

MASTERTHESIS

Advanced Machine Learning – Implementation of Condition-Based and Predictive Maintenance for an Industry 4.0 production plant

vorgelegt von Kenan Deniz zur Erlangung
des Grades eines Master of Science

Angefertigt im Rahmen des Studienganges M. Sc. Automatisierung an der Hochschule für Angewandte Wissenschaften Hamburg, Department Informations- und Elektrotechnik der Fakultät Technik und Informatik

In Kooperation mit:
University of Shanghai for Science and Technology
516 Jungong Road
Shanghai 200093



Erstprüfer: Prof. Dr.-Ing. Florian Wenck
Zweitprüfer: Prof. Dr.-Ing. Shen JianQian

Kenan Deniz

Advanced Machine Learning –
Implementation of Condition-Based
and Predictive Maintenance for an
Industry 4.0 production plant

Masterthesis eingereicht im Rahmen der Masterprüfung im
Studiengang Automatisierung am Department Informations-
und Elektrotechnik der Fakultät Technik und Informatik der
Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Florian Wenck

Zweitgutachter: Associate Prof. Dr.-Ing. Shen JianQiang

Abgegeben am 04. 08. 2024

Kenan Deniz

Thema der Masterthesis

Advanced Machine Learning –
Implementation of Condition-Based and Predictive Maintenance
for an Industry 4.0 production plant

Stichworte

Zustandsbasierte Wartung, Prädiktive Wartung, Industrie 4.0, Internet der Dinge,
Cyber-Physische Systeme, Machine Learning

Kurzzusammenfassung

Diese Arbeit präsentiert die Entwicklung und Implementierung einer zustandsbasierten und prädiktiven Wartungslösung für eine Industrie 4.0 Produktionsanlage. Die Lösung integriert den Datenfluss und die Echtzeitüberwachung über Node-RED, ein prädiktives Wartungsmodell, das in MATLAB trainiert wurde, und die Speicherung von Daten in InfluxDB. Eine Benutzeroberfläche bietet Echtzeit-Datenvisualisierung, automatisierte Warnungen und Benachrichtigungen, um zeitnahe Wartungsmaßnahmen zu gewährleisten.

Kenan Deniz

Title of the paper

Advanced Machine Learning –
Implementation of Condition-Based and Predictive Maintenance
for an Industry 4.0 production plant

Keywords

Condition-based maintenance, predictive Maintenance, Industry 4.0, Internet of Things, Cyber-Physical Systems, Machine Learning

Abstract

This thesis presents the development and implementation of a condition-based and predictive maintenance solution for an Industry 4.0 production plant. The solution integrates data flow and real-time monitoring using Node-RED, a predictive maintenance model trained in MATLAB, and data storage in InfluxDB. A user interface provides real-time data visualization, automated alerts, and notifications, ensuring timely maintenance actions.

Table of contents

Abstract	III
List of Figures	VII
List of Tables	X
List of Abbreviations	11
1 Introduction	13
1.1 Motivation	13
1.2 Aim of the thesis.....	13
1.3 Structure of the thesis.....	14
2 Theoretical Foundations	15
2.1 Maintenance	15
2.2 Maintenance Strategies	16
2.3 Reactive Maintenance	17
2.4 Preventive Maintenance.....	18
2.4.1 Periodic Maintenance.....	18
2.4.2 Condition-Based Maintenance	20
2.4.3 Predictive Maintenance.....	24
2.5 Industry 4.0	26
2.5.1 Cyber-Physical Systems	27
2.5.2 Smart Objects	29
2.5.3 Internet of Things	29
2.5.4 Internet of Things and Services.....	31
2.5.5 Big Data	31
2.6 Machine Learning	32
2.6.1 Supervised Learning	33
2.6.2 Unsupervised Learning.....	35
2.6.3 Reinforcement Learning	36
2.6.4 Machine Learning Algorithms	36
2.6.5 Support Vector Machines.....	37
2.6.6 Isolation Forest	46
2.6.7 Artificial Neural networks	48

2.6.8 Deep Learning	62
3 The Industry 4.0 Production Plant	64
3.1 Products and production process	65
3.2 Components	69
3.2.1 IMS Stations	69
3.2.2 KUKA Robot	76
3.2.3 ERP-Lab	78
3.2.4 The production plant in terms of Industry 4.0	81
3.2.5 Controller network and data blocks	82
4 Concept	86
4.1 Functional requirements analysis	86
4.2 Selection of necessary tools.....	88
5 Implementation and deployment	94
5.1 Communication and data flow.....	94
5.1.1 Prerequisites	94
5.1.2 Data flow within Node-RED	95
5.1.3 Storing the data in InfluxDB	100
5.1.4 Accessing the data in Grafana	101
5.2 Condition Monitoring and Alert Management	102
5.2.1 Condition monitoring	103
5.2.2 Maintenance alerts, notifications and alert-data	106
5.2.3 Notification interruptions	116
5.3 User interface and dashboards	118
5.3.1 Node-RED dashboards	118
5.3.2 Grafana dashboards	123
5.4 Data analysis through Machine Learning.....	126
5.4.1 Acquiring representative data	127
5.4.2 Data analysis of conveyor belts	128
5.4.3 Anomaly detection for the KUKA robot	139
5.5 Integration.....	143
5.5.1 Performing predictions within MATLAB	143
5.5.2 Accessing the predictions within Node-RED	146
6 Functional Testing	148

6.1	General functionality	148
6.2	Accessibility of data	149
6.3	Condition-based and preventive maintenance capabilities.....	152
6.4	Notifications	155
7	Conclusion	158
7.1	Summary.....	158
7.2	Outlook	160
	Bibliography	161
	Appendix	168

