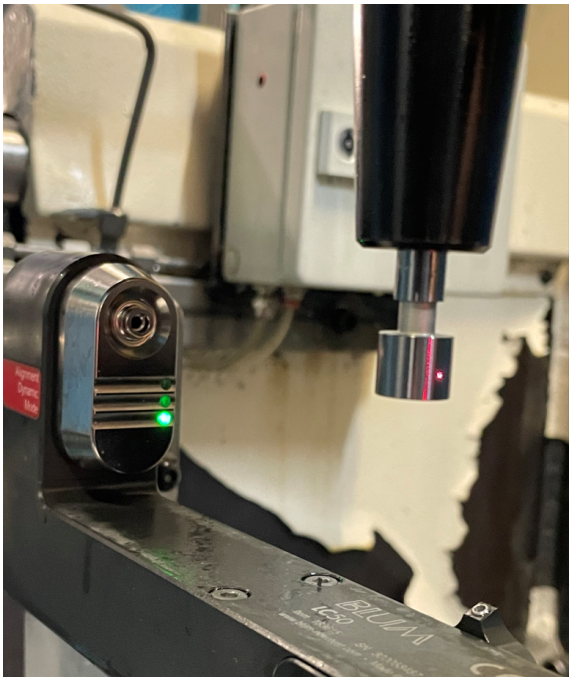
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2022
Industrial IT and Automation

Automatic data collection, visualization and predictive maintenance during probe calibration in CNC machines



Walter Johansson

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2022
Title: *Automatic data collection, visualization and predictive maintenance during probe calibration in CNC machines*
Pages: *97*
Keywords: *CNC machines, Probe calibration data, Automatic data collection, Machine learning, Predictive maintenance, Principal component analysis*

Student: *Walter Johansson*
Supervisor: *Håkon Viumdal*
External partner: *Kongsberg Terotech AS, GKN Aerospace Norway AS*

Summary:
GKN Aerospace is a world leading supplier of engine parts in the aerospace industry, for both military and commercial engine programs. One of GKN's 51 manufacturing locations is GKN Aerospace Norway AS (GAN), located in Kongsberg. GAN has numerous CNC machines that produces engine parts with strict tolerances. To measure parts and machine tools the CNC machines contain several measuring probes, needing frequent calibration. The visualization of the calibration data for analysis is a manual job that needs to be automated. Further, possibilities to utilize the calibration data for predictive maintenance would be useful for GAN. The objectives of the project is therefore to define a system for automatic collection and visualization of the calibration data, and to investigate possibilities regarding the use of Machine Learning (ML) to predict machine- or equipment health. In collaboration with GAN, a 5-axis vertical machining center (M5081) was chosen as the concept machine. A new log type with a uniform notation and setup was developed an tested on M5081. The system defined was tested locally with the same tools used in the CoPilot environment at GAN. CoPilot is GAN's chosen application for monitoring, where the new functionality is thought implemented. Regarding predictive maintenance a Nonlinear Autoregressive Neural Network was trained to predict future values of a trigger constant. The resulting prediction was found insufficient for predicting time until the next machine stop. To investigate possible correlations between the calibration data and available temperatures in the machine, PCA was performed. Considering the correlations found here and the possible implementation of a spindle vibration sensor at GAN, it is concluded that a Nonlinear Autoregressive with External Input Neural Network should be tested with temperatures and spindle vibration as inputs, to possibly obtain better predictions.

Preface

This report is written by Walter Johansson, a student attending the Industrial IT and Automation Industry Master program at the University of South-Eastern Norway. This report describes the work and research done regarding automatic collection and visualization of probe data, and possibilities to use machine learning for multistep prediction, in collaboration with GKN Aerospace Norway AS (GAN). The thesis is written for operators and engineers at GAN to increase understanding and utilization of probe calibration data, as well as investigate possibilities to monitor machine health. Prerequisite knowledge about CNC machines, programming and machine learning are assumed prior to reading this report.

The software used for planning, pre-processing and storage of data, visualization and machine learning in this project are: MS Project, Node-RED, Grafana, Visio, Visual Studio Code, pgAdmin, MATLAB and Overleaf implementing LuaLaTeX compiler. The task description and source code are attached as appendices.

The front page photo has been taken by Walter Johansson and shows a LC50 Digilog laser and Blum master tool under calibration in M5081.

Special thanks are directed to: Stefan Köwerich, CADCAM Engineer at GKN Aerospace Norway AS for the supply of data, specifications, NC-programming and other contributions to the project. Main supervisor Associate Professor Håkon Viumdal at University of South-Eastern Norway, for guidance through the project.

Kongsberg, 16th May 2022

Walter Johansson

Contents

List of Figures

List of Tables

Nomenclature

Symbol	Explanation
ABS	Absolute value
ANN	Artificial neural networks
API	Application programming interface
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CNC	Computer numerical control
ERD	Entity relationship diagram
FFT	Fast Fourier transform
GAN	GKN Aerospace Norway AS
GAS	GKN Aerospace Sweden AB
GUI	Graphical user interface
HCA	Hierarchical cluster analysis
I4.0	Industry 4.0
IOT	Internet of Things
IPG	In-process gauge
JSON	JavaScript object notation
KTT	Kongsberg Terotech AS
LAN	Local area network
MES	Manufacturing execution system
ML	Machine learning
MSE	Mean square error
M5081	Abbreviation for Machine number 5081
NAR	Nonlinear Autoregressive
NARX	Nonlinear Autoregressive with External Input
NCU	Numerical processing unit
PCA	Principal component analysis
RMS	Root mean square
RUL	Remaining useful life
RPC	Remote procedure call
SVM	Support vector machine
SQL	Structured query language
TCP/IP	Transmission control protocol/Internet protocol

Symbol	Explanation
UTF-8	Unicode transformation format - 8 bit character encoding

1 Introduction

This chapter gives an introduction to the background of the project, a system description of the desired automatic data collection, objectives for the project including predictive maintenance and methods used to achieve this. The structure of the report is listed at the end of the chapter.

1.1 Background

GKN Aerospace is a global business in the aerospace industry and the world's leading multi-technology tier 1 aerospace supplier with facilities all over the world, serving over 90% of the world's aircraft and engine manufacturers. GKN consists of 51 manufacturing locations in 14 countries, including 14 manufacturing sites only in Europe. Among these 14 manufacturing sites is GKN Aerospace Norway AS (GAN), located in Kongsberg, with about 350 employees. GAN manufactures several components for jet engines and gas turbines for the world's largest air craft manufacturers. GAN participates in several commercial and military engine programs and specialises in production of shafts, cases, turbine exhaust cases, turbine rear frames and vanes. [1]

When manufacturing these high precision parts with strict tolerances for customers in the aerospace industry, high precision CNC-machines are needed. GAN holds about 100 different CNC-machines in their factory, including milling, turning and grinding machines among others. These CNC-machines produces parts with tolerances down to 1 micrometer. To be able to deliver parts that satisfies these strict tolerances the CNC-machines are equipped with different types of probes for measuring the tools in the machines and the parts being produced. To ensure accurate measurements and machining, these probes are calibrated several times a day. These calibrations forms the basis for the work and research in this project.

When running the measurement cycles for calibrating these probes machine coordinates are read and stored. In this context these values are called trigger constants. Meaning the position of the machine axes when the probes triggers at fixed points. These trigger constants are logged in a file on the machine after every calibration run for further inspection. The logs are then collected manually and visualized using Microsoft Excel. GAN would like to continue using these log files, but the process of collecting and visualize them

needs to be automated for the data to be utilized by the operators and engineers at GAN. Further GAN would like to know if collection and visualization of this data can be used for predictive maintenance purposes, considering that maybe this repetitive measurement data can give a status on the machine- or equipment health.

1.2 System description

The system GAN wants to achieve is a continuation of the system they have today, but with additional functionality for collection and visualization of the calibration data. The main reasons for this is to remove the time consuming job it is to gather the data, as well as increase utilization of the data by making it available for the operators and engineers. Figure 1.1 shows the steps in the prevailing process from development of calibration data logs to manually visualize the data. The calibration cycles are run in the machine, which for this project is machine M5081. The programmers programming the CNC-machines (NC-programmers) writes the appropriate trigger constants from the different calibration methods to log files for each calibration method. These log files are stored locally on a computer on the machine. From here the files are copied manually to a shared network drive on one of GAN's servers by a GAN employee, and further visualized in a MS Excel sheet.

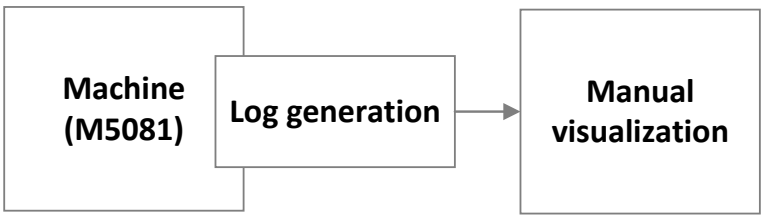


Figure 1.1: System sketch describing the existing manual functionality for transferring and visualizing log files.

1.3 Objectives and Predictive Maintenance

To be able to fulfill GAN's desired functionality with automatic collection and visualization of this data a system needs to be defined including an application for later to run on their server. This application needs to read the received log files, preprocess them and write the data to a database. From here the data can be visualized using Grafana, which is the chosen software for analytics and monitoring by GAN. Figure 1.2 shows the data flow for automatic collection and visualization of the calibration data.

Figure 1.2: System description of desired automatic functionality.

Introducing automatic collection and visualization of the calibration data will increase the utilization of it. And by making the data available in GAN's monitoring system, the operators performing the calibrations can get the results visualized within seconds. This can help increase their understanding of the machine and alert maintenance personnel of abnormal calibration values and deviant behaviour in the machine. Further the maintenance personnel setting the nominal trigger constants used for validation of the probe calibrations, can use the visualized data to justify their results after replacing or performing maintenance on the probes in the machine.

This leads to another important aspect which is predictive maintenance, where the main idea is to monitor the machine condition and predict the machines reliability. This way maintenance can be planned and performed early to maximize up-time on the machine, in relation to repairing when something is broken. In general, CNC machine maintenance can be divided into three main categories: *Preventive maintenance*, *Predictive maintenance* and *Reactive maintenance*. Preventive maintenance is typically schedule based and maintenance are performed with fixed intervals. Here issues in the machine are usually identified before they cause a stop in the machine, but preventive maintenance does not help against unforeseen machine errors. This kind of maintenance is the strategy most widely used at GAN, where maintenance are performed based on fixed intervals or hour counters on specific machine parts. On the other side of the scale is reactive maintenance, which is based on fixing the machine when it is broken and replace parts when they are worn out. Between these two strategies is predictive maintenance, which uses data from sensors in the machine to monitor the machine condition and identify possible errors before they occur in the machine. This strategy can be cost-saving for companies because

they can shorten regular maintenance stops by having spare parts available and avoid unnecessary stops. [2]

The calibration logs contains trigger constants which is machine positions read from the measurement system on the machine axes. In theory these trigger constants should be equal after every calibration run, therefore any changes in these values can directly be used to monitor the machine axes and machine probes, and possibly give an indication on the machine health. As mentioned in Chapter 1.1 GAN would like to know if these values can be used for predictive maintenance, and if there are any correlation between variables in the machine and the calibration runs being outside their tolerance. This way they can detect critical changes in the behaviour of the machine before it causes defective parts or machine stops. The work in this project will contribute with conclusions regarding the use of the calibration data to predict the machine health or equipment health, and investigate possibilities to use calibration data to foresee an error with a machine learning model.

1.4 Methods

To accomplish the goals in this project the following methods will be used: Provide general theory on the machine structure and measuring principles on a concept machine for this project. Research existing work done regarding machine learning and health monitoring of CNC machines. Define and simulate the process with automatic collection and visualization of calibration data with the tools desired by GAN. Explore machine learning methods for use to utilize calibration data and possibly predict machine stops. Optionally make a related machine learning model that can later be implemented at GAN.

1.5 Report structure

Chapter 1 gives an introduction to the background of the project, and describes objectives regarding utilization of probe data and predictive maintenance for the existing system.

Chapter 2 gives a brief description on machine theory and measuring principles for the concept machine and the calibration logs.

Chapter 3 describes the file transfer and preprocessing of the log files and how the calibration data is stored.

Chapter 4 describes the visualization of the calibration data in Grafana.

Chapter 5 gives a short overview on machine learning and describes previous work done by GAN and others regarding machine learning in CNC machines. And multistep prediction of trigger constants with a nonlinear autoregressive neural network is tested.

Chapter 6 discusses the results, solutions and suggested improvements on the automatic collection and visualization of the calibration data, and the machine learning model tested.

Chapter 7 brings conclusions to the objectives and goals of the project, and suggestions for further work.

2 Machine description

This chapter gives a brief introduction to the main components, construction, concepts and measuring principles of the concept machine used in this project. The machine is visualized with a model of the machine made in Siemens NX which is a computer-aided manufacturing (CAM) software.

2.1 Machine geometry

GAN's manufacturing site in Kongsberg has over 100 CNC-machines in different shapes, sizes and configurations. All these machines have different sets of probes and gathers large sets with data. Therefore one machine which covers several measuring principles is chosen as the concept machine for this project. This is the Forest Linè V 1250 FA (M5081) which is a 5-axis vertical machining center. A computer-aided design (CAD) model of the machine can be seen in Figure 2.1.

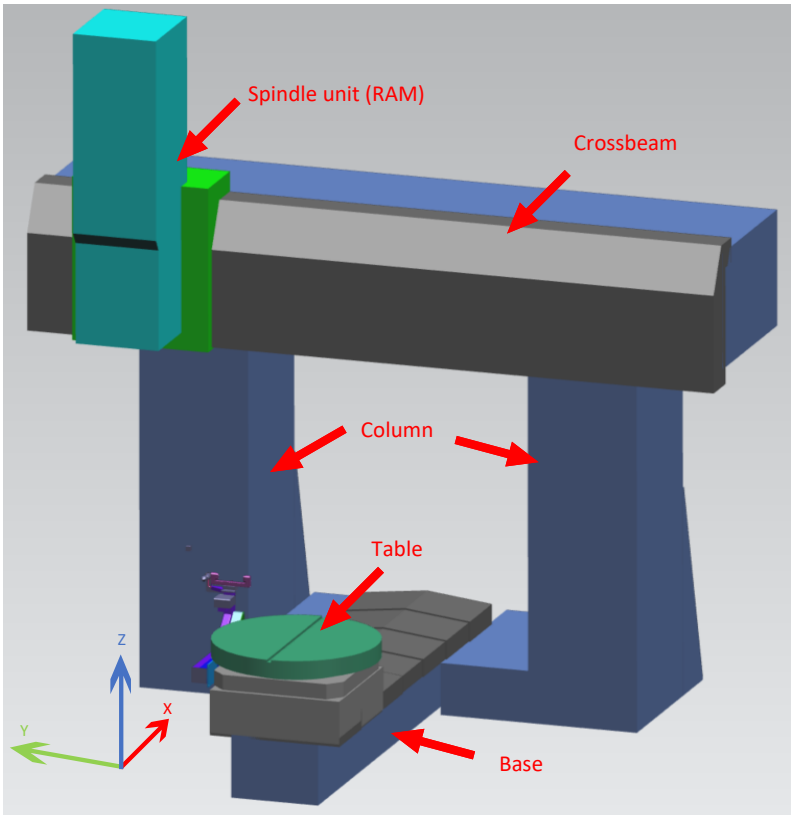


Figure 2.1: CAD model of the Forest Linè V 1250 FA (M5081) machine. [3]

The ”5-axis” term means the number of directions the tool in the machine spindle can move. In this machine the tool can move along three linear axes and two rotational axes. From Figure 2.1 the directions of the three linear axes are visualized with the three arrows in the bottom left corner. The rotating table can be moved along the base towards the column of the machine in the x-axis. The turquoise spindle box here referred to as the RAM can be traversed along the crossbeam in the y-axis, and moved up and down towards the table in the z-axis. The two rotational axes are the rotation of the table referred to as the C-axis and a rotational spindle head referred to as the B-axis. Figure 2.2 shows the model of the machine seen from the x-direction. The head with the rotational B-axis is placed in the green cart and can be changed into the machine to obtain the full 5-axis functionality. The linear axes are equipped with linear encoders used to measure the position of the axes in relation to the machines zero point, and the rotary axes have rotary encoders for speed and position control.

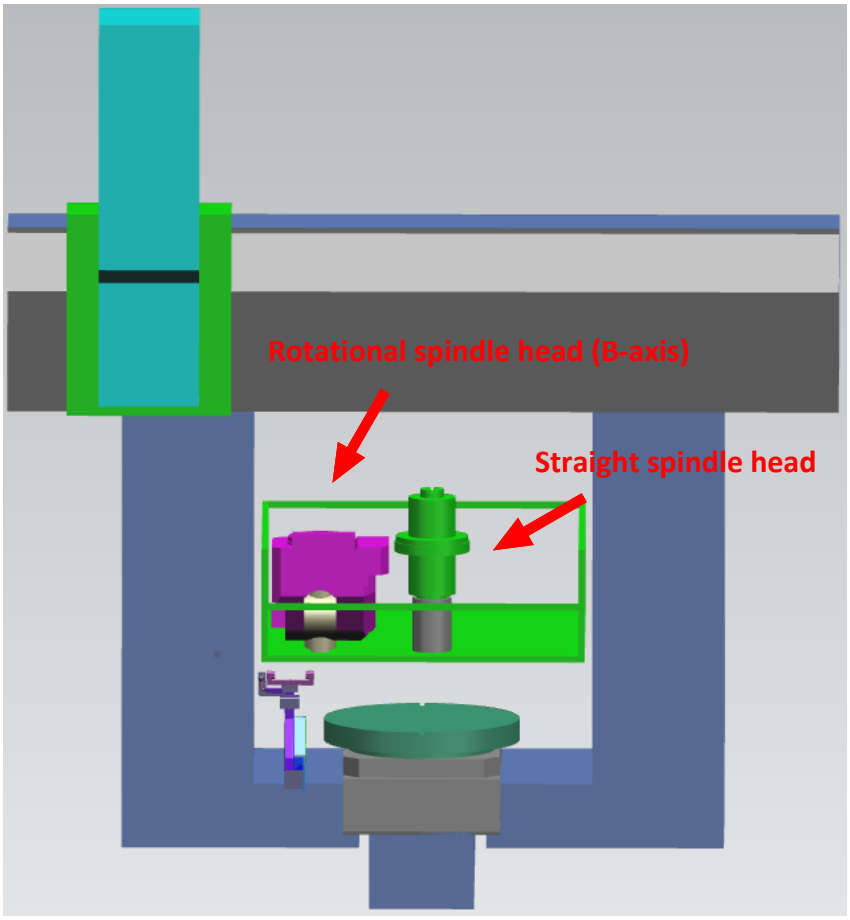


Figure 2.2: CAD drawing of M5081 with the green cart containing the rotational and straight spindle head. [3]

2.2 Measuring principles

M5081 is equipped with different probes for measuring machine tools and the parts being processed in the machine. These different measuring probes and measuring principles can be divided into four categories:

- The in-process gauge (IPG) which is commonly referred to as the part probe at GAN.
- The probes with an attached cube which is commonly referred to as tool probes at GAN.
- Blum laser.
- Light sensor.

