = Unlocked185(1:i185);
save(’A185’, ’A185’);

% Close file
fclose(fileID);

For signal 202 and 203 a simpler version of the code above can be used since the hexadecimal data must not be transformed into binary data. The code for signal 202 and 203 can be found below.

```
% Extract header
header = fgetl(fileID);

% Index for counting number of values 202
i_202 = 0;

% Loop through the file and extract only signal 202
while ~feof(fileID)
    nLine = fgetl(fileID);
    [Time, channel, id, flag, DLC, Data0, Data1, Data2, Data3, Data4, Data5, Data6, Data7, counter ]...
        = strread(nLine, '%s %s %s %s %s %s %s %s %s %s %s %s %s %s', 'delimiter', ',');

    if strcmp(id, '202')
        i_202 = i_202 + 1;
        Data_202(i_202) = str2double(Data0);
        Time_202(i_202) = str2double(Time);
    end
end

% Save all the extracted data in the matrix A_202
A_202 = zeros(i_202, 2);
A_202(:,1) = Time_202(1:i_202);
A_202(:,2) = Data_202(1:i_202);
save('A_202', 'A_202');
fclose(fileID);

B.2 ClassifyData.m

The MATLAB code below is used for the classification of the data for signals 184-187. It assumes that the data has been saved in a matrix on the form produced by the code ReadFileGetl.m as displayed in Appendix B.1.

Time_187 = A_187(:,1);
Landed_187 = A_187(:,2);
Locked_187 = A_187(:,3);
Unlocked_187 = A_187(:,4);

L = 118000; % Maximum size of data
var_187 = zeros(1,L); % Variable for saving the classification
i=1; % Index for looping through data

% Loop through all data
while i <= L
    if Landed_187(i) == 0 % If spreader is in the air
        if ( Locked_187(i) == 0 & Unlocked_187(i) == 1 )
            var_187(i) = 0; % Idle (no container)
        elseif ( Locked_187(i) == 1 & Unlocked_187(i) == 0 )
            var_187(i) = 3; % Lifting container
        end
        i = i+1; % Update index
    elseif Landed_187(i) == 1 % If spreader landed on the container
        % Determine for how long it is landed
        start = i;
        while Landed_187(i) == 1
            i = i+1;
        end
        if ( Locked_187(start) == 0 & Unlocked_187(start) == 1 ) % If unlocked at start
            var_187(start:i) = 4*ones(1,i-start+1); % Oscillation (unlocked at end, no change)
        elseif ( Locked_187(i) == 1 & Unlocked_187(i) == 0 ) % If locked at end
            if (( sum(find(Locked_187(start+15:i)))>0 | sum(find(Unlocked_187(start+15:i)))>0 )...& i-start > 15 )
                var_187(start:i) = 5*ones(1,i-start+1); % Other fault (change more than once)
            end
        end
    end
end
else
    var187(start:i) = 1*ones(1,i-start+1); % Locking
end
else
    var187(start:i) = 5*ones(1,i-start+1); % Other fault
end

elseif ( Locked187(start) == 1 & Unlocked187(start) == 0 ) % If locked at start
    if ( Locked187(i) == 1 & Unlocked187(i) == 0 & sum(find(Unlocked187(start:i))) == 0 )
        var187(start:i) = 4*ones(1,i-start+1); % Oscillation (locked at end, no change)
    elseif ( sum(Locked187(start:i) == Unlocked187(start:i))>5 )
        var187(start:i) = 5*ones(1,i-start+1); % Other fault (Locked and unlocked same)
    elseif ( Locked187(i) == 0 & Unlocked187(i) == 1 )
        var187(start:i) = 2*ones(1,i-start+1); % Unlocking
    else
        var187(start:i) = 5*ones(1,i-start+1); % Other fault
    end
end
i = i+1; % Update index
end

B.3 GetLockAndUnlockTime.m

The MATLAB code below is used for estimating the time for locking/unlocking using classified data. It is assumed that ClassifyData.m has been run prior to running GetLockAndUnlockTime.m

Time187 = A187(:,1);
var187 = var187;
i = 1; % Loop index
nrLift = 1;
nrLand = 1;
nrAir = 1;
while ( i <=length(var187) )
    if var187(i) == 1 % Locking starts
        start = i; % save start index
        startTimeLift(nrLift) = Time187(start); % save start time
        while ( i<=length(var187) & (var187(i) == 1 | var187(i) == 4) ) % If oscillation or locking
            i = i+1;
        end
        liftTime(nrLift) = Time187(i) - Time187(start); % Save time for locking
        nrLift = nrLift+1;
    elseif var187(i) == 2 % Unlocking starts
        start = i; % save start index
        startTimeLand(nrLand) = Time187(start); % save start time
        while ( i<=length(var187) & (var187(i) == 2 | var187(i) == 4) ) % If oscillation or unlocking
            i = i+1;
        end
        landTime(nrLand) = Time187(i) - Time187(start); % Save time for unlocking
        nrLand = nrLand+1;
    end
    i = i+1; % Update index
end

60
C Additional figures