List of Figures

Figure 2.1: Maintenance Strategies (adapted from [3], p. 18; [5], p. 27 and [6], p. 5).....	16
Figure 2.2: Visualization of the reactive maintenance strategy (adapted from [27])	17
Figure 2.3: Visualization of the periodic maintenance strategy (adapted from [27]).....	18
Figure 2.4: Visualization of condition-based maintenance (adapted from [27]).....	20
Figure 2.5: Visualization of the predictive maintenance strategy (adapted from [27])	24
Figure 2.6: The four industrial revolutions (adapted from [19], p. 30).....	26
Figure 2.7: Cyber-physical systems (adapted from [21], p. 11).....	28
Figure 2.8: Evolution of industrial hierarchies through IoT (adapted from [26]).....	30
Figure 2.9: Machine Learning vs classic imperative programming (adapted from [31])	32
Figure 2.10: Identification of anomalies (adapted from [34], p. 14).....	36
Figure 2.11: Two groups of data points in a feature space (adapted from [38])	37
Figure 2.12: Example of insufficient hyperplanes (adapted from [38]).....	38
Figure 2.13: Hyperplanes with additional datapoints (adapted from [38])	39
Figure 2.14: Optimal hyperplane with support vectors (adapted from [38])	39
Figure 2.15: Hard-margin-classification and outliers (adapted from [34], p. 157)	41
Figure 2.16: Soft-margin-classification (adapted from [35], p. 71)	42
Figure 2.17: Soft-margin-classification and parameter C (adapted from [34], p. 157)	42
Figure 2.18: Adding another feature dimension (adapted from [40])	43
Figure 2.19: Separation by hyperplane and back transformation (adapted from [40])	44
Figure 2.20: Visualization of an iForest (adapted from [42]).....	46
Figure 2.21: Perceptron [43]	49
Figure 2.22: Sigmoid, tanh and ReLU activation function (adapted from [46], p. 36)	50
Figure 2.23: Fully connected feedforward neural network (adapted from [46]).....	51
Figure 2.24: Optimizing through loss score (adapted from [51], p. 11)	53
Figure 2.25: Gradient descent on a one-dimensional curve [52]	54
Figure 2.26: Learning rate, too low vs too high (adapted from [34], p. 122)	55
Figure 2.27: Plateau, local and global minimum within loss function (adapted from [34]) ...	56
Figure 2.28: 3-fold cross-validation [51].....	58
Figure 2.29: Structure of an Autoencoder [54]	63
Figure 3.1: Industry 4.0 production plant [67]	64
Figure 3.2: Individual workpieces (adapted from [57]).....	65
Figure 3.3: Schematic plant overview (adapted from [58]).....	66
Figure 3.4: RFID read and write units and carrier flow (adapted from [58])	67
Figure 3.5: Conveyor belt IMS Station 1 [67]	70
Figure 3.6: IMS Station 3a (adapted from [57] (left) and [67] (right)).....	71
Figure 3.7: IMS Station 4a (adapted from [57] (left) and [67] (right)).....	72
Figure 3.8: IMS Station 4a (adapted from [57] (left) and [67] (right)).....	73
Figure 3.9: IMS Station 6 (adapted from [57] (left) and [67] (right)).....	74
Figure 3.10: IMS Station 7 (adapted from [57] (left) and [67] (right)).....	75
Figure 3.11: KUKA robot KR6 R700 (adapted from [57] (left) and [67] (right))	77
Figure 3.12: Design and structure of ERP-Lab [58]	78
Figure 3.13: ERP-Lab v2.1.0 (Order view) [67].....	79
Figure 3.14: ERP-Lab v2.1.0 (SCADA) [67]	80
Figure 3.15: OpenCart web shop [67]	81
Figure 3.16: TIA Portal network overview showing PLC and KRC4 connections [67]	82
Figure 3.17: Cropped view of PLC_1's data blocks [67].....	83
Figure 4.1: Black box of the condition-based and predictive maintenance solution [67]	86

Figure 4.2: Grey box including the necessary tools [67]	88
Figure 4.3: White box including the specific combination of tools [67]	92
Figure 4.4: Outline of the tool's architecture and data flow [67]	93
Figure 5.1: OPC UA Item nodes within Node-RED [67]	95
Figure 5.2: Central timestamp and OPC UA Client [67]	97
Figure 5.3: Settings of OPC UA Client Central [67]	97
Figure 5.4: Addresses of the OPC UA server [67]	97
Figure 5.5: Data flow of IMS Station 1 [67]	98
Figure 5.6: Settings of the InfluxDB-out node [67]	99
Figure 5.7: Stored data within InfluxDB [67]	100
Figure 5.8: View of the tabular data and Flux query within InfluxDB [67]	101
Figure 5.9: InfluxDB as a data source within Grafana [67]	102
Figure 5.10: Query language setting in Grafana [67]	102
Figure 5.11: Condition monitoring flow	103
Figure 5.12: Condition monitoring of IMS Station 4a [67]	104
Figure 5.13: Settings of an excursion node [67]	104
Figure 5.14: Settings of the delay node [67]	105
Figure 5.15: Central control of notifications [67]	106
Figure 5.16: Creation of maintenance alerts and pop-up notifications [67]	107
Figure 5.17: Initialization of variables [67]	108
Figure 5.18: Iteration over each Item within the array [67]	109
Figure 5.19: Adding the last part of the alert message [67]	110
Figure 5.20: Settings of e-mail node (left) and notification node (right) [67]	110
Figure 5.21: Function for audio node [67]	111
Figure 5.22: Logging alert-data into the database [67]	111
Figure 5.23: Alert-Table function nodes [67]	112
Figure 5.24: Alert-table in central flow storage [67]	113
Figure 5.25: Adding rows to the alert-table [67]	114
Figure 5.26: Alert table for real-time data	115
Figure 5.27: Nodes providing silencer functionality [67]	116
Figure 5.28: Dashboard layout IMS (Counter-View) [67]	119
Figure 5.29: Charts of IMS Station 1 within the Energy-View [67]	120
Figure 5.30: Dashboard layout IMS (Energy-View) [67]	120
Figure 5.31: Charts of the axis motor temperature within the KUKA Overview [67]	121
Figure 5.32: Layout of the Alert-Dashboard [67]	122
Figure 5.33: Test audio signal button [67]	122
Figure 5.34: Tabs and links of the user interface [67]	123
Figure 5.35: Creation of a time series graph with flux queries in Grafana [67]	124
Figure 5.36: The current of each KUKA axis as an excerpt of its dashboard [67]	124
Figure 5.37: Dashboard of IMS Station 4a [67]	125
Figure 5.38: List of all Grafana dashboards [67]	126
Figure 5.39: Normal power consumption of the conveyor belt motors [67]	129
Figure 5.40: Ground level of circular arranged conveyor belt motors [67]	129
Figure 5.41: Preprocessing [67]	130
Figure 5.42: Data preprocessed and centered around zero [67]	130
Figure 5.43: Visualization of the different classes [67]	131
Figure 5.44: Scatter plot of the signal statistics peak value ... mean	133
Figure 5.45: Creating a session in the "Classification Learner" app [67]	134
Figure 5.46: Excerpt of the ranking based on validation accuracy [67]	135
Figure 5.47: Validation confusion matrix of quadratic SVM with all features [67]	135

Figure 5.48: Ranked classifiers trained with smaller sample set [67].....	137
Figure 5.49: Validation confusion matrix of trained quadratic SVM [67].....	137
Figure 5.50: Quadratic SVM training and test results [67].....	138
Figure 5.51: normal data (left) and anormal data (right).....	139
Figure 5.52: Creation of the binary SVM [67].....	140
Figure 5.53: Confusion matrix SVM [67]	140
Figure 5.54: Creation of the Isolation Forest [67]	141
Figure 5.55: Confusion matrix iForest [67].....	141
Figure 5.56: Creation of the AE [67]	142
Figure 5.57: Defining the threshold [67]	142
Figure 5.58: Confusion matrix AE [67]	143
Figure 5.59: Communication between MATLAB and InfluxDB [67].....	144
Figure 5.60: Subtract baseline values [67].....	144
Figure 5.61: Performing the prediction [67]	145
Figure 5.62: Calling the machine learning prediction in threads [67].....	146
Figure 5.63: Prediction data flow [67]	147
Figure 6.1: Sidebar menu [67].....	148
Figure 6.2: Excerpt of dashboards [67].....	149
Figure 6.3: Plant-Overview [67].....	150
Figure 6.4: Grafana dashboard [67]	151
Figure 6.5: Grafana time range selector [67].....	151
Figure 6.6: Alert tables [67]	152
Figure 6.7: Predictive maintenance output [67]	153
Figure 6.8: Logging of threshold exceeding data in InfluxDB [67].....	153
Figure 6.9: Visualization of the states within Grafana [67]	154
Figure 6.10: Maintenance-Alert pop-up [67]	155
Figure 6.11: Machine Learning alert [67]	156
Figure 6.12: E-Mail notifications [67].....	156
Figure 6.13: Alert-Dashboard [67].....	157