To ensure accurate measurements with these probes they need to be calibrated frequently. This is done with various methods, and for the tool probes a master tool is used. The master tool for calibrating the tool probes have a `T` shape and touches three sides of the cube when calibrating. There are two master tools for the tool probes in machine M5081, one for each of the different cubes in the machine. One of these master tools is also used to calibrate the light sensor. For calibrating the part probes a calibration ring gauge is used. The master tools and ring gauge are visualized later in Section 2.3.

2.2.1 Part probe

The part probe used in M5081 is a Renishaw RMP600 high accuracy machine probe, which can be seen in Figure 2.3 along with the calibration ring gauge. This is the most common part probe used in GAN's machines. The part probe is attached to a tool holder which is changed into the machine spindle. This part probe is used for part set-up to verify the position of the part relative to the machine and for part verification by in-process measurements. These in-process measurements are probably more frequent in a turning machine to measure diameters of the work piece in between cuts, but can also be used in a milling machine to measure part dimensions as well as machine off sets. The rod with the small red ball in the end is called the stylus, and the stylus ball is the contact point between the parts and the measurement probe. When this ball touches the part or the calibration ring gauge a fast input on the numerical control unit (NCU) is triggered and the position of the machine axes are read and stored as trigger constants. The part probe is wireless and powered with batteries. The probe unit in the spindle is the sender and communicates with the receiver by radio transmission.

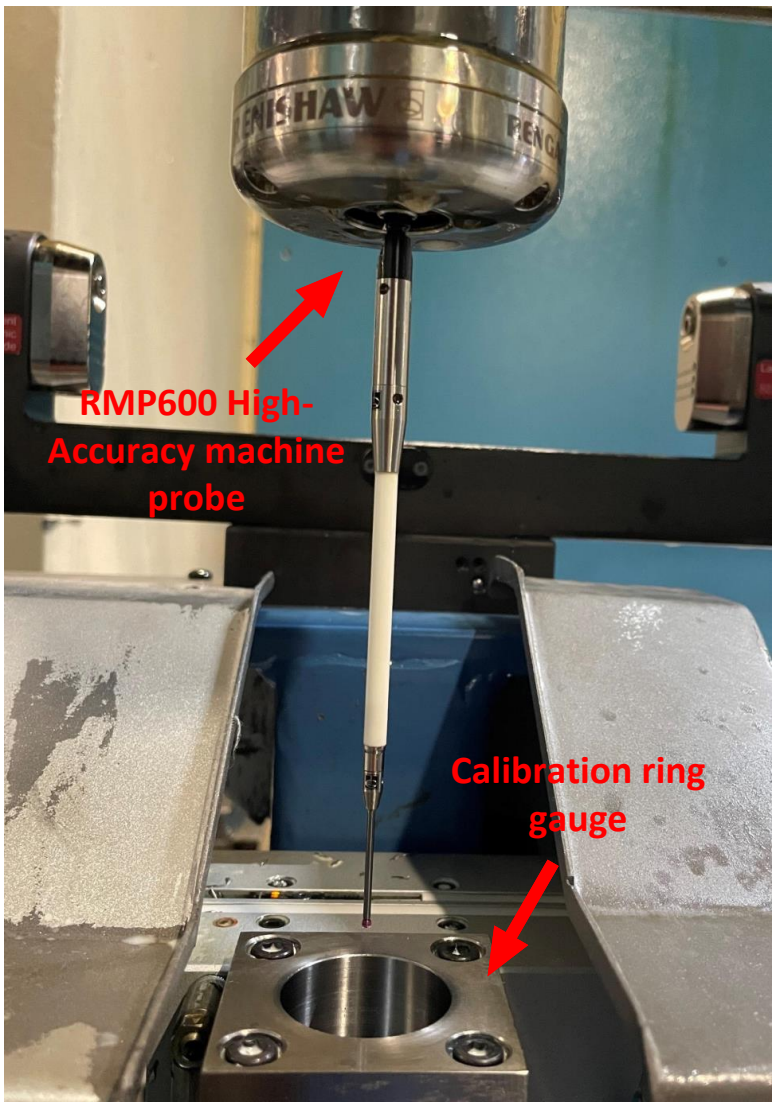


Figure 2.3: Renishaw RMP600 High-Accuracy machine probe and the calibration ring gauge. Photo: Walter Johansson

2.2.2 Tool probe

The second measuring principle is the tool probe. For machine M5081 this is a probe of type Renishaw LP2H, which is a touch probe with radio transmission as the Renishaw probe described in subsection 2.2.1. The difference is that the LP2H probe position is fixed and the stylus contains a quadratic cube. Figure 2.4 shows the tool probe with the cube attached to the stylus along with the master tool for calibration, which is changed into the spindle. The LP2H probe has a higher spring force which makes it more resistant to machine vibrations which can cause faulty trigger constants. The tool probe is especially exposed for machine vibrations because the trigger constants are set while the machine tools are turning. With known machine axis coordinates and a fixed tool probe position, the length and radius of the machine tools can be measured by touching the different sides of the cube.

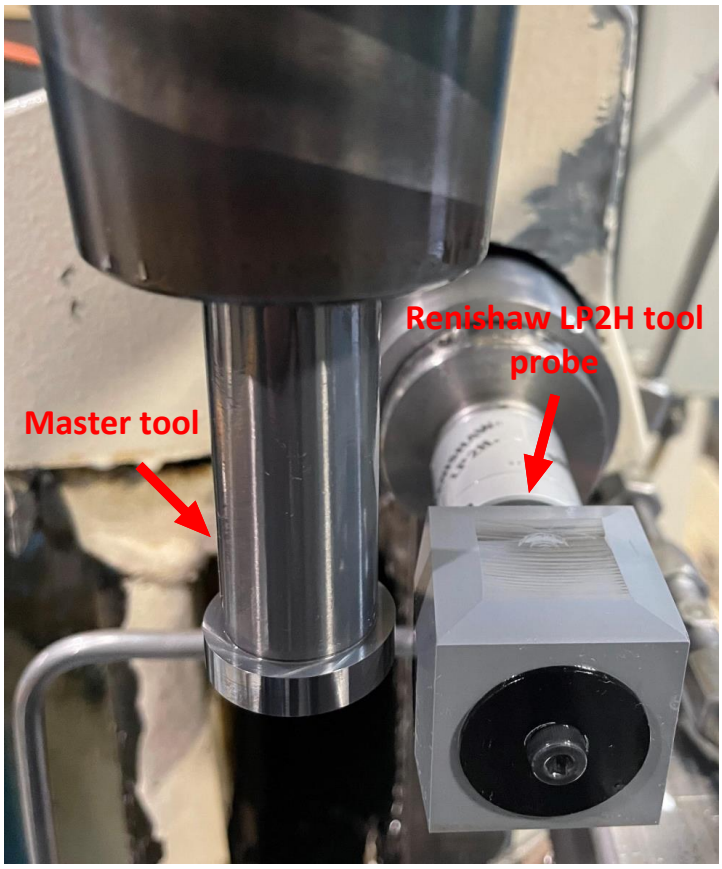


Figure 2.4: Master tool for calibration and the Renishaw LP2H tool probe with a 30x30mm cube. Photo: Walter Johansson

2.2.3 Blum laser and cube

The third measuring equipment used in M5081 is the Blum LC50-Digilog laser. The Blum LC50 laser is a non-contact measurement system with a wide range of applications within tool measurement and tool monitoring. Possible applications for the Blum LC50 laser could be monitoring of tool wear and run out measurements to detect poor tool holders. At GAN LC50 lasers are installed in several machines and are mainly used for measuring tool length and radius. The laser unit is installed on a bracket with a fixed position next to the machine table, as seen in Figure 2.5.

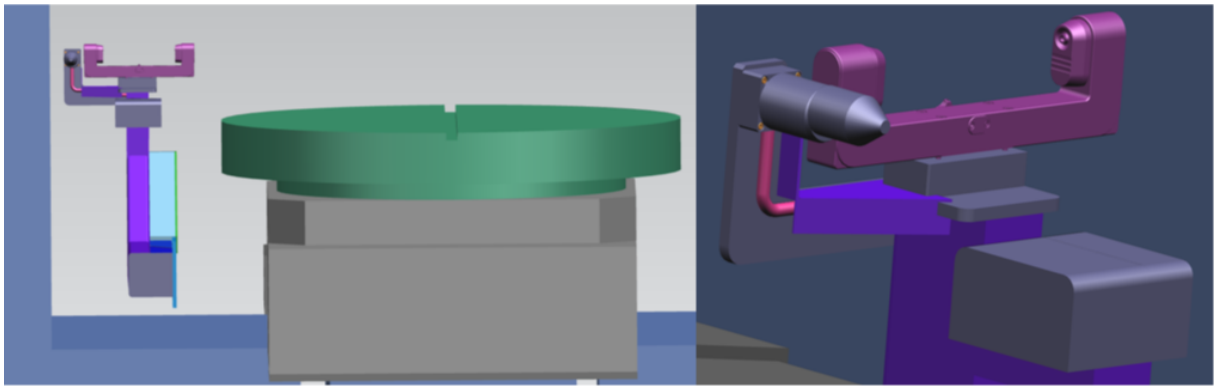


Figure 2.5: CAD drawing showing placement of the Blum laser and tool probe.

In addition to the laser unit a Blum TC76 probe is installed on the same bracket as the laser unit. The probe has a stylus with a cube similar to the Renishaw probe in subsection 2.2.2. This Blum cube TC76 works in the same way as the Renishaw probe but is primarily used to measure the alignment of angle heads changed into the spindle. Both the Blum laser and the Blum cube are calibrated with the same master tool. The Blum TC76 probe and the master tool can be seen in Figure 2.6.

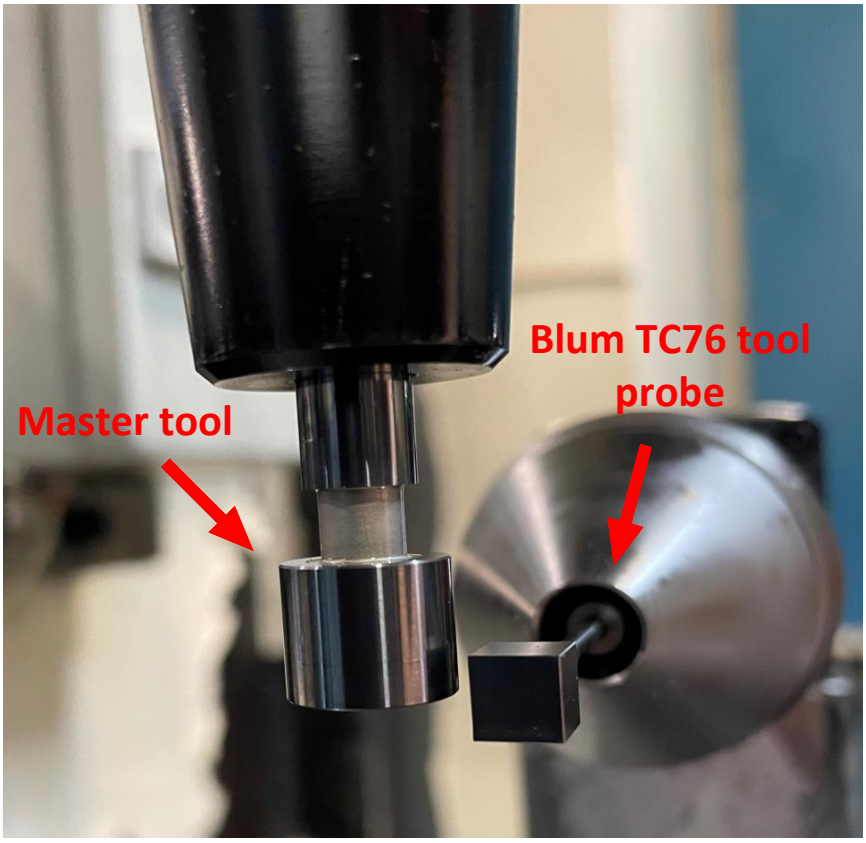


Figure 2.6: Master tool and Blum TC76 tool measuring probe with a 6x6mm cube. Photo: Walter Johansson

2.2.4 Light sensor

The light sensor consists of a Balluff BOS00WF(sender) and a Balluff BOS00WL(receiver).[4] These Balluff sensors are laser sensors with a fairly high accuracy, but is referred to as the light sensor at GAN and is used as a protection for the Renishaw cube. The light sensor is used to measure the tool length with a broad tolerance, meaning it will only detect if the actual tool is significantly larger or smaller than the desired tool. This way the light sensor acts as a barrier for the cube to prevent collision if the wrong tool is changed into the spindle. The light sensor has a fixed position on the same bracket as the Renishaw cube. An illustration of the light sensor can be seen in Figure 2.7.

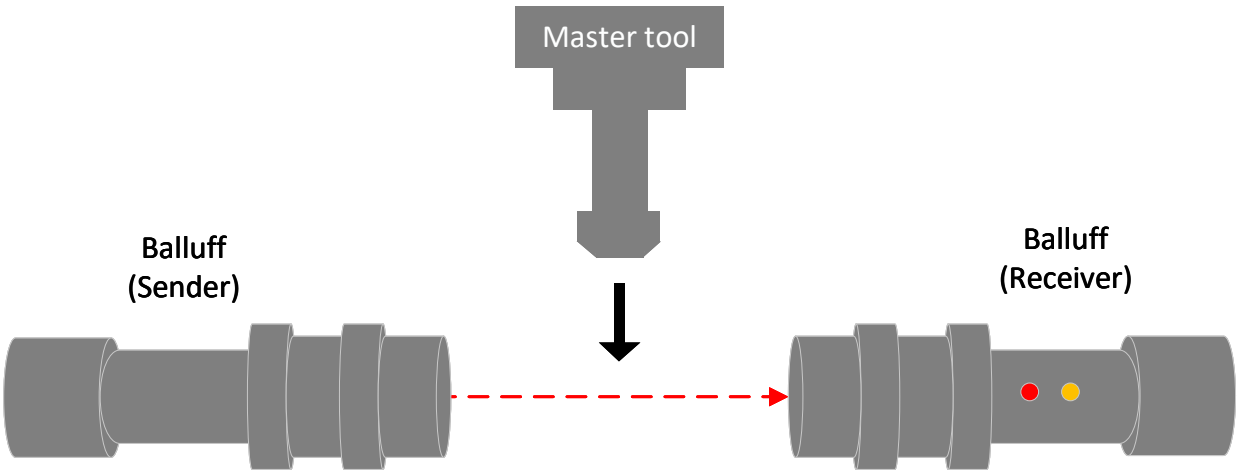


Figure 2.7: Visualization of the light sensor at GAN.

The light sensor is calibrated with the same master tool as the Renishaw cube. The master tool, trigger constants and different logs are explained further in Section 2.3.

2.3 Calibration logs

The calibrations done in the machines at GAN are divided into different log types based on their measuring principle. The calibrations in M5081 are performed in G17, meaning that the XY-direction is chosen as the axis plane. For machine M5081 there are four different measuring principles, resulting in four different log types. When calibrating several probes in different machines with different axis planes, it is difficult to find a uniform descriptive naming for all the trigger points. Therefore a notation consisting of a two-dimensional array has been used in the various log files. For the part probes the two-dimensional array are noted $[u, v]$, where the first index u will be the probe number, and the second index v will be the different trigger points. The first log is referred to as *probelog* and holds the trigger points from the calibration of the part probes. The calibration ring gauge can be seen in Figure 2.8 where it is seen from above in the machine's z-direction. The index u is the probe number for the different part probes the machine magazine holds, and the second index is the trigger points. Index 0 is the diameter of the stylus ball, 1 and 2 are the trigger points in the machine's x-direction, 3 and 4 are the y-direction while 5 is on top of the ring gauge in the machines z-direction.

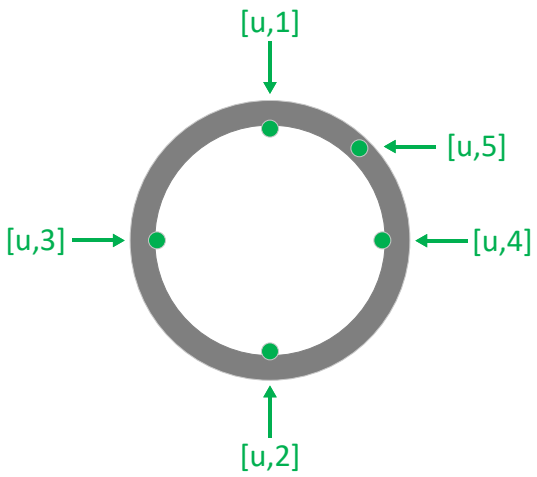


Figure 2.8: Trigger points on the calibration ring gauge.

The objective when calibrating the part probe is to find the position of the probe ball relative to the center of the spindle nose. To calculate a possible misalignment of the stylus a standard Siemens cycle are run in the NCU. This cycle performs two calibration runs at the desired points visualized in Figure 2.8 with spindle orientation at 0° and 180°. The Siemens cycle calculates the misalignment(skew) of the stylus ball in x and y direction and stores it in the same two-dimensional array at index 7 and 8.