In this appendix two additional examples of lifts where the behaviour of the spreader differs from a normal lift are presented. The first example considers a lift for node 4 (signal 184 and 202) which can be seen in Figure 36. The spreader starts out in the Idle state. At $t \sim 560s$ a locking command arrives from the crane (signal 202 has value 2) and the spreader goes to the Locking state. However, for some reason the spreader does not reach the Lifting state until $t \sim 1080s$. When the locking command arrives at $t \sim 560s$ locked goes to 1 and unlocked goes to 0 as they should but landed stays at 1. At $t \sim 780s$ a command to unlock comes from the spreader (signal 202 has value 1) and locked then goes to 0 whereas unlocked goes to 1. This puts the spreader in the appropriate starting position for going to the state Lifting which it eventually does at $t \sim 1080s$. After that the spreader’s behaviour is back to normal again. It stays in the state Lifting until $t \sim 1130s$ with the exception of an oscillation at $t \sim 1120s$, then it goes to the state Unlocking and then back to Idle again. It is unclear why the spreader does not go to the Lifting state after locked and unlocked have changed values. This behaviour does perhaps not seem like an actual fault since the system handles it and manages to begin the lift after some time, but the spreader is stationary for almost 500s which is not desired when efficient loading and unloading of containers is the aim.

The second example considers a lift for node 5 (signal 185 and 202) which can be seen in Figure 37. Here the spreader starts out in the Idle state and then goes to Locking at $t \sim 5740s$ when a locking command arrives from the crane (signal 202 goes to 2) and then continues to the Lifting state. At $t \sim 5830s$ the spreader lands and here the unusual behaviour starts. Within a short time span both a command to unlock (signal 202 goes to 1) and then to lock again (signal 202 goes to 2). After locking the spreader moves on to the Lifting state at $t \sim 5840s$ and then to the state Unlocking at $t \sim 5890s$ and then back to the Idle state again. Hence the spreader here lands for a short time to first unlock, then lock again and then goes on with the lift. This is of course not a fault, but depending on the reason for landing to lock and unlock it might be something that should be recorded. Note also that during this lift there is an abnormally long “oscillation” in the landed variable starting at $t \sim 5870s$. It is longer than the time where the spreader lands to unlock and lock and could become a problem if the system believes that the spreader actually landed and starts to unlock.

Figure 36: A lift for node 4 when the spreader does not lift the container even though it locks properly. In the upper figure locked, unlocked and landed extracted from signal 184 are plotted and in the lower figure signal 202 is plotted.

Figure 37: A lift for node 5 (signal 185 and 202) which can be seen in Figure 37. Here the spreader starts out in the Idle state and then goes to Locking at $t \sim 5740s$ when a locking command arrives from the crane (signal 202 goes to 2) and then continues to the Lifting state. At $t \sim 5830s$ the spreader lands and here the unusual behaviour starts. Within a short time span both a command to unlock (signal 202 goes to 1) and then to lock again (signal 202 goes to 2). After locking the spreader moves on to the Lifting state at $t \sim 5840s$ and then to the state Unlocking at $t \sim 5890s$ and then back to the Idle state again. Hence the spreader here lands for a short time to first unlock, then lock again and then goes on with the lift. This is of course not a fault, but depending on the reason for landing to lock and unlock it might be something that should be recorded. Note also that during this lift there is an abnormally long “oscillation” in the landed variable starting at $t \sim 5870s$. It is longer than the time where the spreader lands to unlock and lock and could become a problem if the system believes that the spreader actually landed and starts to unlock.
Figure 37: A lift for node 5 where the spreader lands for a short time during a lift, unlocks, locks and then starts lifting again. Also note the oscillation in the landed variable at around $t = 5870$ which has a very long duration. In the upper figure locked, unlocked and landed extracted from signal 185 are plotted and in the lower figure signal 202 is plotted.