List of Tables

<i>Table 2.1: Generally used kernels</i>	<i>45</i>
<i>Table 3.1: Conveyor belt [59]</i>	<i>70</i>
<i>Table 3.2: IMS Station 3a and 3b [59]</i>	<i>72</i>
<i>Table 3.3: IMS Station 5a and 5b [59]</i>	<i>73</i>
<i>Table 3.4: IMS Station 6 [59]</i>	<i>74</i>
<i>Table 3.5: IMS Station 7</i>	<i>76</i>
<i>Table 3.6: Data IMS Stations (adapted from [59])</i>	<i>84</i>
<i>Table 3.7: Data KUKA robot (adapted from [59])</i>	<i>85</i>
<i>Table 4.1: Tool Decision Matrix</i>	<i>89</i>
<i>Table 5.1: Local and network addresses of InfluxDB, Node-RED and Grafana [67]</i>	<i>94</i>

List of Abbreviations

DIN	Deutsches Institut für Normung
EN	European Norm
CPS	Cyber-physical systems
IoT	Internet of Things
IoTS	Internet of Things and Services
IP	Internet Protocol
CT	Computer Tomography
SVM	Support vector machine
IF	Isolation Forest
MSE	Mean squared error
ANN	Artificial neural network
NN	Neural network
FNN	Feed forward neural network
RNN	Recurrent neural network
ReLU	Rectified linear unit
Tanh	Tangens hyperbolicus
SGD	Stochastic gradient descent
AE	Autoencoder
PLC	Programmable Logic Controller
ERP	Enterprise Resource Planning
MES	Manufacturing Execution System
CoAP	Constrained Application

List of Abbreviations

UDP	User Datagram Protocol
HTML	Hypertext Markup Language
API	Application Programming Interfaces

1 Introduction

1.1 Motivation

The concept of Industry 4.0 represents the ongoing digital transformation within the production industry, aiming to connect all machines, systems, and processes within a network through information and communication technologies (cf. [28]). By integrating smart and intelligent systems and creating an interconnected network, information can be accessed and processed automatically.

Industry 4.0 has significantly evolved over the years, with growing emphasis on the connectivity of equipment, enhancing efficiency, and improving system availability. Today the implementation of maintenance measures not only aims to reduce costs and prevent failures but also to predict and identify potential issues before they occur. This maintenance strategy helps to minimize downtime and extend the lifespan of equipment. As a result, the demand for condition-based and preventive maintenance solutions is growing.

1.2 Aim of the thesis

The development of condition-based and predictive maintenance concepts is an ongoing and growing task, where not much complete solutions and accessible examples exist, especially on a smaller scale. Most existing work focuses on theoretical approaches and the development of complex machine learning algorithms, without a following practical implementation and integration within actual production plants, machines or systems.

The goal of this thesis is to implement a condition-based and predictive maintenance solution for an Industry 4.0 production plant manufactured by Lucas-Nuelle and located at the Shanghai-Hamburg College of the University of Shanghai for Science and Technology. The focus is on developing a solution that integrates and displays all available data and informational inputs, making them accessible to operators and maintenance personnel. This solution aims to provide a comprehensive real-time overview, developing incorporating condition monitoring and machine learning outputs. The objective is not to create the most advanced

machine learning algorithm but to develop a practical and effective solution suited for the task and real-time application.

1.3 Structure of the thesis

Chapter 2 introduces the theoretical foundations relevant to this thesis, covering various maintenance strategies such as condition-based and predictive maintenance, Industry 4.0, and machine learning concepts.

Chapter 3 presents the Industry 4.0 production plant, detailing its setup and operations.

Chapter 4 describes the development of the condition-based and predictive maintenance solution, including the functional requirement analysis and selection of necessary tools.

Chapter 5 focuses on the implementation of the solution, starting with the establishment of the data flow from the PLC to Node-RED, data storage, and the integration of condition monitoring. It also covers the creation of maintenance alerts, dashboards, and the training of a machine learning algorithm in MATLAB.

Chapter 6 tests the solution, presenting its user interface and functionalities, in regard to the defined requirements.

Chapter 7 provides a summary of the thesis and offers future outlooks for further development and improvements.

2 Theoretical Foundations

The goal of this thesis is to implement a condition-based and predictive maintenance solution for an industry 4.0 production plant based on machine learning and the internet of things. In the following chapter the fundamentals of maintenance, as well as industry 4.0 and machine learning are explained.

Due to the nature and content of this thesis, all following theoretical foundations are explained within a production context. For mathematical equations, the following notation is defined, unless otherwise specified: Vectors are denoted by bold lowercase, such as \mathbf{v} and matrices are represented by bold uppercase letters, such as \mathbf{A} . Sets are represented by uppercase letters and scalars are denoted by lowercase letters.

2.1 Maintenance

Maintenance unites the actions of servicing, inspection, repair and improvement of a plant, machine, system, or unit in order to keep its functionality and performance as intended, as well as to prevent any breakdowns or, in case of a failure or malfunction, restore it to its desired state (cf. [1], p. 4). The purpose of maintenance measures in a production environment is to prolong and maximize the current run- and total lifetime of a unit as well as to minimize downtime and ensure a consistent quality and reliability of the production processes.

According to the DIN 31051 and DIN EN 13306 maintenance is defined as the entirety of all administrative, managerial, and technical steps with the purpose to maintain and restore a functional state, i.e. the functionality, of a unit during its life cycle (cf. [1], p. 4 and [2]). The application and frequency of these measures, as well as the specific order in which they are implemented or planned, depend on the chosen maintenance strategy. Used effectively, maintenance strategies contribute to increased productivity, reduced operational costs, improved safety, and overall efficiency (cf. [3], pp. 15-16 and [4]).

2.2 Maintenance Strategies

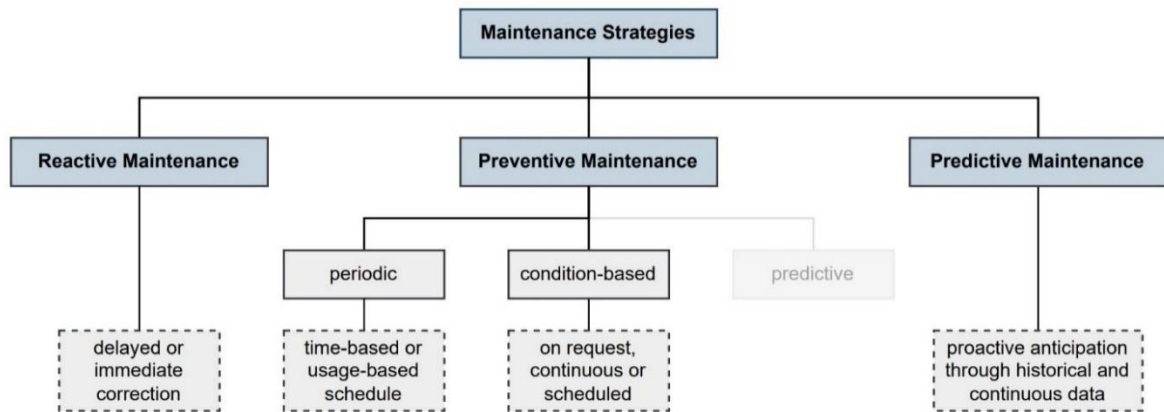


Figure 2.1: Maintenance Strategies (adapted from [3], p. 18; [5], p. 27 and [6], p. 5)

Maintenance strategies describe the approach to achieve defined maintenance goals through specific maintenance measures. These maintenance strategies can be divided into three distinct groups: reactive maintenance, preventive maintenance, and predictive maintenance. While predictive maintenance can be considered a preventive maintenance strategy, since it aims to achieve the shared objective to prevent breakdowns or malfunctions through predictive measures before they occur, it is treated hereafter as a separate maintenance strategy (cf. [7], p. 189 and see Fig. 2.1).