The second log is *Kubelog* which contain the trigger points gathered from calibration of the tool probes. The master tools used for calibration are visualized in Figure 2.9 and the trigger points on the cubes can be seen in Figure 2.10. As mentioned previously there are two different cubes in M5081. One Renishaw probe with a quadratic cube of 30mm, and one Blum probe with a quadratic cube of 6mm. The cubes are calibrated with two separate master tools that both have similar design.

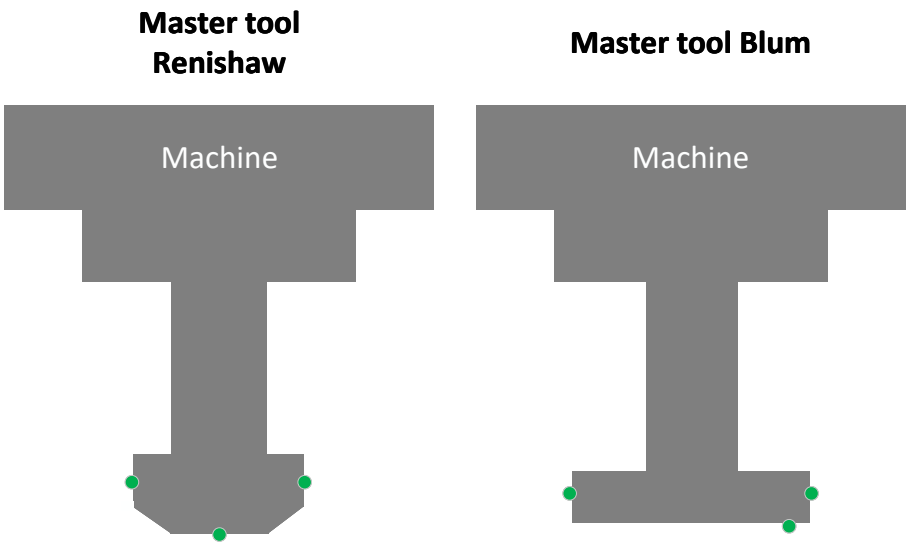


Figure 2.9: Sketch of the master tools for the Renishaw and Blum probes with trigger points marked in green.

As for *probelog*, *kubelog* has a structure with a two-dimensional array $[u, v]$. The cube in Figure 2.10 is seen in the machines x-direction. The trigger point with index $[u, 4]$ are in the machine's z-direction. Because of the positioning, size and mounting of the Renishaw cube it is not possible to trigger on the surface facing down in the machines z-direction, or the surfaces in the machines x-direction. Therefore only three trigger points are applicable for the Renishaw cube. The same trigger points are used on the Blum cube, these trigger points are visualized on the cube in Figure 2.10.

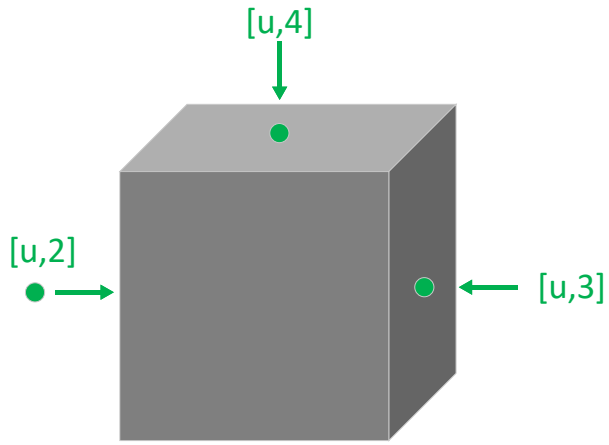


Figure 2.10: Trigger points for the Renishaw and Blum cubes.

The third log called *laserlog* consists of the trigger points from calibration of the Blum laser. The laser beam seen from the machines y+ direction with belonging trigger points can be seen in Figure 2.11. The calibration of the laser is done with the same master tool as the Blum cube. $[u, 0]$ and $[u, 1]$ are the trigger points in the tools length direction which is the machine's z-direction, while $[u, 2]$ and $[u, 3]$ are the trigger points in the tools radius direction which is the machine's x-direction.

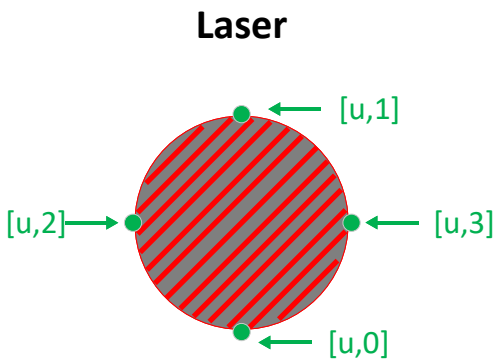


Figure 2.11: Trigger points on the laser beam with the Blum master tool.

The last log is *lyslog* which holds the trigger points from calibration of the light sensor. The light sensors placement makes it difficult to access four trigger points, therefore only $[u, 0]$ in the machine's z-direction and $[u, 3]$ in the machine's y-direction are relevant. As mentioned in Chapter 2.2.4 the light sensor acts as a barrier for the Renishaw cube and has a wide tolerance, therefore it is sufficient enough to gather two trigger constants for the light sensor.

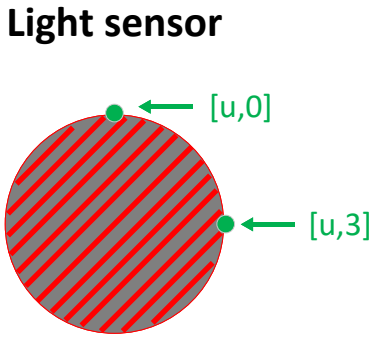


Figure 2.12: Trigger points on the light sensor with the Blum master tool.

With numerous different machines with several measuring principles it is obvious that there will be a large amount of logs with different set of trigger constants and indexing. One important aspect with automatically storing, processing and visualizing this calibration data is to have the raw data in a database. This way the data can be accessed multiple times by several applications and processed again outside the database for example for machine learning purposes. The configuration of the different log types up until the point of this project has been designed in a way to easily being able to manually copy the data in to excel sheets. This configuration was not optimal to develop further, and therefore a new configuration of a more uniform log file considered working for possibly all log types at GAN was developed. The new log type called *CALIB_LOG* is described in Table 2.1.

The new log type has a start and end identifier. In case of a transfer of a log file from the machine to the server fails the log will remain on the machine until the next call for a file transfer. For calibrations done in between these two transfers the new calibration data is appended to the existing log file, resulting in a log file possible containing data from several calibration runs. Therefore it is necessary with a start and end identifier to safely be able to separate the different logs inside the log file. The `MASKINNR` variable is the machine number used to reference the machine at GAN. The `INDEX` row holds the probe number equal to the index u in the two-dimensional array notation used earlier in Chapter 2. The *DATE* row contains the full timestamp which in combination with the machine number is used to uniquely identify each calibration run when referencing them in the database later on in Chapter 3 and 4. $x_0 - x_{10}$ are the trigger constants similar to the index v in the two-dimensional array notation. $nom_0 - nom_{10}$ are the nominal values for the trigger constants $x_0 - x_{10}$. These nominal values can be compared to the trigger constants, which forms the basis for the *diff* value which is the tolerance that the trigger constants can deviate from the nominal values. This *diff* value is a relatively large tolerance in this context, normally $0.15mm$. Which gives:

$$ABS(x_n - nom_n) < 0.15mm$$

The reason for this wide tolerance is natural variations with temperature in the machine metal and mounting brackets. With a significantly smaller value for the *diff* tolerance the calibration runs would be outside the thresholds very often. The benefit with this tolerance is rather in monitoring the tendency and repeatability to identify changes in the machine axes and structure. And to detect metal shavings or debris on the calibration equipment. The *delta_diff* is a smaller tolerance, normally in the area of $0.02mm$ and is calculated the following way:

$$ABS((x_n - x_{n+1}) - (nom_n - nom_{n+1})) < 0.02mm$$

This tolerance can be significantly smaller because it isn't affected by machine variations in the same way. The trigger constant on one side of the cube or laser is subtracted

from the trigger constant on the opposite side and compared to the sum of the respective nominal values. Because of bending of the stylus on the probe when triggering and the laser not triggering exactly at the edge of the laser beam this calculation is not an exact measure of the cube- or laser diameter, but can be thought of that way. However, it should remain fairly constant. Any changes in this calculation implies changes in the laser quality or cube surfaces and can be used to validate the accuracy of the probes and changes in the machine.

The `average` variable is a tolerance that is currently only used in two turning machines with non-rotational tool holders at GAN, but is thought implemented in milling machines in the future. The `average` variable is further described in Chapter 4. The last variables in the table are the alternative temperatures $temp_0 - temp_9$. Since different machines have different number of temperature sensors available there is allocated space in the log for up to ten temperatures. In M5081 there are four temperatures available: Machine temperature, ambient temperature, cooling liquid temperature and spindle oil temperature. The temperatures are written in the log file at the end of each calibration run and are therefore synchronized with the calibration data. The use of these temperatures to potentially improve a Machine Learning (ML) model will be further investigated in Chapter 5. A third party software used to transfer the log files with a method mentioned later in Chapter 3.1 appends a timestamp to the filename before transferring the files. This timestamp includes year, minute and seconds and is used to separate different log files. Because of natural limitations in the machine and file transfer it is not possible to perform two calibrations within a second and therefore the resolution in second is sufficient enough for unique file naming.

Variables in the logs that are not applicable for the current calibration run are set to *NULL*. This way variables that are not applicable for the different visualizations described in Chapter 4 can be filtered out.

Table 2.1: New log format: *CALIB_LOG*.

Log rows	**Description**
start xxxxxlog	Start identifier
maskinnr:	Machine number
index:	Probe number
date:	Timestamp dd-mm-yyyy hh:mm:ss
x_0:	Trigger constant
x_1:	Trigger constant
x_2:	Trigger constant
x_3:	Trigger constant
x_4:	Trigger constant
x_5:	Trigger constant
x_6:	Trigger constant
x_7:	Trigger constant
x_8:	Trigger constant
x_9:	Trigger constant
x_{10}:	Trigger constant
nom_0:	Nominal value trigger constant
nom_1:	Nominal value trigger constant
nom_2:	Nominal value trigger constant
nom_3:	Nominal value trigger constant
nom_4:	Nominal value trigger constant
nom_5:	Nominal value trigger constant
nom_6:	Nominal value trigger constant
nom_7:	Nominal value trigger constant
nom_8:	Nominal value trigger constant
nom_9:	Nominal value trigger constant
nom_10:	Nominal value trigger constant
$diff$:	$diff > ABS(x_n - nom_n)$
$delta\_diff$:	$delta\_diff > ABS((x_n - x_{n+1}) - (nom_n - nom_{n+1}))$
average:	$average > ABS(\frac{x_n - x_{n+1}}{2})$
$temp_0$:	Alternative temperature
$temp_1$:	Alternative temperature
$temp_2$:	Alternative temperature
$temp_3$:	Alternative temperature
$temp_4$:	Alternative temperature
$temp_5$:	Alternative temperature
$temp_6$:	Alternative temperature
$temp_7$:	Alternative temperature
$temp_8$:	Alternative temperature
$temp_9$:	Alternative temperature
end xxxxxlog	End identifier

3 Automatic collection of probe data

This chapter describes the communication protocol used to transfer the calibration logs from the machine to GAN's server, the application built to preprocess the data and the database used.

3.1 File transfer

The communication with the machine is performed using Siemens RPC SINUMERIK. RPC (Remote Procedure Call) is a method used in client-server applications, and the communication interface between the machine and the host computer is based on the Ethernet and TCP/IP protocol. GAN has a Windows server is installed as a virtual machine in their company network and acts as the host for various machines using RPC communication. At GAN the RPC interface is mainly used for two purposes, file transfer and machine monitoring. A software developed by a third party company is used, where file transfer can be initiated both from the machine- and host side. Operators can start transfer of part programs from the host side, and the NC-programmers can request transfer of for example log files from their programs on the machine. In addition RPC communication is used in monitoring applications to read status signals such as hour counters and operating modes. RPC Sinumerik is widely used on both older and newer machines in GAN's factory. Therefore it is a desire to continue to transfer the calibration data in the form of a log file.

3.2 Node-RED application

One initial idea and wish from GAN was to store the raw data from the calibration logs in a database for further visualization, giving the advantage of making the data available for multiple applications to run queries against, and perform calculations any number of times. To process the calibration logs and write the data to a database, an application was developed in Node-RED. Node-RED is a programming tool for event-driven applications. It is ideal for wiring together APIs (application programming interface) and handle data from several hardware devices. Its runtime is built on Node.js and the flow based editor can combine JavaScript functions and many built-in libraries. [5] Node-RED is used

because it is the chosen software for handling data in GKN Aerospace Sweden AB (GAS) monitoring application called CoPilot. GAS are the developers of CoPilot and GAN are currently setting up their own CoPilot environment after a test on a selection of their machines from GAS' environment. The idea is that the functionality developed in this project and tested locally can be applicable for integration at GAN when their CoPilot installation is finished.

The flow developed in Node-RED for processing the calibration logs and writing the data to a database can be seen in the Figures 3.1 and 3.2. The flow consists of several nodes which transfer data between them as message objects, where the data is referred to as payload. The flow of data is from left to right and Figure 3.2 is a continuation from the *SPLIT* node seen in Figure 3.1.

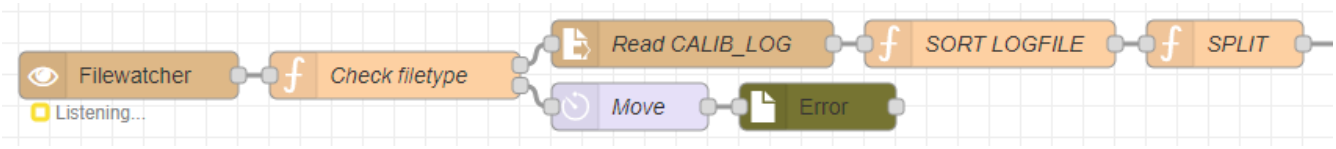


Figure 3.1: Part 1 of *Filewatcher*-flow in Node-RED.

The node called ***Filewatcher*** [6] is a pre-made open source node used to detect new files or changes to files in a selected directory, which in this case is intended set to the directory the calibration logs from the machine are transferred to. The filename and path is then pushed to ***Check filetype*** which is a function node using a regular expression (regex) to validate the filename and type of the received file. Based on the result of this regex test the result is routed to output one or two, where one is ***Read CALIB_LOG*** which reads the file and passes the result in a message as a single utf8 string, and two is a configurable delay followed by a move node that moves the file to the error directory. In the case of a valid file the message from ***Read CALIB_LOG*** is passed through a set of function nodes performing checks and processing the data string. The first node called ***SORT LOGFILE*** separates the different variables in the utf8 string and inserts the data into a two-dimensional array. This array is sent to the function node ***SPLIT*** which checks for start- and end identifiers and calculates the number of logs, and splits the data into separate messages for each log in the file.

The fourth function node ***FORMAT QUERY*** checks the format and length of the array and throws exceptions if the array doesn't meet the requirements for the insert query in the node ***INSERT***. If exceptions are thrown the message is sent through output two, which moves the file to the *Error* folder. Otherwise if the message is the last in a sequence of messages, the log file is moved to the *Arkiv* folder after the last message is sent. The ***INSERT*** node generates a query with the Mustache template format and passes it to the ***toQuery*** node which moves the payload to a query parameter accepted by the ***PostgreSQL*** node. This node creates a connection to and runs the query against a PostgreSQL database, further described in Chapter 3.3.

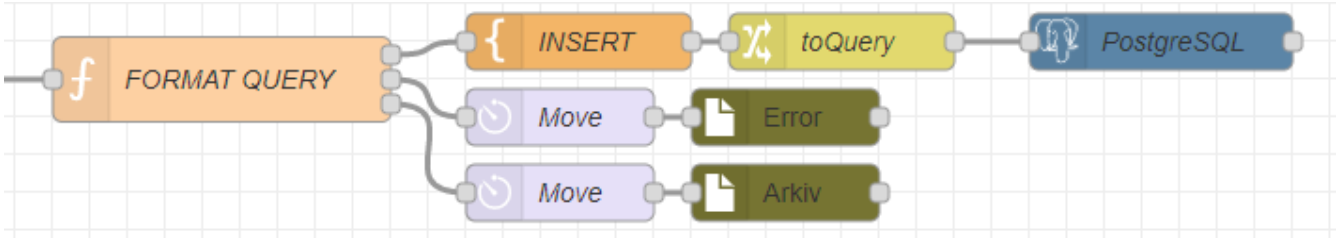


Figure 3.2: Part 2 of *Filewatcher*-flow in Node-RED.

The directory setup with ***Error*** and ***Arkiv*** folders for processed files is a standard at GAN used for other applications in their factory. The source code and JSON model for the described flow in Node-RED are available in Listings B and C.

3.3 PostgreSQL database

PostgreSQL is an open source object-relational database that supports SQL (Structured Query Language) and additional features to store small and large amounts of data. PostgreSQL is a robust and reliable database system that has a reputation for being the most advanced Open Source database in the world. To interact with and generate the database and tables a web-based GUI (Graphical User Interface) tool called pgAdmin was used. pgAdmin is an open source management platform for use with PostgreSQL databases.[7] Both pgAdmin and PostgreSQL are used in CoPilot and was therefore chosen for this project. pgAdmin has its own built-in ERD (Entity relationship diagram) tool that was used to create the tables for the different log types. The ERD tool generates queries for the tables that can be reused to expand and generate tables for other logs in the future. The queries along with other relevant code developed will be delivered with this report with the purpose to be uploaded in GitLab, which is GAN's chosen software for source code and version control.

An overview of the tables created in pgAdmin can be seen in Figure 3.3.

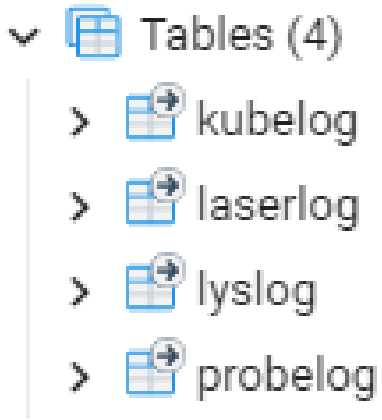


Figure 3.3: The tables created in the PostgreSQL database.