In the following, the different maintenance strategies as well as their associated measures are explained:

2.3 Reactive Maintenance

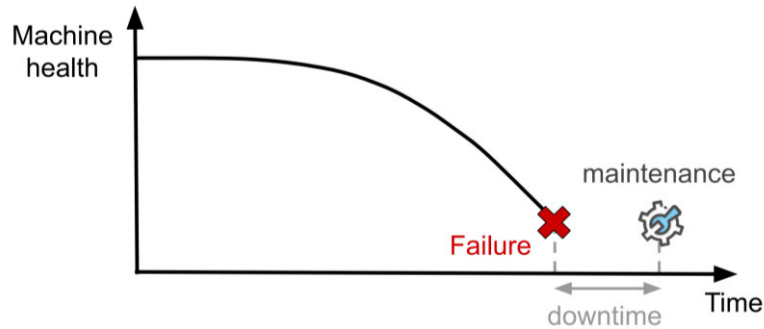


Figure 2.2: Visualization of the reactive maintenance strategy (adapted from [27])

Reactive maintenance, also known as corrective maintenance or breakdown maintenance, follows a simple principle: if it is broken, it will be fixed. In accordance with this principle reactive maintenance takes place if and after a fault, failure or breakdown occurs (see Fig. 2.2). It involves reacting to faults and replacing or repairing broken or faulty parts with no prior maintenance schedules or planning. Therefore, it requires a quick maintenance response and available maintenance personnel, as faults cannot be anticipated and can occur at any time and scale as well as on different systems or parts simultaneously. Moreover, for a replacement or repair to take place, the actual fault needs to be localized or identified, if not already known. As a result, heavier breakdowns, a shortage of spare parts or waiting for external suppliers or personnel can lead to longer downtimes (see Fig. 2.2). This in turn can lead to higher maintenance and production costs as well as safety risks since the safety and continuous availability of the production plants cannot be guaranteed or lead times cannot be met.

Reactive maintenance should therefore only be used on parts and machine components that won't significantly impact the productivity and safety, in case a failure or breakdown occurs, and are less critical to the overall operation of a system or plant as well as cost-efficient and easy to identify and replace. Since this maintenance strategy does not require prior planning or scheduling and the entire lifecycle of a component is used at the time of replacement, it can potentially reduce maintenance costs compared to preventive maintenance (see Section 2.4), if the risks resulting from a failure are low. Faults or failures can be corrected immediately or repairments can be delayed according to their criticality and the impact of a systems functionality (cf. [3], p. 18). Paired with preventive maintenance strategies, a balanced cost-efficiency and system reliability can be achieved.

2.4 Preventive Maintenance

The core strategy behind preventive maintenance is to replace parts or machine components regularly before a fault or malfunction occurs. The frequency hereby can be chosen or influenced based on time, usage, manufacturer instructions and usage-experience, for example whether a component is prone to failure or not. The DIN EN 13306 defines it as maintenance, that is “carried out at predetermined intervals or according to prescribed criteria to reduce the probability of failure or the probability of limited functional performance of a unit” (cp. [2] and [6], p. 7).

Based on the selected criteria and execution, preventive maintenance can be categorized into two groups: periodic and condition-based maintenance. These will be described in the following sections.

2.4.1 Periodic Maintenance

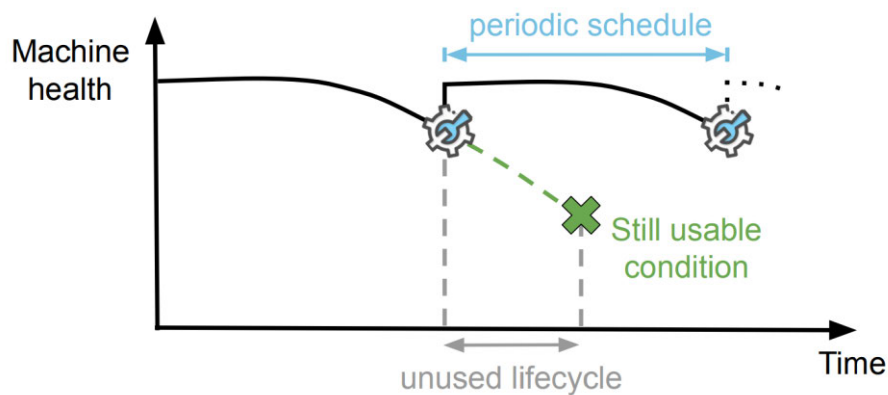


Figure 2.3: Visualization of the periodic maintenance strategy (adapted from [27])

Periodic maintenance, also known as predetermined maintenance (cf. [5], p. 19), is carried out according to predesigned schedules that follow specific usage-criteria (see Fig. 2.3). These criteria or the chosen periodic maintenance schedules can be time-based, for example based on the time of first installment, last maintenance measure, fixed dates or number of instructed maintenance measures per year, or usage-based, based on the time a machine or production unit is actually running, the number of parts produced or the number of activations of a unit, motor, part, sequence etc.

The maintenance or replacement measures that are carried out based on mentioned schedules are hereby not affected or influenced by the actual condition of the plant or machine components at the time of service. Instead, they are influenced by manufacturer instructions, usage-experience within the production environment and, if applicable, knowledge of previous breakdowns during the calculation and determination of the maintenance schedules prior to their execution (cf. [3], pp. 28-30; [5], p. 19 and [7], pp. 174-175).

The benefit of this maintenance strategy is, that malfunctions and failures are prevented or delayed by preventive measures, unlike reactive maintenance where maintenance measures take place after faults or breakdowns occur. This makes maintenance measures and related tasks easier to plan and manage. Maintenance measures can be carried out during reserved and specific time slots where a plant or machine is not running and the production is not disturbed. In addition, spare parts can be prepared and stocked accordingly. The required amount of maintenance personnel can be reserved and since the task is known beforehand and performed regularly, the processes can be optimized and the downtime limited or minimized to a needed minimum.

This maintenance strategy requires a lot of planning and preparation, but in return, it is able to prolong and secure the safety and availability of a production plant or unit, while reducing both planned and unplanned maintenance expenses and repair times. However, periodic maintenance can have the opposite effect if components of a system are stressed too often by preventive measures, which increases both the costs and the time required, as far too much time is spent on parts that are actually working, potentially causing damage to the surroundings. Moreover, if parts are replaced regularly without utilizing most of their lifecycle (see Fig. 2.3), aside from the costs, potential value and remaining functionality are lost, leading to increased waste of material and spare parts as well as environmental impact due to unnecessary disposal. On the other hand, this maintenance strategy is only beneficial and fulfills its purpose, if a part is replaced in time before the entire lifecycle is used and a breakdown occurs. This means that the amount of preventive measures must be carefully assessed, as there is a fine line between correct and incorrect preventive or periodic maintenance, especially given that the plant is not under constant monitoring and the actual condition of the plant, a machine or part is unknown (cf. [3], p. 29 and [8], p. 17).