The database consists of the four tables *kubelog*, *probelog*, *laserlog* and *lyslog* previously mentioned in Chapter 2.3. The database containing these tables are created with an extension called TimescaleDB. This extension can be created on PostgreSQL databases to improve functionality when working with time-series data. The extension does not influence the PostgreSQL syntax and makes it possible to access a certain time period without searching through the whole table, making the queries quicker to run. To achieve this the tables are converted into hypertables by adding a unique index on two columns. For the tables in Figure 3.3 the columns creating this index are the *date* : and *maskinnr* : columns from Table 2.1. The query for creating the tables in the postgreSQL database can be seen in Listing D.

4 Visualization of calibration data

This chapter describes the monitoring and analytics application Grafana, and the dashboards created for monitoring the calibration data at GAN.

Grafana is an open source web application that allows the developer to build graphs, charts and dashboards, and to query data to be monitored and analyzed. [8] To access data Grafana needs to connect to a data source, which for this project is the PostgreSQL database described in Chapter 3.3. Grafana does not validate the safety of the queries being run and therefore the user provided for connection with Grafana should only be granted *SELECT* permissions. This way deleting data and dropping tables will be prevented from the Graphical User Interface (GUI). After setting up the data source the dashboards seen in Figure 4.1 was created.

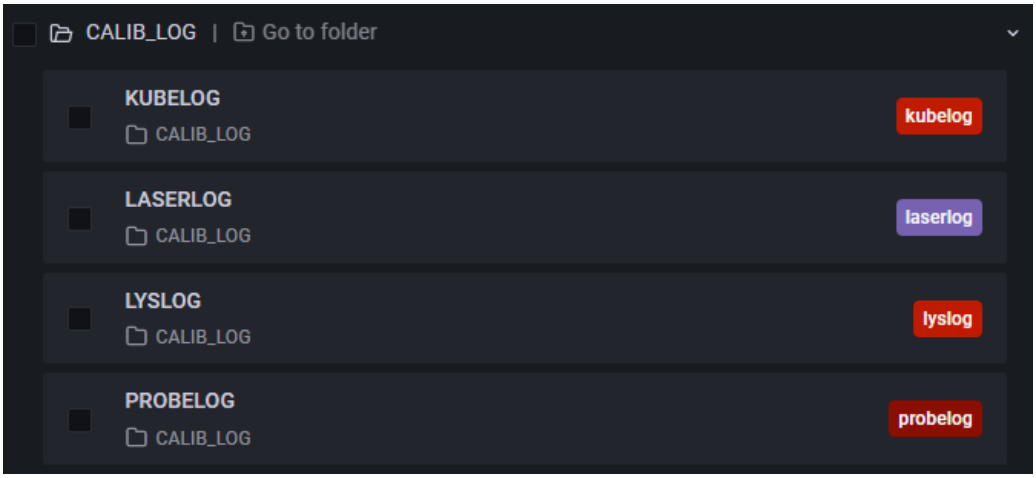


Figure 4.1: The dashboards for the different log types created in Grafana.

The dashboards in Grafana are created and customized with an administrator account and can be implemented into GAN's Grafana environment in their company network, to make them available from the GAN employees viewer accounts. Several ways to set up dashboards for the different log types where considered, however a single dashboard for each log type was found convenient.

Figure 4.2 shows an example of the environmental variables and rows in the *kubelog* dashboard. The environmental variables are configured as drop-down menus to be able to view machines and different probes explicitly. The environmental variables contain queries run against the database to generate lists of distinct machines and probes, which can further be used in other queries to filter data. The panels in the dashboards are graphs and tables which are sorted into rows which can be minimized to increase maneuverability within the dashboard.

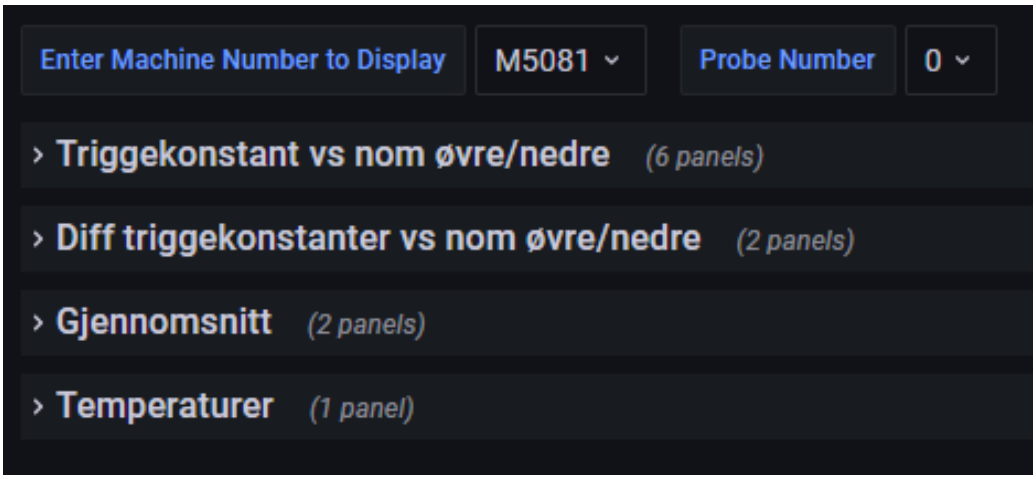


Figure 4.2: Dashboard setup with drop-down menus for environmental variables and rows for panels.

Each dashboard has a global time range, configurable update frequency and links to other dashboards, as seen in Figure 4.3. The time range is used as a search criteria and views the actual time range if using the zoom functions in the different panels.

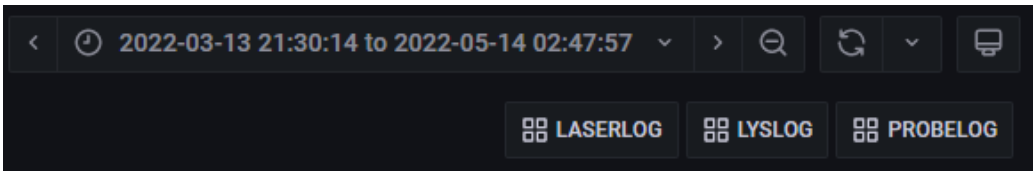


Figure 4.3: Dashboard setup for global time range and links to other dashboards.

The JSON model of the dashboard is the data structure that defines each dashboard. This model can be exported from each dashboard and imported into GAN's Grafana environment for easy implementation.

Figure 4.4 shows an example of a panel from *kubelog* where the trigger constant x_2 is plotted versus the nominal upper and lower limits given by the $diff$ tolerance from Table 2.1. Each of the dashboards for the different log types have panels for each of the actual trigger constants versus their respective nominal values. These panels are Grafana's *Time series* graph which gives a good representation of the trigger constants. The panels have the possibility to change the time range by marking a field in the graph, which changes the global time range for the dashboard and all belonging panels. The legend in each panel is at the bottom left corner and shows the variables available. The variables can be deselected to show only distinct variables or all together. When hovering over the panel the actual values for each variable are available in the tooltip. This is an important feature because the old visualization done in MS Excel does not include the unique timestamps in the tooltip, which makes it more difficult to analyze the data. The query for this panel can be seen in Figure 4.5.

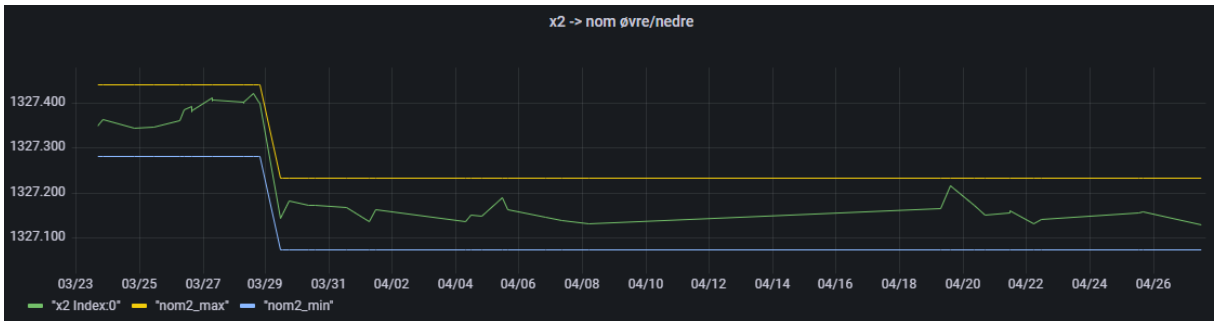


Figure 4.4: x_2 trigger constant plotted versus the nominal limits.

```
SELECT
  time AS "time",
  x2 AS """x2 Index:$Index""",
  nom2 + diff AS """nom2_max""",
  nom2 - diff AS """nom2_min"""
FROM public.kubelog
WHERE
  $__timeFilter("time") AND
  machineno = '$MachineNo_new' AND
  index = '$Index'
```

Figure 4.5: Query for the panel showing x_2 vs the nominal limits.

To increase the readability of the time series each graph has a table linked to it that follows the same time range. The table for the x_2 trigger constant from *kubelog* can be seen in Figure 4.6.

x2 -> nom øvre/nedre

Time	"x2 Index:0"	"nom2_max"	"nom2_min"
2022-03-23 16:02:11	1327.3491	1327.4414	1327.2814
2022-03-23 19:56:55	1327.3645	1327.4414	1327.2814
2022-03-24 20:14:12	1327.3456	1327.4414	1327.2814
2022-03-25 10:52:57	1327.3462	1327.4414	1327.2814

Figure 4.6: Table for the x_2 trigger constant and nominal limits.

Another panel that is common for several of the dashboards are the graph that shows the difference between trigger constants with related upper and lower limits. Figure 4.7 shows the difference $x_2 - x_3$ versus the nominal limits which is calculated based on the $delta\_diff$ tolerance from Table 2.1. As mentioned earlier the tolerance for these limits can be a lot smaller because this check is less affected by machine variations than the $diff$ check. Optimally this difference between the respective trigger constants is supposed to be zero. If the laser were to trigger exactly at the edge of the laser beam, or the probes triggered immediately when touching the master tool these differences would be exact measurements of the laser and cube diameter. Because of bending in the stylus of the probes and accuracy of the laser this is not the case.

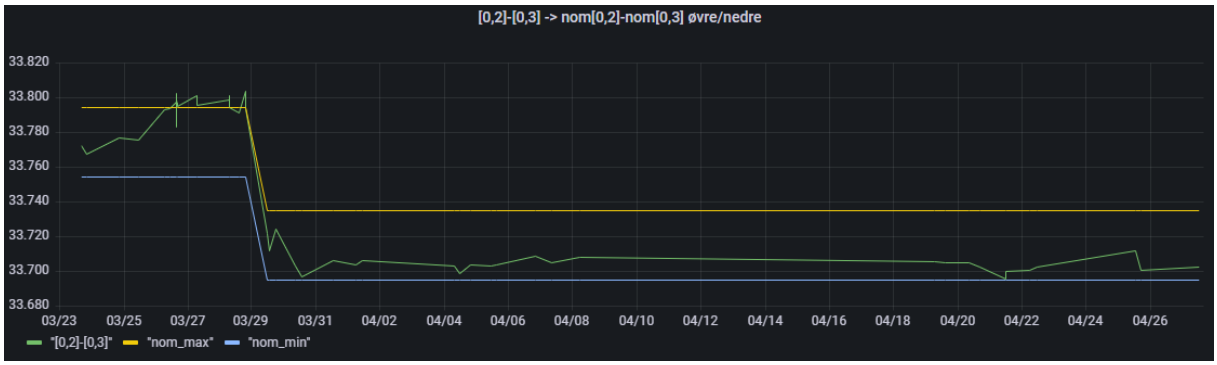


Figure 4.7: The difference $x_2 - x_3$ versus nominal limits.

In addition to increasing utilization and understanding of the calibration data for the machine operators and engineers, the graphs gives a good overview for maintenance personnel. The nominal values that forms the basis for the limits seen in the previous figures are set by the maintenance personnel at Kongsberg Terotech AS (KTT). Previously they have not had the historical data available when changing a probe and setting new nominal values, which has often resulted in a unnecessary shifting of the nominal limits. By having the historical data in the previous figures available while setting new nominal values after maintenance or replacements of the probes, the maintenance personnel can with higher certainty declare the machines safe for production after a machine stop or repair.

As mentioned in Chapter 2.3 the part probes are calibrated with a standard Siemens cycle. Since M5081 is a milling machine with a rotating spindle it is possible to calibrate the probe with rotation, resulting in the opportunity for the Siemens cycle to calculate the skew of the stylus ball in the X and Y direction. The results of this calculation are written in variable x_7 and x_8 in *probelog*. By running the calibration cycle at the calibration ring gauge with 0° and 180° spindle rotation, the x_1,x_2 and x_3,x_4 will be exact opposites while the displacement of the stylus ball is calculated in variable x_7 and x_8. In M5081 this displacement is used in position control to correct the placement of the stylus ball when using the part probe. An example of the panel showing displacement of the stylus ball in X and Y direction can be seen in Figures 4.8 and 4.9.

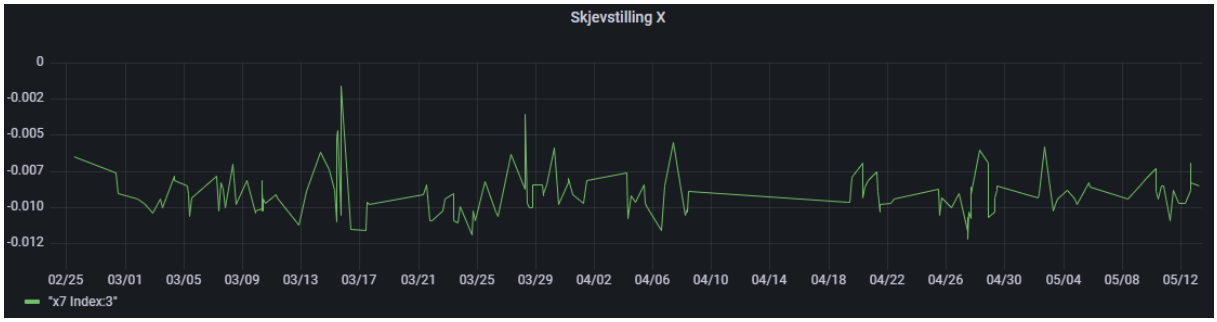


Figure 4.8: Calculated skew on part probe in the x-direction.

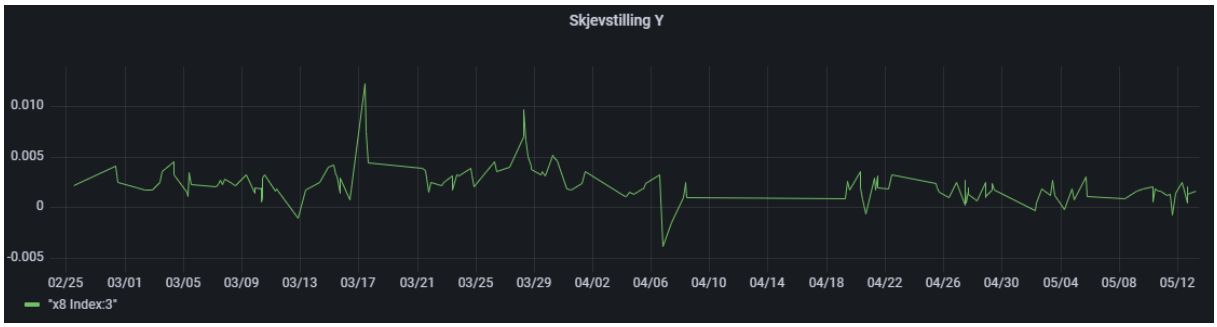


Figure 4.9: Calculated skew on part probe in the y-direction.

An average value like the skew calculated gives negligible value for other measuring principles in M5081, but in for example a turning machine with a non-rotating tool holder it can be of great use. In a turning machine with no possibility to rotate the part probe during calibration, there is no compensation in position based on misalignment of the stylus ball. Therefore GAN has recently introduced average calculation on two turning machines that uses the *average* variable from Table 2.1 as tolerance. The *average* tolerance check is not active in M5081 as of today, but is thought used to monitor the skew in the future.

A temperature panel like the one seen in Figure 4.10 is included in all the Grafana dashboards. The temperatures shown are the relevant temperatures from each log given by the variables $temp0 - temp9$ from Table 2.1. Previously the various temperatures in M5081 was not included in the calibration logs, and therefore there are no history data on the temperature available for this machine. Machine-, ambient-, coolant- and spindle oil temperatures are now included in the calibration logs and will be available for monitoring in the future. As mentioned in Chapter 2.3 an interesting aspect can be if there is any correlation between temperatures in the machines and the trigger constants, and if they could potentially be used to improve a ML model.

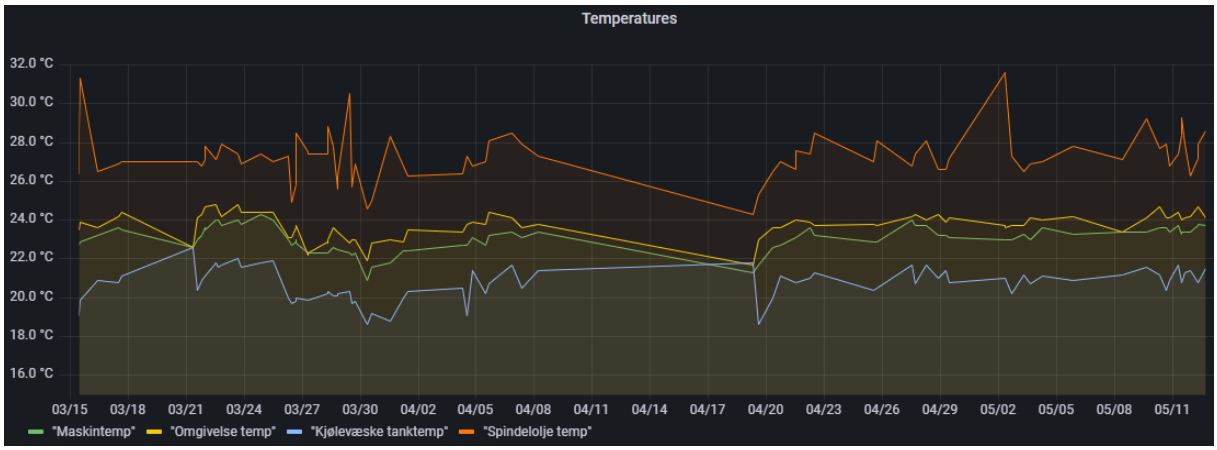


Figure 4.10: The machine-, ambient-, coolant- and spindle oil temperatures for M5081.

5 Machine learning in CNC machining

This chapter will give a short overview on machine learning and describe previous work done by GAN and others on calibration and vibration data, tool wear and machine diagnostics in CNC machines. Utilization of the calibration data with a nonlinear auto regressive neural network will be tested, and Principal Component Analysis (PCA) will be used to investigate possible correlations between the temperature measurements and the trigger constants available in the machine.

5.1 Machine learning overview

Industry 4.0 often abbreviated I4.0 is a term for the fourth industrial revolution that involves gathering of large amounts of data through Internet of Things (IOT), as well as analytics and machine learning techniques used to increase and improve manufacturing. By introducing new technologies for data gathering in the recent years and retrofitting older machines with new sensors, GAN gathers a lot of data from the machines in their factory that with a better understanding probably could be utilized for monitoring and predictive maintenance. According to Z. M. Cinar Et al. [9] ML is seen as a part of artificial intelligence and is defined as any program or algorithm that can learn with small or no additional support, and has already existed for decades in various applications. In a predictive maintenance perspective ML can be used to predict failure based on models created of historical data.

Traditionally ML can be divided into the three categories: *Supervised learning*, *Unsupervised learning* and *Reinforcement learning*. The selection of which category to use is based on the type of data available and how the problem is defined.

Supervised learning is a type of methods where the algorithm uses data sets that are labeled. Labeled means that the datasets contains the desired solutions. *Supervised learning* uses training data sets including the inputs and correct outputs to train a model. During training, the weights are adjusted until the error is sufficiently minimized through a validation process. Two important classes for supervised algorithms are classification and regression. Classification algorithms are used with discrete data where the goal is to find the decision boundary and divide the data set into two or more classes. This way it can be possible to map the input variable with the output variable, to be able to predict the class

of the output. For regression problems, Linear- and logistical regression algorithms are often used to understand the relationship between dependent and independent variables. Regression algorithms works with continuous data and finds the line that best fits the data set to be able to predict the output. [10] [11]

Unsupervised learning methods uses algorithms that works with unlabeled data sets. These algorithms can cluster and analyze data to find data groupings and hidden patterns. Examples of clustering algorithms used in unsupervised learning models can be K-means, which is an exclusive clustering method and Hierarchical Cluster Analysis (HCA). One-class SVM and Isolation Forest are examples of anomaly detection algorithms. These can be used to search through large data sets to find abnormal data points. Another important category is dimension reduction and visualization, where PCA is most common. PCA can be used to reduce redundancies and compress data sets as well as for preparation to visualize the data sets. [12] [11]

Reinforcement learning is a very different approach, where the learning system, often called *agent* interprets its environment, acts and then updates its strategy based on a trial and error manner. [11]

5.2 Previous work on machine learning for CNC machines

In 2021 some research on error detection and probe calibration data in CNC machines was done for GAN by K.A.S Guldbjørnsen. [13] The aim with the work in his report was to test a statistical method on data sets provided from GKN and relate these results to condition-based maintenance and fault recognition. GAN provided calibration data from a vertical lathe machine called Carnaghi as well as belonging temperature data, and GAS provided spindle run-out data and bearing data from a milling machine. The report concluded that it was possible to see a trend in the spindle run-out and bearing data, because they had a known period where the spindle had a break down. The analysis on the calibration data set showed possibilities to identify deviations from normal calibration. The analysis performed was done by scaling the data with a pre processing method called Z-score, and using PCA and Mahalanobis distance. The work done did not produce any solution for automatic gathering of the calibration data, nor implementation of the statistical method at GAN.

Z. Tang et al. [14] performed a study on calibration data from touch probes on a CNC boring-milling machine with a Siemens 840D SL control system, similar to the control system on M5081 at GAN. They used probe data from calibrating a touch probe on a calibration ring gauge for duo active error detection, and wrote a NC-program on the CNC machine to write log files containing the probe data to an external device host over LAN. The log files were then processed on the external device/host computer and the data was stored in a Manufacturing Execution System (MES) database. This is similar to

the solution implemented at GAN where the log file is written on the machine and finally transferred to a host computer, and shows that the analysis and gathering of probe data are relevant.

GAN's machine park contains machines from several decades. From machines coming into production in recent years to machines from before year 2000. Naturally the older machines don't have the same technology when it comes to sensor and communication technology as the newer machines. Therefore retrofitting of older machines to adapt to I4.0 is an important aspect. D.F.Hesser et al. [15] equipped an old CNC milling machine with a Bosch XDK programmable sensor device with a built in accelerometer to measure acceleration data. The aim with their project was to train an Artificial Neural Network (ANN) to see if they could classify the tool wear and remaining useful life (RUL) of the machine tool. In the report it is concluded that retrofitting older machines with new sensors is useful to utilize older machines for I4.0, and calculation of tool wear/RUL can be achieved with an ANN based on supervised regression, with a sufficient amount of useful historical data available.

Regarding acceleration data GAS have had monitoring of spindles on numerous machines for several years. GAN has vibration data available for monitoring from one of their machines, and are looking at alternatives with additional sensors on other machines to utilize spindle vibration data. Therefore utilization of vibration data is an interesting aspect, and vibration data will be available at GAN in the future.

5.3 Machine health monitoring

With the rise of IOT in the industry and easier access to machine specific data, examples of predictive maintenance and condition monitoring can be found. Y.M Al-Naggar et al. [16] performed a study where four CNC milling machines was equipped with accelerometers, where the spindles were the source of vibration. The spindle vibration was sampled in real time and stored in a database and then processed with Python. A fast fourier transform (FFT) technique was applied to be able to plot the vibration in the frequency domain. By using the root mean square (RMS) velocity V_{RMS}, the spindle vibration was used to determine if the machine was good, satisfactory, unsatisfactory or unacceptable according to ISO 10816. The decision on the machine condition by comparing the machine vibration with the vibration severity per ISO 10816 was done manually by inspecting the GUI, and no ML was utilized. Hence, it was concluded that the vibration monitored from the sensor mounted on the spindle in the CNC machine could be used to evaluate the machine condition.

The idea with using machine signals to determine machine health and condition monitoring is an interesting aspect, and to utilize the trigger constants in a predictive maintenance

manner and alert before something is about to happen, would be a great achievement for GAN.

5.4 Exploring Machine learning models at GAN

The calibration data for the different measurement principles in machine M5081 are data with a belonging nominal value. The nominal values are targets for the measured inputs and therefore a supervised learning approach could be convenient. The trigger constants gathered are accurate data in the sense that they are measured with precise measurement systems down to 1 micrometer accuracy. As mentioned earlier $diff$ and $delta\_diff$ are strict tolerances used to validate the calibrations in the machine, which means that a machine learning model would have to have a high degree of accuracy to have any value in predicting these calibrations. A common error in M5081 are machine stops as a consequence of the trigger constants from calibrating the cubes exceeding the $delta\_diff$ tolerance. The cube calibrations are performed with a rotating master tool, which results in wear on the cubes as more calibrations runs are performed. A goal was therefore to investigate if the data from *kubelog* can be used to predict the next machine stop, caused by a trigger constant exceeding the $delta\_diff$ tolerance. The target in such prediction will be the variable $x_2 - x_3$ as shown in Figure 4.7, which is the difference between the two trigger constants x_2 and x_3. This gives a direct correlation between the inputs to the model and the target.

5.4.1 Multistep Prediction with a Nonlinear Autoregressive Neural Network

To configure, train and test a Nonlinear Autoregressive (NAR) neural network MATLAB was used. NAR is a neural network that can be trained on a time series and predict future values for that same time series. The model is trained in open loop form with real values as feedback, then in closed loop mode the network can predict future values with internal feedback. The available relevant trigger constants for the Renishaw cube from the *kubelog* table was extracted from the PostgreSQL database, resulting in trigger constants from 1159 calibration runs available for training and testing the neural network. To be able to use all the trigger constants as one time series and deal with shifting in the variables due to occasional changes in the nominal values, each variable were subtracted from its nominal value. Following this outliers were removed with the *rmoutliers* function in MATLAB, with median absolute deviations (MAD) which is the default method. The MATLAB code for creating the target time series and removing the outliers can be seen in Appendix E.

To build and test the NAR neural network with configurable setup, a function called *NARkubelog* was created in MATLAB. The code can be seen in Listing F. This function loads the time series created and extracts a configurable number of trigger constants used to validate the predictions. The training of the neural network was performed in open loop mode, meaning that real inputs where fed to the network during training. To predict the future outputs with internal feedback the network was simulated in closed loop mode, where the final input states and layer states from the open loop training was used as initial values for the closed loop simulation. The closed loop architecture of the neural network can be seen in Figure 5.1.

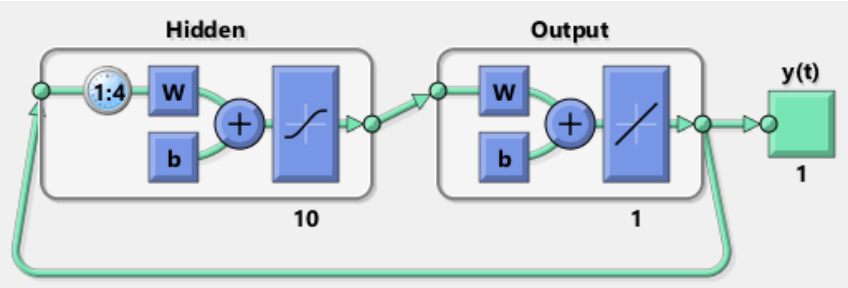


Figure 5.1: Closed loop architecture of the NAR Neural Network with four feedback delays and ten neurons in the hidden layer.

The *trainlm* and *trainbr* network functions in MATLAB were chosen as two alternatives to train the model. *trainlm* is generally a fast backpropagation algorithm using Levenberg-Marquardt optimization to update the weights and biases, used for supervised problems. The training is stopped when the network performance fails to improve or stays constant for a maximum number of epochs. The other training function *trainbr* where *br* stands for Bayesian regularization backpropagation also uses Levenberg-Marquardt optimization to update the weights and biases, but the training continues until the algorithm finds the optimal combination of squared errors and weights. [17] [18] When testing and training the models with the *trainlm* and *trainbr* training functions, several configurations with different number of hidden neurons and delays were tested. A configuration with 10 neurons in the hidden layer, four feedback delays and *trainbr* as the training function gave the results seen in the process overview in Figure 5.2. Before training the model the data was divided into blocks where 80% of the inputs was used for training and 20% was used to test the network, which is default for the *trainbr* function. Mean squared error (MSE) was the quality criterion and the training stopped on $\boldsymbol{Mu}$ after 215 iterations with a performance of $1.81e^{-05}\boldsymbol{MSE}$. $\boldsymbol{Mu}$ is a control parameter that is incremented and decremented with an increase factor based on the performance value. When the performance fails to improve for a number of iterations making the $\boldsymbol{Mu}$ parameter exceed a maximum limit, the training is stopped. $\boldsymbol{Mu}$, hidden neurons, feedback delays and the other parameters in the process overview are hyperparameters determined before the training starts and works as external parameters controlling the learning process, but they are not part of the final model. For simplicity other hyperparameters where set according

to MATLAB's default values.

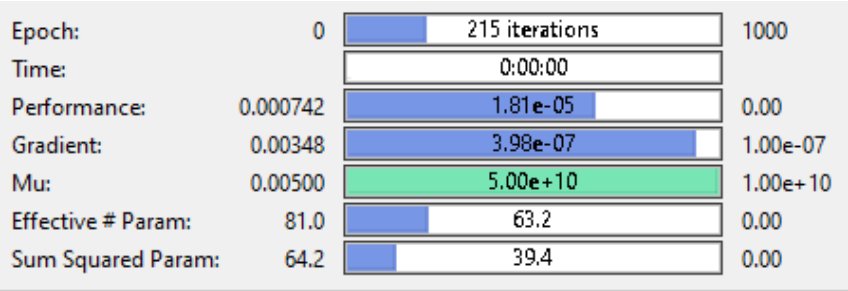


Figure 5.2: Process overview after training the neural network with Bayesian regularization backpropagation algorithm.

A plot used to validate the network performance is the autocorrelation plot seen in Figure 5.3. For an ideal prediction model the autocorrelation function should have one non-zero value that appear at zero lag. The autocorrelation plot shows the correlation between a value at one time step and a number of time steps back, and can be used to find patterns or describe randomness in the data set. The plot was used to find an adequate number of delays for the model. The correlation values falls approximately in under the confidence limit for all the different lags which can indicate an appropriate model.

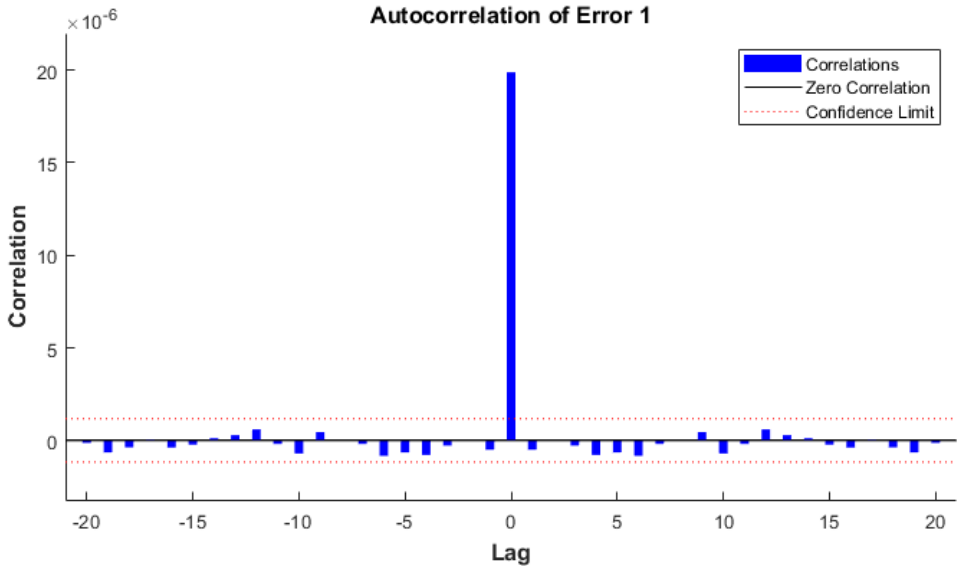


Figure 5.3: Plot of the autocorrelation function after training the neural network with Bayesian regularization backpropagation algorithm.

Another plot used to validate the network performance is the regression plot seen in Figure 5.4. This plot displays the outputs with respect to targets for the training set, test set and finally both combined. Optimally the data should be on the 45 degree dotted line which would indicate a perfect fit. Even though the data set has some noise after the first round of removing outliers, $R = 0.88659$ overall is a fairly accurate result.

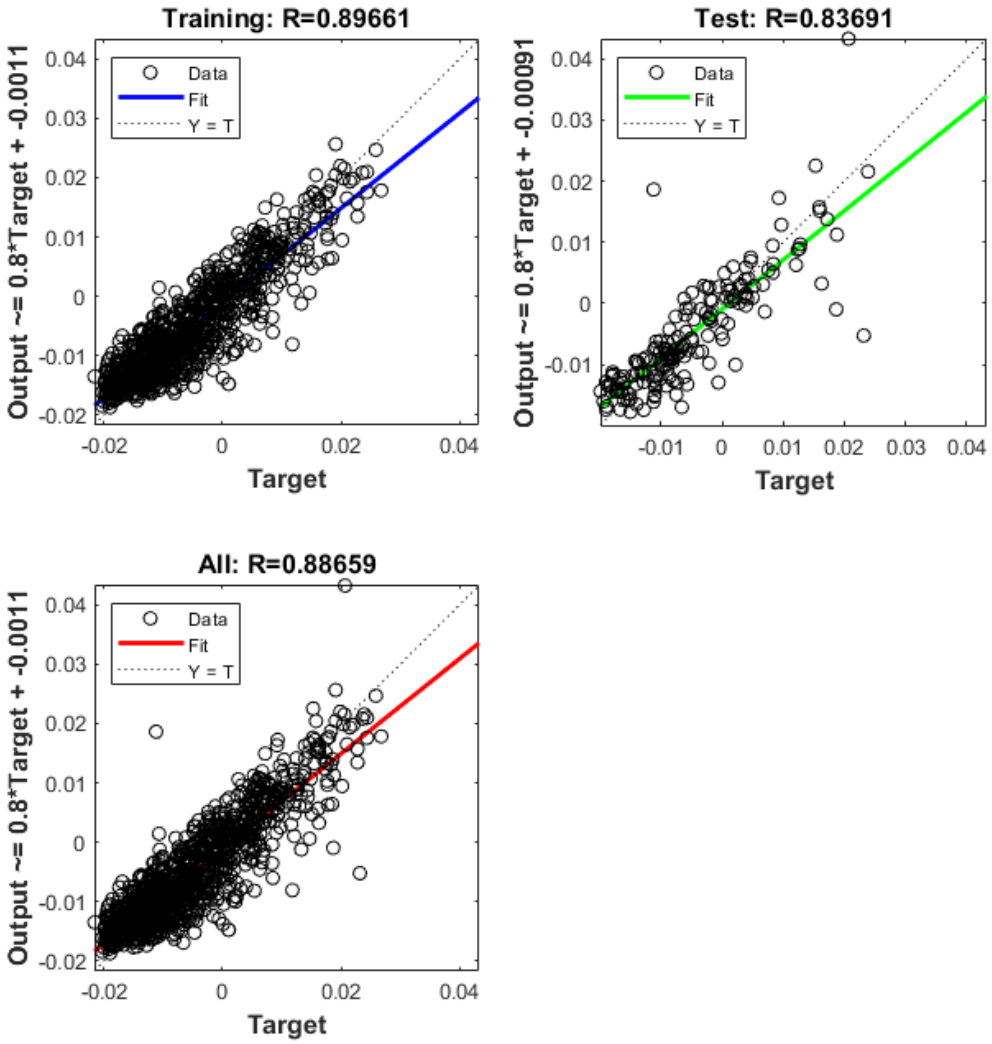


Figure 5.4: Regression plots of training set and test set after training the model with Bayesian regularization backpropagation algorithm.