2.4.2 Condition-Based Maintenance

Condition-based maintenance is a preventive maintenance strategy that determines maintenance measures depending on the technical condition of a part or unit, rather than predetermined schedules (see Fig. 2.4). It relies on sensors and monitoring devices to continuously collect data and supervise the condition of the technical equipment. The collected real-time data can be analyzed immediately and continuously, according to a schedule or upon request to determine if maintenance measures are necessary (see Fig. 2.1).

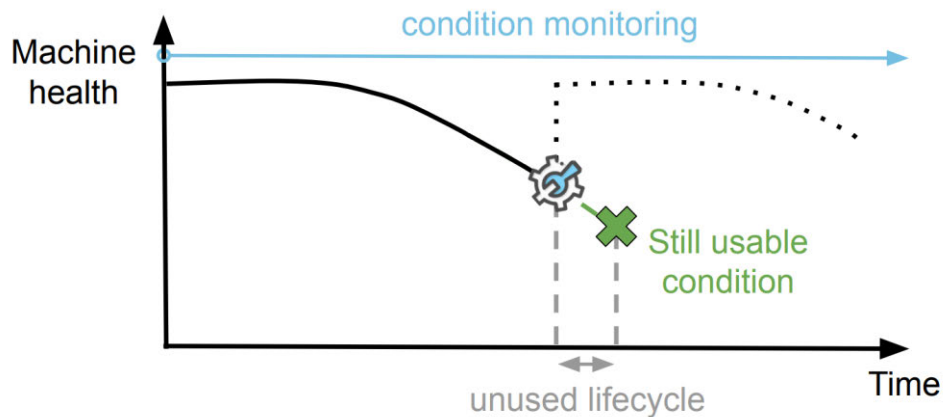


Figure 2.4: Visualization of condition-based maintenance (adapted from [27])

Condition-based maintenance is defined as “maintenance, that consists of monitoring the operation and/or representative measurements as well as the subsequent measures” (cp. [2] and [6], p. 7). It enables the user of this maintenance strategy to implement adaptive procedures and plan maintenance activities based on the collected data and the condition of the plant, machine, or unit they represent. This maintenance strategy heavily relies on monitoring and data acquisition, which function as a foundation to identify specific indicators and defined thresholds in order to determine and assign resulting maintenance tasks.

Depending on the components and structure of a machine or system as well as safety and production-relevant parts that need to be observed and monitored, different technologies and approaches can be used to collect representative data for its condition [9]. These technologies, the collected data and possible indicators for related conditions of a part or unit in a production environment will be explained in the following paragraphs (cf. [9], [11], [12] and [13]):

Monitoring:

- Vibrations: Motors, fans, compressors, and pumps contain rotating components to generate the necessary movement to convert energy or transport liquids and gases. These rotating components also produce vibrations, which can be measured and evaluated to obtain information about their condition. Vibration sensors are used to measure the intensity, frequency, and amplitude of these vibrations in order to detect deviations and irregularities before a malfunction or breakdown occurs. Increased vibrations result from various factors such as age, wear, bearing damages, imbalance, loose elements, or other faults which, if left unattended, can lead to breakdowns or failures.
- Acoustic-Signals: Mechanical errors and leakages, such as worn-out or loose components, as well as leaks of lubricants, liquids, or gases, can generate noise or alternating sounds within a system. Sensors, microphones, or ultrasonic measurements are used to detect anomalies or capture high- or low-frequency sound waves. Early detection of lubrication deficiencies or slight mechanical deformations as well as pressure or vacuum leaks prevent further issues and help to identify the origin of resulting problems.
- Pressure: Pressure levels can be monitored and evaluated using pressure sensors and appropriate measurement devices. This enables precise control of pressure within tanks, tubes, and pipelines, as well as the adjustment of gas and liquid flow rates and velocities using additional formulas such as Bernoulli's equation [10]. Leakage detection is also possible through continuous pressure monitoring, allowing for early identification and mitigation of potential leaks.
- Temperature and infrared radiation: Temperature sensors or infrared cameras are utilized to monitor the temperature of technical equipment and to identify any signs of overheating or temperature loss. Temperature monitoring helps to prevent abnormal temperatures from causing potential damage to the equipment itself, its surroundings, or the production environment, thereby minimizing safety hazards. Monitoring of gases, liquids, insulation, motors, bearings as well as identifying points of

temperature loss and leaks, are possible use cases contributing to maintain operational efficiency and safety standards.

- Power consumption: The current, voltage or power consumption of a machine or unit can be monitored in order to detect spikes, drops or increased consumption as well as their origins. This enables the implementation of necessary maintenance measures and early detection of wear and tear.
- Oil and lubricants: Oil and lubricants are utilized for cooling and to reduce vibrations, wear, friction and corrosion within machines or systems in a production environment. Over time, particles get suspended into the oil or lubricant and can serve as indicators of wear and tear within the machine or system they are used in as well as if the oil and lubricants are still usable themselves. Monitoring the presence and concentration of these particles can provide insights into the condition of the equipment and can help to determine resulting maintenance needs.
- Operational performance: Another factor indicating the condition of a machine, unit, or system, which can be monitored, is its operational performance. During production, a machine must achieve specific operational requirements while reliably and consistently performing its intended function. These requirements can include positional accuracies, a certain amount of force or speed, as well as other requirements influencing the quality and condition of the final product manufactured, created, or made within or with the help of the system. If the operational performance cannot be maintained reliably, accurately, and consistently, it can indicate a deterioration in its condition, with wear and tear being a common example.
- Environmental conditions: External factors that can influence a machine or its components and their operational performance, as well as sensors. These factors can include, for example, solar radiation, ambient temperature and air humidity in the production environment.

Depending on the technologies, data and indicators as well as operational specifications of the machines, systems or units, thresholds can be defined to trigger corresponding

maintenance alarms or measures. This ensures that maintenance measures are carried out when the deterioration or degradation of a system reaches or surpasses a certain point or degree. Based on these thresholds, their types, and the corresponding circumstances, such as whether an immediate response is required, if all resources are available, or if preparations for spare parts, maintenance personnel, or other resources need to be initiated and completed within an estimated time window, the optimal timing for maintenance can be determined and the corresponding tasks can be performed [14].

If implemented and used correctly, this maintenance strategy helps to lower costs and maintenance expenses due to early detection of anomalies and faults, therefore reducing heavier and expensive breakdowns and preventing unnecessary measures, such as replacements and repairs on healthy machines by monitoring the condition. Coupled with the ability to perform preventive measures, when indicated by the condition of a machine, part or unit, it increases the availability and safety by minimizing potential damages and failures as well as repair and measure induced unavailability and production loss, positively influencing the useful life- and runtime overall. In comparison to periodic maintenance, it enables further utilization of a component's lifecycle as illustrated by comparing the Figures 2.3 and 2.4. Additionally, spare parts can be prepared when needed, therefore vast quantities of spare parts as well as large storages are not required, which makes this maintenance strategy all in all technically and economically efficient (cf. [3], p. 27 and [15]).

Due to the necessity of sensors and monitoring devices, this maintenance strategy is cost-intensive at an early stage, during implementation and, if applicable, during upgrading or retrofitting existing machines or units. The latter can be particularly challenging if the data flow infrastructure is also non-existent, where a common goal is to collect as much data with as few sensors as possible (cf. [5], p. 31). Initially, aside from the costs and installation, configuring the sensors, the data flow as well as the underlying parameters and algorithms also require high and time-consuming effort. Condition-based maintenance is only possible, when the indicators used for this strategy are measurable and can be used to detect issues and determine reliable maintenance actions effectively. When a failure or breakdown occurs before a threshold is reached or without a known indicator, it can lead to unexpected and longer downtimes, due to troubleshooting and repair efforts as well as a possible spare part shortage, negating the intention and benefits of condition-based maintenance. Therefore this strategy should only be used when its technically applicable and more cost- and time-

effective in the long run compared to other maintenance strategies, taking into account the nature of the task and the monitored machine or unit [5].

2.4.3 Predictive Maintenance

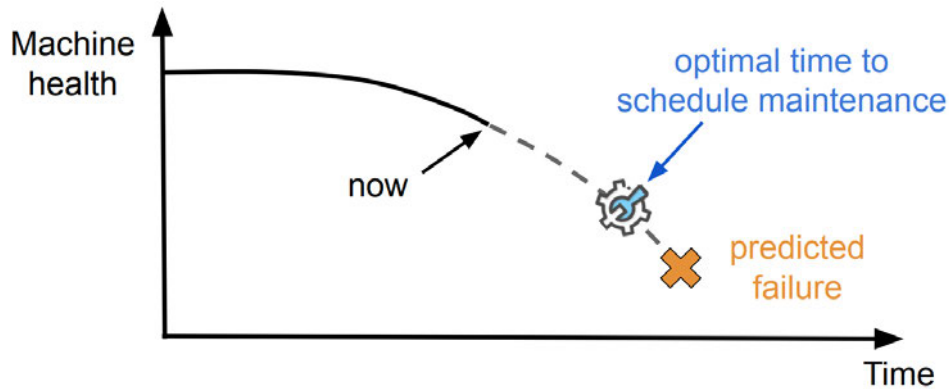


Figure 2.5: Visualization of the predictive maintenance strategy (adapted from [27])

Predictive maintenance is a further development of the condition-based maintenance strategy (cf. [5], p. 31). While it equally relies on sensors and monitoring devices to continuously collect data, it goes beyond supervising the condition of the technical equipment and aims to anticipate failure-modes and patterns as well as fault-conditions and signs of deterioration and breakdowns (see Fig 2.5). It is therefore defined as “condition-based maintenance, which is performed based on a prediction derived from repeated analysis or known characteristics and the determination of key parameters that indicate the deterioration of a unit” (cp. [2]). To identify preliminary stages of failure in or in addition to subtle changes in the condition of a machine, identify the origin and estimate the remaining useful lifetime, computer-aided algorithms and models, such as machine learning algorithms and neural networks (see Section 2.6), are used to analyze, evaluate and classify the corresponding sensor-data and compute accurate predictions.

These predictions are achieved by constantly monitoring the condition and performance of technical equipment, using the same technologies and monitoring techniques described in Section 2.4.2, and registering anomalies in comparison to its normal and healthy behavior. This entails that a sufficient amount of historical data, representative of both normal and abnormal behavior, alongside the collected real-time data, is available.

Predictive maintenance shares the same benefits as its condition-based predecessor (see Section 2.4.2). Implemented correctly, it not only allows the user to identify potential problems before a malfunction or failure occurs but also enables the determination of the underlying cause and origin, as well as an estimated time window during which the observed item is likely to be subject to the identified issue. By recognizing failure-modes, patterns and features, predictive maintenance systems are able to learn from historical data and match inputs to known trends and classes, resulting in:

1. Classification of equipment condition
2. Estimation of remaining useful lifetime

as the two primary use cases of predictive maintenance, relying on the premise that indicators of degradation increase gradually over time [16].

This maintenance strategy, united with analytical techniques and models, and in addition to the benefits of condition-based maintenance (see Section 2.4.2), enables early and specific detection, maximizing the uptime, productivity and lifetime of assets when mitigated. Maintenance measures can be carried out based on the specific needs and the condition of the assets, therefore making them plannable and applicable when needed and at the right time in terms of useful lifetime, remaining value, maintenance costs, safety and overall efficiency (see Fig. 2.5).

Predictive maintenance also shares similar disadvantages and challenges as condition-based maintenance, regarding the high initial costs for sensors and monitoring devices. Additionally, data processing units, databases and the necessary infrastructure contribute to the fundamental costs, along with the essential expertise in analytics and data science for the development and training of appropriate models and algorithms [16].

Another challenge at the beginning of establishing a preventive maintenance strategy is the collection of sufficient and representative data. In a production environment, the goal is to run machines and units without a breakdown or failure occurring. Therefore, data representative of abnormal or unhealthy behavior is often not available or only partially available. Moreover, differences in mounting, mechanical tolerances, as well as operational and environmental circumstances, can lead to data variability between assets in a production

environment that share the same specifications, which influences the reusability of algorithms and models [17]. Simulations and data augmentation can help overcome this obstacle, but they need to be accurate to improve the reliability of predictive algorithms.

Machine connectivity and data availability play an essential role in predictive maintenance. With the development of cyber-physical systems, the internet of things and the concept of Big Data, all within the framework of Industry 4.0, the potential for real-time monitoring and analysis of machine performance has greatly expanded. Machine data can be collected at any stage of the production process and made available across multiple production facilities worldwide. Thus, the behavior of machines can be fully observed, and vast facets of data can be used to train and tune predictive maintenance algorithms and models.

Contributors from different departments and facilities can benefit from each other's knowledge and combined with expertise of different backgrounds (cf. [18], p. 21), such as operators, maintenance personnel and machine manufacturers, representative datasets for specific problems and failure modes can be created, know-how can be combined and developed into accurate and automated predictive maintenance solutions, enhancing failure detection and anticipation, minimizing downtime, material and spare part wastage as well as overall maintenance costs [17]. In the following, Industry 4.0 and its components are further explained:

2.5 Industry 4.0

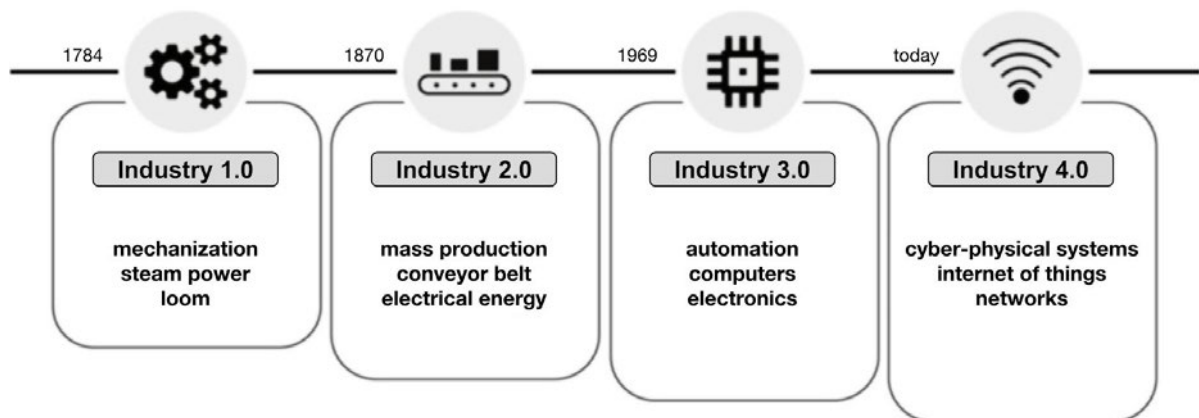


Figure 2.6: The four industrial revolutions (adapted from [19], p. 30)