To get an indication on the outliers and further investigate the network performance the error histogram seen in Figure 5.5 can be convenient. This histogram shows that the majority of the instances are gathered around zero error between -0.007 and 0.009. And the plot indicates an apparent normal distribution where the errors are centered around the mean, where the noise are white noise from the measurements. The approximately 20 instances outside the mentioned limits can be potential outliers seen on the regression plot.

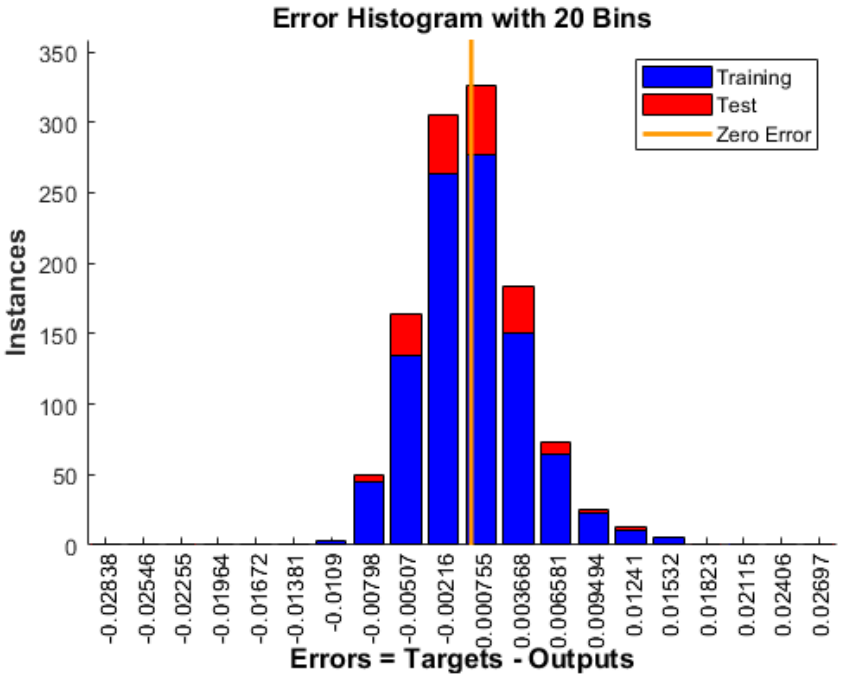


Figure 5.5: Error histogram between targets and predicted values after training the nonlinear autoregressive neural network.

Finally after validating the network performance and finding the network giving the best performance, the network was simulated in closed loop mode to predict the future outputs. The idea with predicting the future outputs is to find a time estimate until the next machine stop. If it is possible to predict a number of calibrations ahead accurately, and the number of calibrations per day can be assumed to be somewhat constant, the time until the next machine stop caused by calibration can be indicated. Figure 5.6 shows the results after the multistep prediction in closed loop mode. Before training the network the last 6 instances was extracted from the time series to be used to validate the prediction.

The top subplot shows the real outputs 6 steps ahead indicated by the blue line, and the predicted outputs from the model indicated by the green line. The bottom subplot shows the difference between the real output and the predicted output, to better visualize the magnitude of the error. The plot indicates a maximum absolute error at calibration 4 of

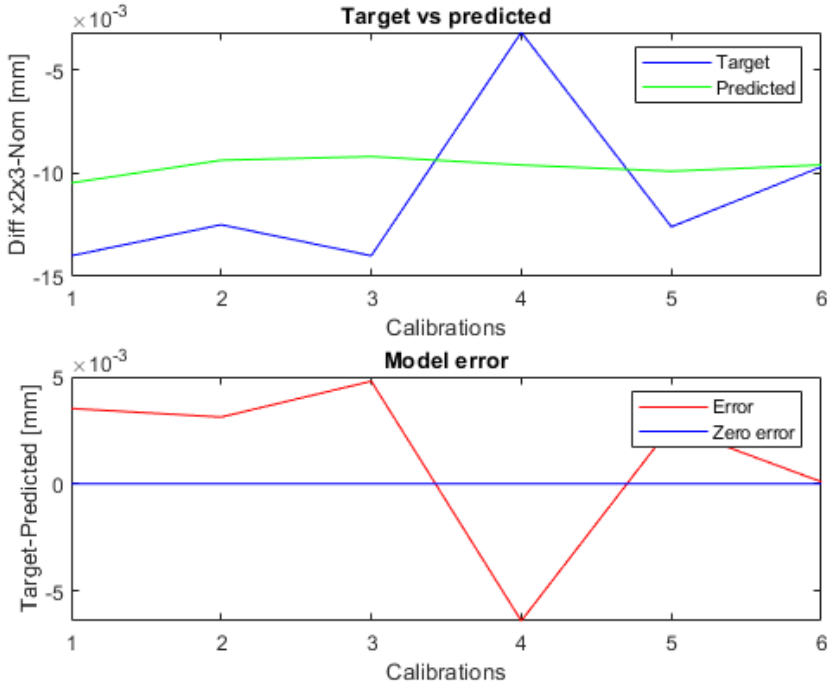


Figure 5.6: Plot of the predicted output versus the real targets extracted from the time series, and the model error.

approximately $6 \times 10^{-3} mm$, which is relatively close to the target. However, the predicted outputs does not seem to follow the same trend as the real outputs. As mentioned earlier the $delta\_diff$ variable is a small tolerance of about $0.02mm$, which means that with a model with a possible prediction error of $6 \times 10^{-3} mm$, it will be difficult to accurately predict the time until a calibration is out of tolerance with this model. The MATLAB code for creating and training the NAR neural network and plotting the predictions can be seen in Appendix F.

5.4.2 PCA

As mentioned previously it can be interesting to investigate the correlation between the temperature measurements available and the trigger constants. To look into possible correlations PCA was used. PCA is a dimensionality reduction method used to reduce the number of variables in a large data set, but maintain as much information as possible. For the interested reader more information about PCA are available at [19] [20]. Since the four available temperatures in M5081 was included in the new calibration logs late in the project period, only 63 samples was available at the time of testing. A dataset containing the variables x_2, x_3, x_4 and $x_2 - x_3$ along with the four temperatures was created and loaded in MATLAB. Further the dataset was normalized and the *pca* function in MATLAB was used to get the principal component coefficients, scores and variances. The *pca* function centers the data and uses the singular value decomposition algorithm. The plot of the eight variable vectors and how they contribute to principal component 1 and 2 combined with the scores can be seen in Figure 5.7.

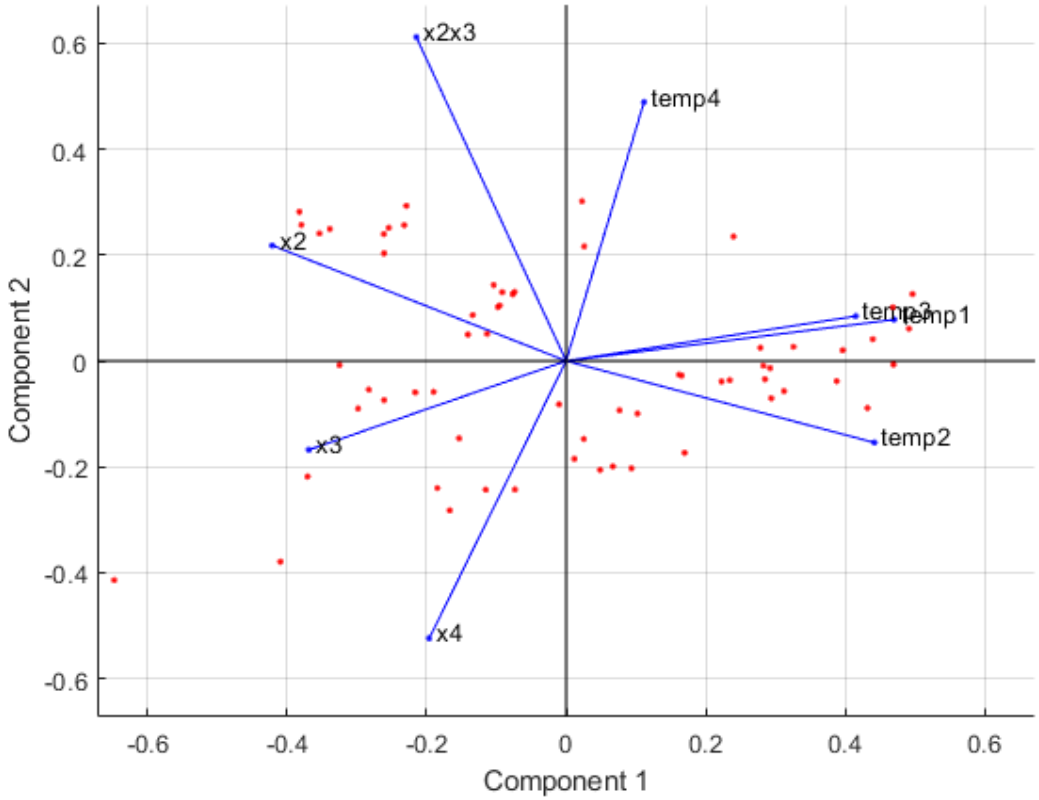


Figure 5.7: The eight loadings and scores along principal component 1 and 2.

By looking at the first component along the horizontal axis it is clear that it has positive

coefficients for the temperature variables, and negative coefficients for the trigger constants. Meanwhile the second component on the vertical axis have positive coefficients for the variables x_2, x_2x_3, $temp1$, $temp3$ and $temp4$, and negative coefficients for the variables x_3, x_4 and $temp2$. The three loadings x_2, x_3 and x_4 are roughly pointing in opposite directions of the temperatures $temp1$, $temp2$ and $temp3$, respectively. $temp1$ is the machine temperature measured inside the machine room, and $temp3$ is the temperature of the cooling liquid being used when machining. These two temperatures are pointing in approximately the opposite direction of the trigger constant $x3$. A possible explanation for this negative correlation can be the fact that the trigger constant $x3$ is the side of the cube facing inwards in the machine room, and is most exposed to cooling liquid and variations in machine room temperature. Similarly $x2$ is the side of the cube facing towards the tool changer door which is frequently opened, and is exposed to variations in ambient temperature. Along the vertical axis it can be seen that the loadings x_2x_3 and $temp4$, where $temp4$ is the temperature of the hydraulic spindle oil, are pointing in the same direction with relatively high magnitude. This gives an indication that the variables x_2x_3 and $temp4$ are positively correlated. Figure 5.8 shows the percentage of variance explained by the principal components. Adding the variances for component 1 and 2 gives a total variance explained by the two components of approximately 64%, which is relatively low. To describe the data set further principal component 3 are taken into consideration, giving a total variance explained of approximately 77%. The plot containing scores and loadings for principal component 1 and 3 can be seen in Figure 5.9.

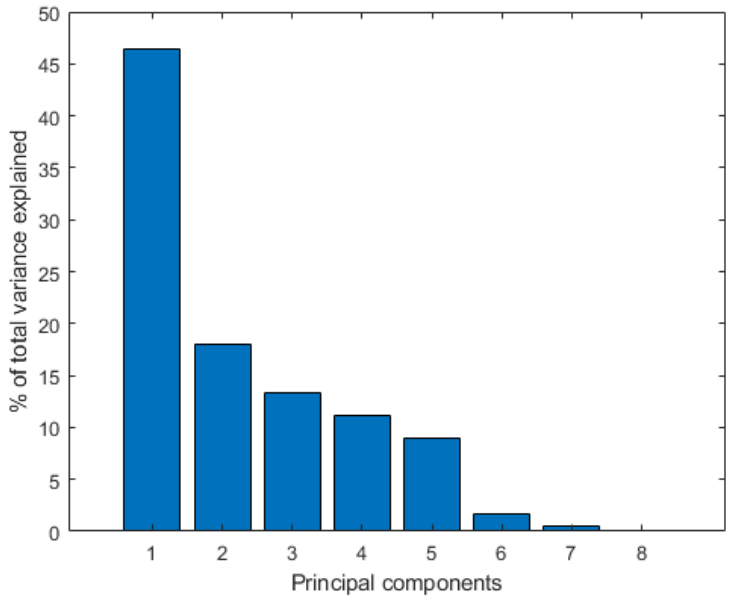


Figure 5.8: Percentage of total variance explained by each of the principal components.

The plot shows that principle component 3 has a slight positive coefficient for x_2x_3 and

negative coefficients for x_4 and $temp4$. Hence, they belong to the same cluster. This strengthens the assumption that the variable x_2x_3 and the spindle oil temperature $temp4$ are correlated. The x_4 variable which is the trigger constant on top of the cube in the machine's z-direction is also located in the same cluster. The fact that variations in spindle temperature affects the trigger constants in the spindle's travelling direction seems reasonable. The negative correlations found can indicate a relationship between the trigger constants and the temperatures measured in and outside of the machine. Taking this into consideration it is reasonable to believe that temperature measurements can be valuable inputs to a machine learning model to improve predictions in the future.

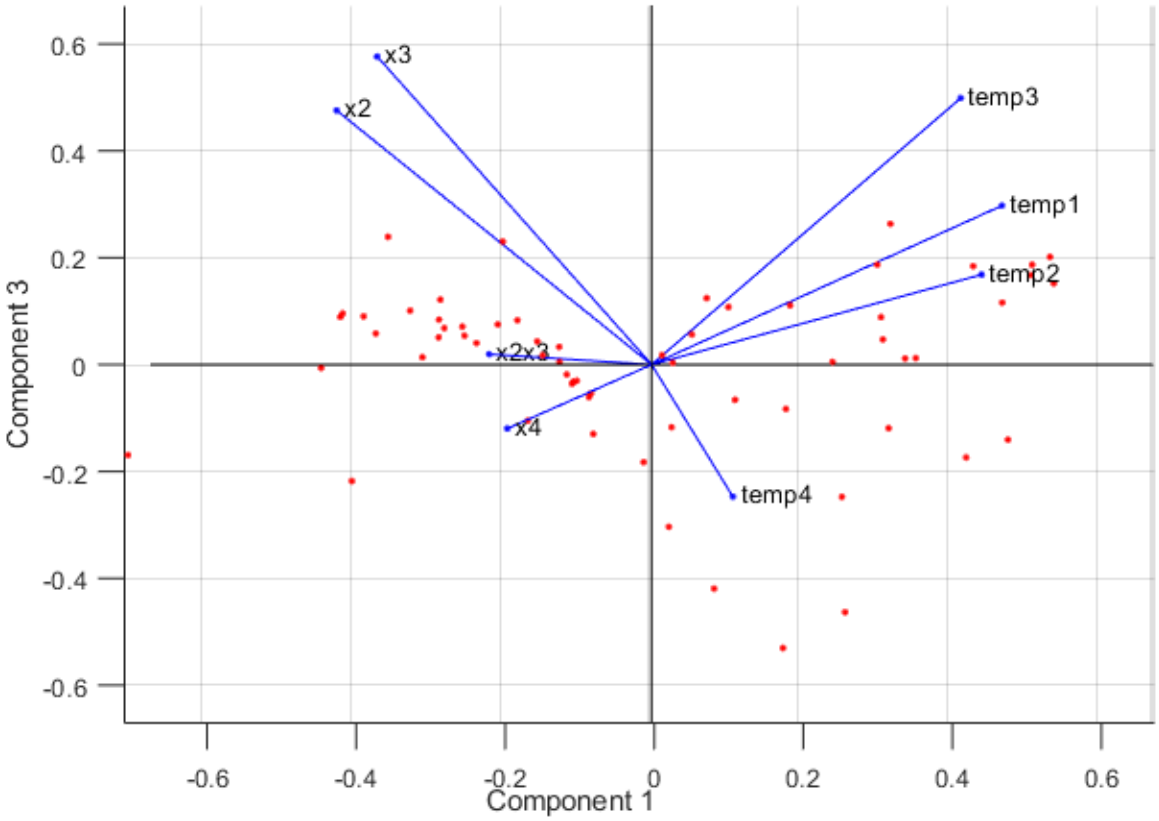


Figure 5.9: The eight loadings and scores along Principal component 1 and 3.

5.4.3 Spindle vibration data

Regarding potential spindle vibration data from M5081 and other machines at GAN the findings from Chapter 5.4.2 are interesting. The calibrations with the master tools are performed with a rotating tool, with a spindle speed in the area of $1000 - 1500 rpm$. Increasing vibration in the spindle could indicate increasing temperature because of the spindle working on a high speed, or possible wear on bearings. Using spindle vibration data as an additional input could be a good idea considering the correlations between the trigger constants, and the measured temperatures from the PCA results.

6 Discussion

In this chapter the automatic collection of probe data and visualization in Grafana are discussed. As well as suggestions for future work, and results from creating an optional machine learning model and the PCA performed.

6.1 Machine theory and automatic collection and visualization of probe calibration data

One of the main objectives of the project was to give a general theory description regarding the measuring principles and machine structure of CNC machines, and to define a system to automatically transfer log files from a CNC machine. Since GAN have several different CNC machines with different structures it was agreed upon M5081 being the concept machine for this project. M5081 is a large machine covering several measuring principles and has a control system and setup similar to many other machines, which would make the system defined suitable for a number of machines. To give an overview of the different trigger constants and their relations to the measuring probes in the machine, the square bracket notation $[u, v]$ was used. Underway in the project after discussing visualization layout and naming of variables in Grafana with personnel at GAN, this notation was found insufficient for further use. Therefore a new template for the calibration log with a more uniform notation was developed. The new calibration template was implemented in GAN's logging routine on M5081, tested and found working sufficient. There are several possibilities in continuously monitoring machine variables over Ethernet in CNC machines, and there are several reasons that the solution with transferring a log file was chosen. Firstly M5081 was delivered with a control system without support for an Open Platform Communications (OPC) Server, which is most common for continuous monitoring of machine variables at GAN. Secondly the third party software used to transfer log files on M5081 are used of other machines at GAN, making future implementation of the new calibration log template and changes in the NC-programmers logging routines easier. Other arguments for choosing the transfer of a log file instead of continuous monitoring of variables is the local storage of the log file in case of network issues, and the additional functionality demanded regarding handshake and synchronisation with continuous monitoring of the event driven trigger constants. After testing CoPilot for some time GAN are currently setting up their own docker environment for a full scale installation

of CoPilot. Since this installation was postponed until the end of the project period, the defined system in this project had to be tested locally on a windows computer, resulting in some simplifications. To be able to reuse parts of the functionality developed, the tools used in CoPilot including Node-RED, Grafana and pgAdmin implementing the postgreSQL database was chosen. The script for creating the tables with hyperindexes in the postgreSQL database, as well as the JSON model for the Grafana dashboards created can be exported and used directly for implementation in CoPilot. For the functionality developed in Node-RED, the functions regarding decoding of the log files can be reused. But some modifications must be expected on the file watching part, the queries for the postgreSQL database, and in the Node-RED application code generally to prepare for environmental variables. The goal with this project was to suggest and define a system for automatic collection and visualization of calibration data. The functionality was created to test and visualize the flow of data from calibration logs to visualization locally. When implementing the system in GAN's CoPilot environment additional functionality regarding logging and error handling should be considered.