Industry 4.0 describes the latest and fourth industrial revolution (see Fig. 2.6). It represents the ongoing digital transformation within the production industry, aiming to connect all machines, systems, and processes within a network through information and communication technologies (cf. [28]). By integrating smart and intelligent systems and creating an interconnected network, information can be accessed and processed automatically. As a result, the production can be optimized, productivity can be increased, and data can be shared across all production levels and facilities worldwide. The intention is to enable constant availability and accessibility of data and the resulting ability to autonomously derive actions from it (cf. [18], [19], [20], [21], [22] and [24]).

These smart and intelligent systems, in the form of cyber-physical systems and smart objects, along with the Internet of Things as their network, and the resulting availability and accessibility of so-called "Big Data", form the basis for optimization and automated decision-making. These elements are the core and foundation of Industry 4.0 and will be explained in the following sections:

2.5.1 Cyber-Physical Systems

Cyber-physical systems, or short CPS, consist of a group of three components, a physical, an intelligent and a network component, combining the elements of mechanics, electronics and information technology within one system [19]. Incorporating actuators, sensors, processing units, multiple interfaces and digital communication technologies, CPS are capable of influencing and interacting with the physical and digital world (see Fig. 2.7). In detail, they are able to generate, record, store and evaluate data, as well as to derive, pursue and transmit actions or provide, distribute and communicate information [20].

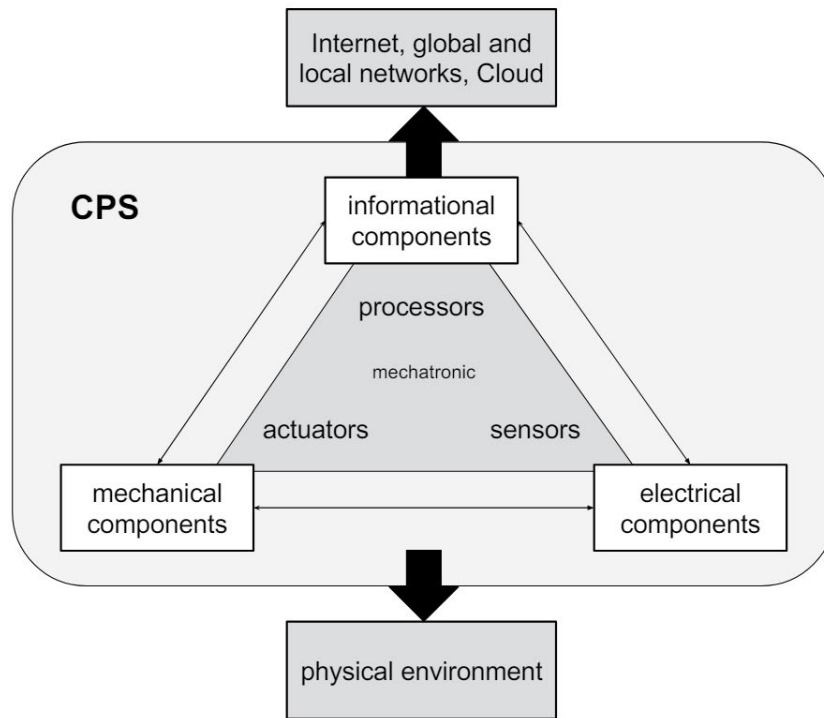


Figure 2.7: Cyber-physical systems (adapted from [21], p. 11)

Generally, CPS consist of mechatronic components and an internet connectivity (cf. [22], p. 112). Actuators convert electrical signals into action, physically affecting the surrounding area. Sensors collect physical data, generated as a result of these actions, or, as a prerequisite to the electrical signals relevant for them. Processors undertake monitoring, processing, and control tasks and additional software and hardware, such as functions and interfaces, enable informational exchange within clouds, global, local, wired or wireless networks, the internet and/or between multiple cyber-physical systems (see Fig. 2.7).

CPS are defined as embedded systems, that, are connected to each other, regardless of the type of interface, provide and use available data and services regardless of their location and entail different options for communication and control in form of human-machine interfaces. All in addition to the ability to generate physical data, influence real processes, process data and derive actions from it (cf. [21], p. 10).

Between CPS, vertical and horizontal communication can be established, which differs from common mechatronic systems (cf. [20], p. 30). Vertical communication refers to communication between multiple layers or levels of automation, control and production hierarchy, such as communication between the one machine, the shopfloor and the entire production

planning level. Horizontal communication refers to communication within one level or hierarchy, such as the communication between two manufacturing steps or production processes. This combination makes a network of CPS flexible, dynamic and accessible (see Fig. 2.8), resulting in increased data transparency and availability (cf. [20], p.30 and [21], p. 11).

All relevant information is available across all production networks and hierarchies in real-time, enabling fast and accurate monitoring, control and plannability. Thanks to the connectivity, changing conditions, errors and quality issues can be identified and mitigated, a synoptic view of the production process can be created, observed and fully documented, enabling increased flexibility, production automation, efficiency and quality [21].

2.5.2 Smart Objects

Smart objects or smart devices are a subspecies or special type of CPS, containing both physical and digital components, but not necessarily actuators or sensors to directly and consciously interact with the physical world. Smart objects consist of informational technology, containing information about themselves, such as their identification number, designation, or remaining useful life, or their state and position in an ongoing production process. The definition of smart objects can be applied to any sort of object, for example smart actuators, smart sensors or smart workpieces, implying that these objects “are self-conscious”, i.e. entail information, about the state, position, or configuration they are currently in, and are able to communicate and provide this information to other participants within a shared network (cf. [20], pp. 31-32). If this applies to an entire factory, it is called Smart Factory.

Smart objects share the same advantages as CPS in terms of connectivity, information/data accessibility as well as resulting efficiency, speed and flexibility within production or regarding the identification and plannability of maintenance measures. A shared network, that can benefit from these advantages, is represented by the Internet of Things and will be explained in the following section:

2.5.3 Internet of Things

The Internet of Things, or IoT, is a network of physical objects equipped with sensors, microchips, appropriate software, and network connectivity, which are connected with each

other and virtually represented within the internet. These objects are uniquely identifiable through individual IP-Addresses and are integrated into the IoT as an extension of the internet, enabling them to communicate with other participants and exchange information.