6.2 Previous work and machine learning at GAN

In the literature survey performed mostly work regarding data collection from CNC machines was found. Most of which referred to work regarding acceleration data and RUL of machine tools. Some relevant work was found regarding classification of machine conditions based on spindle vibration data, but without machine learning methods implemented. The article by Z. Tang et al. [14] showed the process of monitoring probe data from a Siemens 840D SL control system similar to M5081, and wrote a log file directly on a shared network drive on a host computer. This solution could be used at GKN to avoid using a third party software for transferring the log files with RPC Sinumerik, which is a rather obsolete solution. When evaluating different machine learning models suitable for GAN, supervised learning was emphasized because of the calibration data being a labeled data set. The possibilities for GAN in the short term was to investigate possibilities to utilize the calibration data available. The trigger constant x_2x_3 from *kubelog* have been used exclusively when training the machine learning model in the project to delimit the testing. There may be other trigger constants and useful data in the other log types for use with the machine learning discussed in this report.

6.3 NAR neural network and PCA

Until the final part of the project period when the logging of temperature values started, the only variables available for use with a machine learning model was the trigger constants. Several ways to utilize the trigger constants for predictive maintenance were considered. Firstly a multiclass classification approach was considered and thought to be most intuitive for the engineers and operators at GAN, possibly giving several states on the cube condition. A model based on regression could also be used to determine different states. Because of the strict tolerances for the trigger constants the multiclass classification and regression approaches was found inconvenient. Since the x_2x_3 variable being outside of tolerance often is mainly caused by wear on the cube, an idea was to train a neural network to use for multistep prediction to be able to predict the time until the next calibration is outside of tolerance, which could indicate remaining life of the cube. Because the testing of the NAR neural network was exploratory to investigate possibilities with the calibration data available a static machine learning model was created, meaning it was trained offline and used to predict the future time steps. For further testing and possibly better results a dynamic model could be used, and more time could be put into tuning hyperparameters and gathering more data. To investigate the possible correlation between the trigger constants and the temperatures available, a brief PCA was performed. The PCA showed correlations between several trigger constants and the temperatures measured, which can be argued to have a logical explanation. The PCA was performed with few temperature samples, and a new PCA in the future when more temperature samples are available is recommended to investigate the correlations found further. In hindsight knowing the possible correlations with the temperatures in the machine a Nonlinear Autoregressive with External Input (NARX) model would possibly give better multistep prediction of the x_2x_3 variable. For the future when more calibration data including temperature samples, and possibly spindle vibration measurements are available, a NARX model with x_4, temperatures and spindle vibration as inputs is recommended for testing, to see if a better prediction of the trigger constant x_2x_3 can be done.

For creating and simulating the static neural network MATLAB was used. Node-RED has the possibility of running a python node which can execute python scripts. In hindsight for a future implementation of a machine learning model at GAN a python implementation in Node-RED is likely to be used. Therefore it would have been convenient if the development and testing of the neural network were done with python as the programming tool in this project.

7 Conclusion

The first objectives of this project was to give a problem description including a system description, present general theory about measuring principles in CNC machines, perform a literature survey on predictive maintenance and machine learning in CNC machines and to define a system for automatically transfer, preprocess, store and visualize the calibration data. A problem description including a system description of the current solution was presented, and general theory on the 5-axis vertical machining center (M5081) and its measuring principles was given. In collaboration with GAN M5081 was chosen as the concept machine because of its similar configuration with many other machines at GAN. Functionality developed will therefore be applicable on several machines. A new template for calibration logs was developed in collaboration with GAN and tested physically on M5081. The defined system was tested with the same tools used in GAN's monitoring application CoPilot, and simulated locally on real log files generated by M5081 following the new template. The log files generated were copied into a folder on a computer to simulate the log file transfer, and visualized in the developed Grafana dashboards. For a future implementation in GAN's CoPilot environment, some modifications must be made in Node-RED regarding the filewatcher, postgreSQL queries and environment variables. And additional functionality on logging should be developed for troubleshooting. But the automatic collection and visualization of probe data was tested successfully at a local installation with the tools desired by GAN.

The literature survey performed addressed relevant work done regarding logging of probe data on a CNC machine with similar control system as M5081, and spindle vibration data seems to be widely used to calculate RUL of machine tools and tool wear. The literature survey produced no results regarding the use of probe calibration data to determine machine or equipment health. When looking into opportunities for utilizing probe calibration data with a machine learning model, it became clear that the use of multistep prediction to indicate time until the next machine stop caused by calibration of the cubes could be useful. The nonlinear autoregressive neural network developed did not produce sufficient enough predictions to be able to foresee the next machine stop caused by a calibration being outside its tolerance. The PCA was performed on a limited amount of temperature samples but showed possible relationships between several trigger constants and temperatures. With this in mind and the possible implementation of a vibration sensor in the spindle on M5081 in the future, it is concluded that a NARX neural network

with temperatures and vibration data as inputs should be tested to possibly provide more accurate predictions.

Bibliography

[1] 'Gkn aerospace in europe.' (2022), [Online]. Available: https://www.gknaerospace.com/en/about-gkn-aerospace/locations/gkn-aerospace-in-europe/.

[2] 'Cnc machine maintenance: Avoid downtime & keep productivity high.' (2022), [Online]. Available: https://www.machinemetrics.com/blog/cnc-machine-maintenance.

[3] S. Köwerich, 'Picture of cad model provided by stefan köwerich.,' *GKN Aerospace Norway AS*, 2022.

[4] 'Balluff - b0s00wf.' (2022), [Online]. Available: https://www.balluff.com/de-de/products/BOS00WF#cadCaeBtn.

[5] 'Node-red.' (2022), [Online]. Available: https://nodered.org/.

[6] 'Node-red.' (2022), [Online]. Available: https://flows.nodered.org/node/node-red-contrib-watchdirectory.

[7] 'About pgadmin.' (2022), [Online]. Available: https://www.pgadmin.org/.

[8] 'About grafana.' (2022), [Online]. Available: https://grafana.com/docs/grafana/latest/introduction/.

[9] Z. M. Cinar, A. A. Nuhu, Q. Zeeshan, O. Korhan, M. Asmael and B. Safaei, 'Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0.,' *MDPI*, p. 42, 2020.

[10] 'Supervised learning.' (2022), [Online]. Available: https://www.ibm.com/cloud/learn/supervised-learning.

[11] *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly Media, Incorporated, 2015.

[12] 'Unsupervised learning.' (2022), [Online]. Available: https://www.ibm.com/cloud/learn/unsupervised-learning.

[13] K. A. S. Guldbjørnsen, 'Manufacturing analysis and data acquisition in advanced machining.,' p. 67, 2021.

[14] Z. Tang, X. Jiang, W. Zi, X. Shen and D. Zhang, 'Automatic data collecting and application of the touch probing system on the cnc machine tool.,' *Hindawi*, p. 19, 2021.

[15] D. F. Hesser and B. Markert, 'Tool wear monitoring of a retrofitted cnc milling machine using artificial neural networks.,' *Elsevier*, p. 4, 2018.

[16] Y. M. Al-Naggar, N. jamil, M. F. Hassan and A. R. Yusoff, 'Condition monitoring based on iot for predictive maintenance of cnc machines.,' *Elsevier*, p. 5, 2021.

[17] 'Trainlm - levenberg-marquardt backpropagation.' (2022), [Online]. Available: `https://se.mathworks.com/help/deeplearning/ref/trainlm.html`.

[18] 'Trainbr - bayesian regularization backpropagation.' (2022), [Online]. Available: `https://se.mathworks.com/help/deeplearning/ref/trainbr.html;jsessionid=fc4e850f308ab22c32f2110197a9`.

[19] 'A step-by-step explanation of principal component analysis (pca).' (2022), [Online]. Available: `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`.

[20] K. H. Esbensen, *Multivariate data analysis - in practice: An introduction to multivariate data analysis and experimental design.* Oslo: Camo, 2001.

Appendix A

The task description for the project.

This appendix contains the task description for the Master's thesis *Automatic data collection, visualization and predictive maintenance during probe calibration in CNC machines.*

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Automatic collection and visualization of probe data from calibration in CNC-machines and outlooks for machine learning implementation.

USN supervisor: Håkon Viumdal.

External partner: GKN Aerospace Norway AS, Stefan Köwerich and Kongsberg Terotech, Lars Ingebrigtsen.

Task background:
All CNC-machines at GKN Aerospace Norway AS are equipped with different types of probes for measuring tools and parts. These probes need to be calibrated several times a week to ensure accurate measurements. The measurements (trigger constants) from these calibration runs are stored in a logfile on the machine. These measurements can give an excellent status on the machine health if visualized and monitored correctly. This visualization is today a manual job where the calibration results are copied into an excel sheet. This process with collecting, processing, and visualizing the data needs to be automated. The probe calibration data is already stored in a file on the machine and needs to be transferred over ethernet, processed, and written to a database. The data stored in the database needs to be visualized. Further GKN would like to know how the data from the calibrations can be utilized in the future for predictive maintenance.

Task description:

- Problem description including a system description
- General theory about measuring principles in CNC-machines and the machine structure.
- Perform literature survey on predictive maintenance and machine learning, emphasizing automated health monitoring of CNC-machines or similar applications.
- Defining a system to automatically transfer log files from machine to server over Ethernet, and to create and store the calibration data in a database.
- Process and visualize the calibration data, in a way that can later be implemented into GKNs existing monitoring system and utilized for the process operators and the engineers that are monitoring the machines.
- Preprocess the data in order to be applied on machine learning algorithms.
- Evaluate various machine learning methods that can be applied for visualization and monitoring of the machine health.
- Optional: Make a related machine learning model for one of the CNC-machines, according to the evaluations done in the former point.

Student category: Reserved for IIA Industry Master student at Terotech.

The task is suitable for online students (not present at the campus): Reserved

Practical arrangements:
Access to data sets at GKN Aerospace Norway will be provided.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): Skien, 27.01.2022 

Student (write clearly in all capitalized letters): WALTER JOHANSSON

Student (date and signature): Kongsberg 31.01.2022 

Appendix B

Source code for JavaScript function nodes in Node-RED.

This appendix contains the source code for the JavaScript function nodes in the Node-RED flow developed in this project.

```
// Init
bool = false;
field = "";
logtype = "";
filename = "";
moveFile = 'Arkiv';

// Functions
function useRegex(input) {
    let regex = /CALIB_LOG+_\d\d\d\d\d\d\d\d\d\d\d\d\.MPF/i;
    return regex.test(input);
}

// Error handling
bool = useRegex(msg.file);
node.status(bool);
if ( bool == false) { node.status("Feil filtype"); moveFile = 'Error'; }
if ( msg.type == "none") { return null; }


// Switch payload
if (moveFile === "Arkiv") {
   return [ msg, null ];
} else if (moveFile === "Error") {
   return [ null, msg ];
} else {
    node.error("Feilet");
    return null;
}
```

Listing 1: *Check filetype* function node in Node-RED.

```
// Init
twoDarray = new Array(2);
oneDarray = new Array(1);
let message = msg.payload;
timestamp = '';
machinenr = '';
lines = message.split(/\r\n|\r|\n/).length;

// Create array
length = message.split(' ').length;
array = [];
datoarray = [];
machinearray = [];
array = message.split("\n",length);
node.status(array.length);
// Create two-dimensional array
for (let i = 0; i <= array.length - 1; i++) {
    if(array[i].includes('START') || array[i].includes('END')) {
        delimiter = ' ';
    }
    else {
        delimiter = ':';
    }
        // Format 'DATE'
    if(array[i].includes('DATE:')){
        array[i] = array[i].trim();
        datoarray[0] = array[i].substring(0,5);
        datoarray[0] = datoarray[0].replace(':','');
        datoarray[1] = array[i].substr(5);
        datoarray[1] = datoarray[1].replace('.','-').replace('.','-');
        oneDarray = datoarray;
        datoarray = [];
    } else if(array[i].includes('MASKINNR')) {
        machinearray[0] = array[i].substring(0,9);
        machinearray[0] = machinearray[0].replace(':','');
```

```
        machinearray[1] = 'M' + array[i].substr(9);
        // Remove whitespace
        machinearray[1] = machinearray[1].replace(/\s/g, "")
        oneDarray = machinearray;
        machinearray = [];
    }
    else {
        oneDarray = array[i].split(delimiter);
        // Remove whitespace
        oneDarray = oneDarray.map(element => {return element.trim();});
    }

    if (oneDarray[0] != undefined) {
        twoDarray.push(oneDarray);
    }
}

msg.payload = twoDarray;
return msg;
```

Listing 2: *SORT LOGFILE* function node in Node-RED.

```
// Init
start = false;
end = false;
newpayload = [];
numMessages = 0;
msg.lastmessage = false;

i = 2;
while(i < msg.payload.length ){
    if(msg.payload[i][0] == 'START'){
        start = true;
        end = false;
    }
    if(msg.payload[i][0] == 'END'){
        end = true;
        start = false;
    }
    if(start == true && end == false){
        newpayload.push(msg.payload[i])
    }
    if(start == false && end == true){
        newpayload.push(msg.payload[i])
        node.send({payload: newpayload, file: msg.file , filename: msg.filename ,
    lastMessage: msg.lastmessage});
        numMessages = numMessages + 1;
        start = false;
        end = false;
        newpayload = [];
    }
    i++;
}
msg.lastmessage = true;
node.send({payload: newpayload, file: msg.file , filename: msg.filename , lastMessage: msg.
    lastmessage});
```

Listing 3: *SPLIT* function node in Node-RED.

```
// Init
msg.format = 'ok';

if (msg.lastMessage == false)
{
        // Check format
    if(msg.payload.length != 40){
        node.error('Feil format p   fil');
```

```
        msg.format = 'feil';
        return [ null, msg, null ];
    }

    // Switch payload
    if (msg.format === "ok") {
       return [ msg, null, null ];
    }
}

// Last message, move file
if (msg.lastMessage === true) {
   return [ null, null, msg ];
} else {
    node.error("Error");
    return null;
}
```

Listing 4: *FORMAT QUERY* function node in Node-RED.

```
INSERT INTO public.{{payload.0.1}} ("time", machineno, index, x0, x1, x2, x3, x4, x5, x6, x7
    , x8, x9, x10, nom0, nom1, nom2, nom3, nom4, nom5, nom6, nom7, nom8, nom9, nom10,
diff, delta_diff, average, temp0, temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8,
    temp9)
values ('{{payload.3.1}}','{{payload.1.1}}',{{payload.2.1}},{{payload.4.1}},{{payload
    .5.1}},{{payload.6.1}},{{payload.7.1}},
{{payload.8.1}},{{payload.9.1}},{{payload.10.1}},{{payload.11.1}},{{payload.12.1}},{{payload
    .13.1}},{{payload.14.1}},{{payload.15.1}},{{payload.16.1}},{{payload.17.1}},{{payload
    .18.1}},
{{payload.19.1}},{{payload.20.1}},{{payload.21.1}},{{payload.22.1}},{{payload.23.1}},{{
    payload.24.1}},{{payload.25.1}},{{payload.26.1}},{{payload.27.1}},{{payload.28.1}},{{
    payload.29.1}},
{{payload.30.1}},{{payload.31.1}},{{payload.32.1}},{{payload.33.1}},{{payload.34.1}},{{
    payload.35.1}},{{payload.36.1}},{{payload.37.1}},{{payload.38.1}});
```

Listing 5: Insert query in Mustache template format from the *INSERT* node in Node-RED.

Appendix C

JSON model for Node-RED flow.

This appendix contains the JSON model for the Node-RED flow.

Listing 1: JSON model for the flow in Node-Red.

```
[{"id":"2c38f497.2c2a0c","type":"tab","label":"Filewatcher/
    Filelogger","disabled":false,"info":""},{"id":"2c3cccef.41af
    24","type":"file in","z":"2c38f497.2c2a0c","name":"Read
    CALIB_LOG","filename":"","format":"utf8","chunk":false,"
    sendError":false,"encoding":"none","x":510,"y":100,"wires":[
    ["f894ee9a.b57db"]]},{"id":"42b8d509.3bf0ac","type":"
    function","z":"2c38f497.2c2a0c","name":"Check filetype","
    func":"// Init\nbool = false;\nfield = \"\";\nlogtype =
    \"\";\nfilename = \"\";\nmoveFile = 'Arkiv';\n\n// Functions
    \nfunction useRegex(input) {\n    let regex = /CALIB_LOG+_\\
    d\\d\\d\\d\\d\\d\\d\\d\\d\\d\\d\\d\\d\\.MPF/i;\n    return
    regex.test(input);\n}\n\n// Error handling\nbool = useRegex(
    msg.file);\nnode.status(bool);\nif ( bool == false) { node.
    status(\"Feil filtype\"); moveFile = 'Error'; }\nif ( msg.
    type == \"none\") { return null; }\n\n\n// Switch payload\
    nif (moveFile === \"Arkiv\") {\n   return [ msg, null ];\n}
    else if (moveFile === \"Error\") {\n   return [ null, msg ]
    ;\n} else {\n    node.error(\"Feilet\");\n    return null;\n
    }\n\n","outputs":2,"noerr":0,"initialize":"","finalize":"","
    libs":[],"x":320,"y":120,"wires":[["2c3cccef.41af24"],["71a7
    7f6b.76667"]]},{"id":"f894ee9a.b57db","type":"function","z":
    "2c38f497.2c2a0c","name":"SORT LOGFILE","func":"// Init\
    ntwoDarray = new Array(2);\noneDarray = new Array(1);\nlet
    message = msg.payload;\ntimestamp = '';\nmachinenr = '';\
    nlines = message.split(/\\r\\n|\\r|\\n/).length;\n\n//
    Create array\nlength = message.split(' ').length;\narray = [
    ];\ndatoarray = [];\nmachinearray = [];\narray = message.
    split(\"\\n\",length);\nnode.status(array.length);\n//
    Create two-dimensional array\nfor (let i = 0; i <= array.
    length - 1; i++) {\n    if(array[i].includes('START') ||
    array[i].includes('END')) {\n        delimiter = ' ';\n    }
    \n    else {\n        delimiter = ':';\n    }\n        //
    Format 'DATE'\n    if(array[i].includes('DATE:')){\n
    array[i] = array[i].trim();\n        datoarray[0] = array[i]
    .substring(0,5);\n        datoarray[0] = datoarray[0].
    replace(':','');\n        datoarray[1] = array[i].substr(5)
    ;\n        datoarray[1] = datoarray[1].replace('.','-').
    replace('.','-');\n        oneDarray = datoarray;\n
    datoarray = [];\n    } else if(array[i].includes('MASKINNR')
    ) {\n        machinearray[0] = array[i].substring(0,9);\n
           machinearray[0] = machinearray[0].replace(':','');\n
           machinearray[1] = 'M' + array[i].substr(9);\n
```

```
// Remove whitespace\n        machinearray[1] = machinearray
[1].replace(/\\s/g, \"\")\n        oneDarray = machinearray
;\n        machinearray = [];\n    }\n    else {\n
oneDarray = array[i].split(delimiter);\n        // Remove
whitespace\n        oneDarray = oneDarray.map(element => {
return element.trim();});\n    }\n    \n    if (oneDarray[0]
 != undefined) {\n        twoDarray.push(oneDarray);\n    }\
n}\n\nmsg.payload = twoDarray;\nreturn msg; \n\n\n\n\n\n\n\n\n
\n\n\n\n\n\n\n\n\n\n\n\n\n","outputs":1,"noerr":0,"initialize"
:"","finalize":"","libs":[],"x":700,"y":100,"wires":[["34650
371.1a2dbc"]]},{"id":"232c67e2.5a2918","type":"postgresql","
z":"2c38f497.2c2a0c","name":"PostgreSQL","query":"","
postgreSQLConfig":"1f5943dc.3dc0ec","split":false,"
rowsPerMsg":1,"outputs":1,"x":870,"y":300,"wires":[[]]},{"id
":"8f86443b.03a218","type":"change","z":"2c38f497.2c2a0c","
name":"toQuery","rules":[{"t":"move","p":"payload","pt":"msg
","to":"query","tot":"msg"}],"action":"","property":"","from
":"","to":"","reg":false,"x":700,"y":300,"wires":[["232c67e2
.5a2918"]]},{"id":"3c3d4d31.7dc1e2","type":"fs-ops-move","z"
:"2c38f497.2c2a0c","name":"Error","sourcePath":"D:\\Test\\
Filewatcher\\","sourcePathType":"str","sourceFilename":"file
","sourceFilenameType":"msg","destPath":"D:\\Test\\
Filewatcher\\Error","destPathType":"str","destFilename":"
file","destFilenameType":"msg","link":false,"x":670,"y":340,
"wires":[[]]},{"id":"34650371.1a2dbc","type":"function","z":
"2c38f497.2c2a0c","name":"SPLIT","func":"// Init\nstart =
false;\nend = false;\nnewpayload = [];\nnumMessages = 0;\
nmsg.lastmessage = false;\n\ni = 2;\nwhile(i < msg.payload.
length ){\n    if(msg.payload[i][0] == 'START'){\n
start = true;\n        end = false;\n    }\n    if(msg.
payload[i][0] == 'END'){\n        end = true;\n        start
 = false;\n    }\n    if(start == true && end == false){\n
        newpayload.push(msg.payload[i])\n    }\n    if(start
== false && end == true){\n        newpayload.push(msg.
payload[i])\n        node.send({payload: newpayload, file:
msg.file , filename: msg.filename , lastMessage: msg.
lastmessage});\n        numMessages = numMessages + 1;\n
        start = false;\n        end = false;\n
newpayload = [];\n    }\n    i++;\n}\nmsg.lastmessage = true
;\nnode.send({payload: newpayload, file: msg.file , filename
: msg.filename , lastMessage: msg.lastmessage});\n\n","
outputs":1,"noerr":0,"initialize":"","finalize":"","libs":[]
,"x":850,"y":100,"wires":[["7673b0c1.b4bd3"]]},{"id":"7673b0
```

```
c1.b4bd3","type":"function","z":"2c38f497.2c2a0c","name":"
FORMAT QUERY","func":"// Init\nmsg.format = 'ok';\n\nif (msg
.lastMessage == false)\n{\n        // Check format\n    if(
msg.payload.length != 40){\n        node.error('Feil format
 p   fil');\n        msg.format = 'feil';\n        return [
null, msg, null ];\n    }\n    \n    // Switch payload\n
if (msg.format === \"ok\") {\n       return [ msg, null,
null ];\n    }\n}\n\n// Last message, move file\nif (msg.
lastMessage === true) {\n   return [ null, null, msg ];\n}
else {\n    node.error(\"Error\");\n    return null;\n}\n","
outputs":3,"noerr":0,"initialize":"","finalize":"","libs":[]
,"x":370,"y":320,"wires":[["ea913484.bb59b8"],["540d5e5f.867
fd"],["5cc63b82.b48c04"]]},{"id":"ea913484.bb59b8","type":"
template","z":"2c38f497.2c2a0c","name":"INSERT","field":"
payload","fieldType":"msg","format":"handlebars","syntax":"
mustache","template":"INSERT INTO public.{{payload.0.1}} (\"
time\", machineno, index, x0, x1, x2, x3, x4, x5, x6, x7, x8
, x9, x10, nom0, nom1, nom2, nom3, nom4, nom5, nom6, nom7,
nom8, nom9, nom10,\ndiff, delta_diff, average, temp0, temp1,
 temp2, temp3, temp4, temp5, temp6, temp7, temp8, temp9) \
nvalues ('{{payload.3.1}}','{{payload.1.1}}',{{payload.2.1}}
,{{payload.4.1}},{{payload.5.1}},{{payload.6.1}},{{payload.7
.1}},\n{{payload.8.1}},{{payload.9.1}},{{payload.10.1}},{{
payload.11.1}},{{payload.12.1}},{{payload.13.1}},{{payload.1
4.1}},{{payload.15.1}},{{payload.16.1}},{{payload.17.1}},{{
payload.18.1}},\n{{payload.19.1}},{{payload.20.1}},{{payload
.21.1}},{{payload.22.1}},{{payload.23.1}},{{payload.24.1}},{
{payload.25.1}},{{payload.26.1}},{{payload.27.1}},{{payload.
28.1}},{{payload.29.1}},\n{{payload.30.1}},{{payload.31.1}},
{{payload.32.1}},{{payload.33.1}},{{payload.34.1}},{{payload
.35.1}},{{payload.36.1}},{{payload.37.1}},{{payload.38.1}})
;","output":"str","x":560,"y":300,"wires":[["8f86443b.03a218
"]]},{"id":"540d5e5f.867fd","type":"delay","z":"2c38f497.2c2
a0c","name":"Move","pauseType":"delay","timeout":"2","
timeoutUnits":"seconds","rate":"1","nbRateUnits":"1","
rateUnits":"second","randomFirst":"1","randomLast":"5","
randomUnits":"seconds","drop":false,"x":550,"y":340,"wires":
[["3c3d4d31.7dc1e2"]]},{"id":"784989c5.de9888","type":"fs-
ops-move","z":"2c38f497.2c2a0c","name":"Error","sourcePath":
"topic","sourcePathType":"msg","sourceFilename":"file","
sourceFilenameType":"msg","destPath":"D:\\Test\\Filewatcher
\\Error","destPathType":"str","destFilename":"file","
destFilenameType":"msg","link":false,"x":590,"y":160,"wires"
```

```
:[[]]},{"id":"71a77f6b.76667","type":"delay","z":"2c38f497.2
c2a0c","name":"Move","pauseType":"delay","timeout":"2","
timeoutUnits":"seconds","rate":"1","nbRateUnits":"1","
rateUnits":"second","randomFirst":"1","randomLast":"5","
randomUnits":"seconds","drop":false,"x":470,"y":160,"wires":
[["784989c5.de9888"]]},{"id":"f77ccde5.5a2c3","type":"watch-
directory","z":"2c38f497.2c2a0c","folder":"D:\\\\Test\\\\
Filewatcher\\\\","recursive":0,"typeEvent":"create","
ignoreInitial":true,"ignoredFiles":"","ignoredFilesType":"re
","name":"Filewatcher","x":150,"y":120,"wires":[["42b8d509.3
bf0ac"]]},{"id":"5cc63b82.b48c04","type":"delay","z":"2c38f4
97.2c2a0c","name":"Move","pauseType":"delay","timeout":"2","
timeoutUnits":"seconds","rate":"1","nbRateUnits":"1","
rateUnits":"second","randomFirst":"1","randomLast":"5","
randomUnits":"seconds","drop":false,"x":550,"y":380,"wires":
[["9ec236da.eb3a68"]]},{"id":"9ec236da.eb3a68","type":"fs-
ops-move","z":"2c38f497.2c2a0c","name":"Arkiv","sourcePath":
"D:\\Test\\Filewatcher\\","sourcePathType":"str","
sourceFilename":"file","sourceFilenameType":"msg","destPath"
:"D:\\Test\\Filewatcher\\Arkiv","destPathType":"str","
destFilename":"file","destFilenameType":"msg","link":false,"
x":670,"y":380,"wires":[[]]},{"id":"1f5943dc.3dc0ec","type":
"postgreSQLConfig","name":"DB","host":"localhost","
hostFieldType":"str","port":"5432","portFieldType":"num","
database":"probedata","databaseFieldType":"str","ssl":"false
","sslFieldType":"bool","max":"10","maxFieldType":"num","min
":"1","minFieldType":"num","idle":"1000","idleFieldType":"
num","connectionTimeout":"10000","connectionTimeoutFieldType
":"num","user":"nodered","userFieldType":"str","password":"
nodereduser123456789","passwordFieldType":"str"}]
```

Appendix D

CREATE query for the tables in PostgreSQL database

This appendix contains the query for creating the tables in the PostgreSQL database.

```
CREATE TABLE public.''table'' --Replace ''table'' with actual table name
(
"time" timestamp with time zone NOT NULL,
machineno character varying(20) COLLATE pg_catalog."default" NOT NULL,
index integer,
x0 numeric(9,4),
x1 numeric(9,4),
x2 numeric(9,4),
x3 numeric(9,4),
x4 numeric(9,4),
x5 numeric(9,4),
x6 numeric(9,4),
x7 numeric(9,4),
x8 numeric(9,4),
x9 numeric(9,4),
x10 numeric(9,4),
nom0 numeric(9,4),
nom1 numeric(9,4),
nom2 numeric(9,4),
nom3 numeric(9,4),
nom4 numeric(9,4),
nom5 numeric(9,4),
nom6 numeric(9,4),
nom7 numeric(9,4),
nom8 numeric(9,4),
nom9 numeric(9,4),
nom10 numeric(9,4),
diff numeric(9,4),
delta_diff numeric(9,4),
average numeric(9,4),
temp0 numeric(9,4),
temp1 numeric(9,4),
temp2 numeric(9,4),
temp3 numeric(9,4),
temp4 numeric(9,4),
temp5 numeric(9,4),
temp6 numeric(9,4),
temp7 numeric(9,4),
temp8 numeric(9,4),
temp9 numeric(9,4)
);
-- Convert to hypertable and create index
CREATE UNIQUE INDEX ON ''table'' (machineno, time DESC); -- Replace ''table'' with actual tablename
SELECT create_hypertable('kubelog','time');
-- Edit permissions
```

```
46  GRANT SELECT ON TABLE public.''table'' TO "nodereduser"; -- Replace ''table'' with actual tablename
47  GRANT INSERT, REFERENCES, SELECT, TRIGGER, UPDATE ON TABLE public.''table'' TO "nodereduser"; -- Replace ''table'' with actual tablename
```

Appendix E

CreateTargetKubelog code.

This appendix contains the MATLAB code for removing outliers and creating the target time series.

Listing E.1: Code for removing outliers and creating the target time series in MATLAB.

```
function CreateTargetNAR()
%%% Walter Johansson
% 03.05.2022
% Create arrays and import data
data = csvread('targetsNAR1.csv',1,0);
x2 = data(:,1);
x3 = data(:,2);
x4 = data(:,3);
nom2 = data(:,4);
nom3 = data(:,5);
nom4 = data(:,6);
% Init
outside = [];
x2_final = [];
x3_final = [];
x4_final = [];
x2x3_final = [];
% Subtract nominal value to remove shifting
x2_scaled = x2 - nom2;
x3_scaled = x3 - nom3;
x4_scaled = x4 - nom4;
x2x3_scaled = (x2-x3) - (nom2-nom3);
% Detect outliers and find the indexes with rmoutliers
[A,tfA] = rmoutliers(x2_scaled);
[B,tfB] = rmoutliers(x3_scaled);
[C,tfC] = rmoutliers(x4_scaled);
[D,tfD] = rmoutliers(x2x3_scaled);
% Combine indexes
for i = 1:length(tfA)
    if(tfA(i)==1 || tfB(i)==1 || tfC(i)==1 || tfD(i)==1)
        outside(i) = 1;
    end
end
% Create new arrays without outliers
for i = 1:length(outside)
    if (outside(i)~=1)
        x2_final(end+1) = x2_scaled(i);
        x3_final(end+1) = x3_scaled(i);
        x4_final(end+1) = x4_scaled(i);
        x2x3_final(end+1) = x2x3_scaled(i);
    end
end
% Transpose arrays
x2_final = x2_final';
```

```
45 x3_final = x3_final';
46 x4_final = x4_final';
47 x2x3_final = x2x3_final';
48 % Create target series
49 target = [x2_final,x3_final,x4_final,x2x3_final]
50 % Save variables to file
51 save NARkubelog.mat
52
53 end
```

Appendix F

Nonlinear Autoregressive Neural Network MATLAB code.

This appendix contains the function for generating, training and testing the Nonlinear Autoregressive Neural Network created in this project.

Listing F.1: Code for creating, training and testing Nonlinear Autoregressive Neural Network.

```
function NARkubelog()
%%% Walter Johansson
% 03.05.2022
%%% Load data set
load NARkubelog.mat target
%%% Assign targets to y variable
y = target;
%%% Define arrays
cell_x2 = cell(1,length(y));
cell_x3 = cell(1,length(y));
cell_x4 = cell(1,length(y));
cell_x2x3 = cell(1,length(y));

%%% Fill arrays
for i=1:length(y)
    cell_x2{i} = y(i,1);
    cell_x3{i} = y(i,2);
    cell_x4{i} = y(i,3);
    cell_x2x3{i} = y(i,4);
end

%%% Define training parameters
hidden_neurons = 10;
training = cell_x2x3(1:length(y));
testnr = 6;
test = cell_x2x3(length(y)-testnr+1:length(y));

%%% Creating a NAR network with feedback delays and hidden layers
network = narnet(1:4,hidden_neurons);

%%% Choose training function
network.trainFcn = 'trainbr';

%%% Using preparets matlab function to shift and prepare the data
[Shifted_inputs,d_states,l_states,shifted_targets] =...
    preparets(network,{},{},training);
%%% Training the network
network = train(network,Shifted_inputs,shifted_targets...
    ,d_states,l_states);
%%% Calculate the outputs and states after open-loop training
[Y,f_input_state,f_layer_state] = network...
    (Shifted_inputs,d_states,l_states);
%%% Print performance
performance = perform(network,shifted_targets,Y)
```

```
%%% Closeloop function to prepare for multistep prediction
[network_closed,i_input_state,i_layer_state] = ...
    closeloop(network,f_input_state,f_layer_state);
%%% View network in closed loop mode
%view(network_closed)
%%% Predict 'testnr' of future outputs
Y_closed = network_closed(cell(0,testnr),i_input_state,i_layer_state);
%%% Calculate error for plotting
e = cell2mat(Y_closed)-cell2mat(test);

%%%%% Plotting %%%%%
TS = size(Y_closed,2);
% Target vs predicted
subplot(2,1,1);
x1 = 1:TS;
y1 = cell2mat(test);
y2 = cell2mat(Y_closed);
plot(x1,y1,'b',x1,y2,'g')
title('Target vs predicted');
legend('Target','Predicted');
xlabel('Calibrations');
xticks(1:length(x1));
ylabel('Diff x2x3-Nom [mm]');
% Error
subplot(2,1,2);
y3 = e;
plot(x1,y3,'r',x1,zeros(size(x1)),'b')
title('Model error');
legend('Error','Zero error');
xlabel('Calibrations');
xticks(1:length(x1));
ylabel('Target-Predicted [mm]');

end
```

Appendix G

MATLAB code for PCA.

This appendix contains the function for the PCA.

Listing G.1: Code for PCA.

```
function kubelogPCA()
%%% Walter Johansson
% 04.05.2022
%%% Load dataset containing x2,x3,x4,x2x3,Maskintemp,Omgivelse temp,
%%% Kjølevæske tanktemp and Spindelolje temp
load pca1.mat
%%% Init
num_component = 3;
%%% Normalize dataset
norm = normalize(inputs);
%%% Finding coefficients, scores and percentage of total
%%% variance explained by each principal component
[coefficients,scores,~,~,explained] = pca(norm);
disp(explained);
%%% Plot
subplot(2,1,1)
biplot(coefficients(:,1:num_component),'scores',...
    scores(:,1:num_component),'varlabels',{'x2','x3',...
    'x4','x2x3','temp1','temp2',...
    'temp3','temp4'});
subplot(2,1,2)
bar(explained)
xlabel('Principal components');
ylabel('% of total variance explained');

end
```