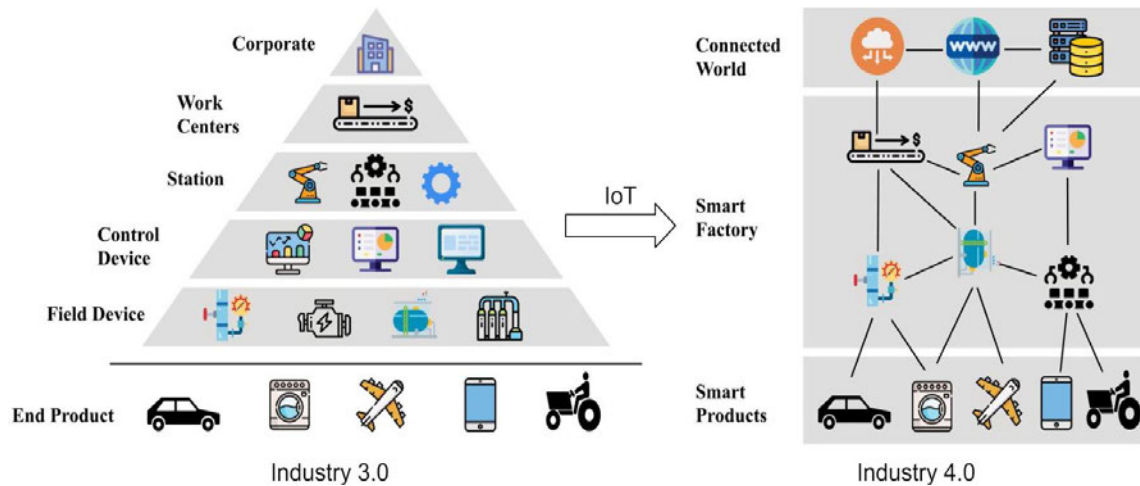


Figure 2.8: Evolution of industrial hierarchies through IoT (adapted from [26])

IoT, within an industrial environment also referred to as IIoT (Industrial Internet of Things), enables manufacturers to fully network their entire production equipment and machinery, leading to increased automation, real-time data transfer and optimization of processes, if wanted, without human intervention (cf. [19], p. 36, [21], p. 9 and [23]). A global multi-platform infrastructure can be established, connecting physical and virtual objects such as multiple CPS, smart objects and computers (see Fig. 2.8), that can enhance automated information distribution and data availability, resulting in improved error detection, failure preventability, process automation and control (cf. [24], pp. 3-6). Human-Machine-Interfaces can also be integrated into the IoT or IIoT, making the data available to interested parties both outside and within the production processes, enabling control, insight, and interaction. This platform differs from the usual hierarchy (Fig. 2.8, left), enabling all participants to communicate with each other vertically and horizontally, as previously described in Section 2.5.1 and illustrated in Fig. 2.8.

Additionally, in terms of services such as condition-based or preventive maintenance, an extended concept of IoT, the Internet of Things and Services (IoTS) has developed, which will be explained in the following section:

2.5.4 Internet of Things and Services

While the focus and main objective of IoT lies on the communication and connection of physical objects within a network to collect and transfer data, the objective of IoTS is to further extend this concept and integrate services within the scope of Industry 4.0 [19].

Within the IoTS, services based on physical objects and/or the data they provide can be connected to other services and objects to extend the automated information distribution and integrate data analysis capabilities. The analyzed data, potentially as a service itself, can be used to enable further automated, intelligent and individual services, such as artificial intelligence or machine learning for predictive maintenance.

Overall, the goal of the IoTS is to create smart and efficient production environments through connectivity and communication between various objects, the ability to process and analyze data, and the ability to derive actions based on individual production needs (cf. [19], p. 36). The collected knowledge can, for example, enable spare part management services to automatically reorder or pre-order spare parts as indicated by condition-based or predictive maintenance systems, considering stock counts and maintenance needs derived from the shared data.

2.5.5 Big Data

An interconnected network of CPS, smart objects and services, such as the IoTS, generates a large and continuously growing amount of raw data. Particularly, data captured steadily over a series of time, known as time series data [25], is an important byproduct of the IoTS due to the continuous monitoring of the entire production plant as well as all related components. The resulting Volume, Variety and Velocity of accumulating data is described by the term “Big Data” (cf. [21], p. 27).

In order to manage and utilize the large amount of data, gain insights and achieve the resulting economic benefits, appropriate databases suited for time series data and analytical tools are necessary to store and process the data in real-time (cf. [21], p. 27 and [20], p. 46). These analytical tools and techniques, in form of machine learning algorithms and neural networks used for predictive maintenance, will be explained in the following sections:

2.6 Machine Learning

Learning is defined as a process based on targeted effort, experience, practice, or observation that leads to any change, often referred to as a relatively permanent change (cf. [29]), or any form of performance improvement (cf. [30]). Machine Learning incorporates methods and techniques that enable a machine or computer to learn autonomously from data and recognize patterns. Instead of being provided with an explicit program or algorithm to find a solution or result based on the embedded knowledge and input-data, the machine or computer is given a set of data and the desired corresponding results, as illustrated in Fig. 2.9 below [31]:

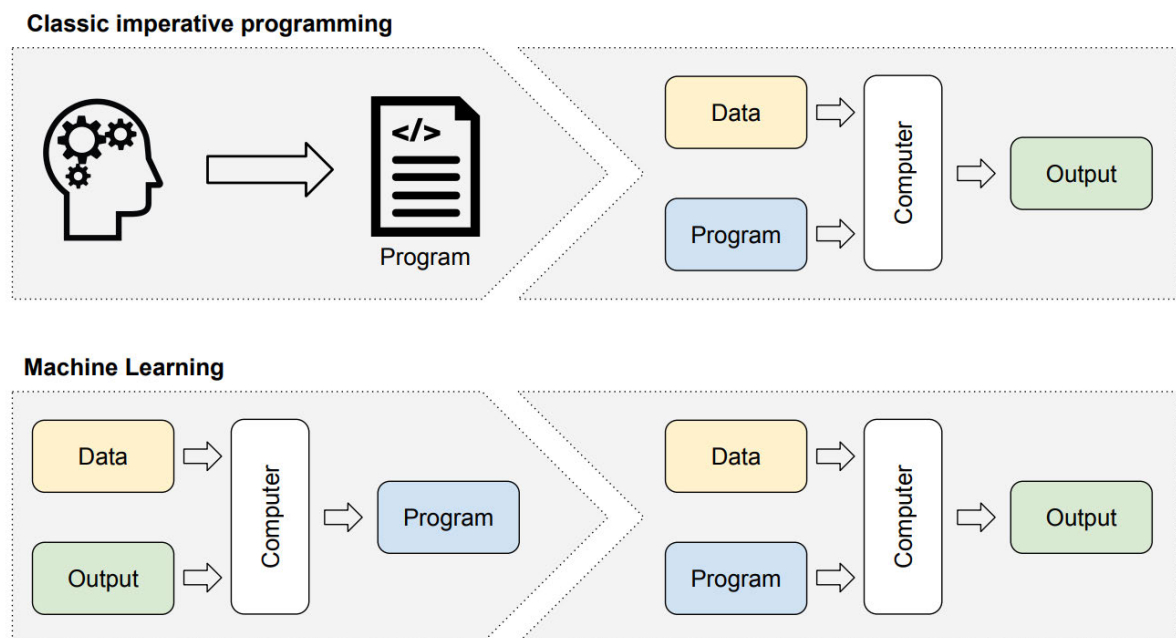


Figure 2.9: Machine Learning vs classic imperative programming (adapted from [31])

Unlike classic imperative programming, where programs are written with step-by-step commands to process data and compute outputs (see Fig 2.9, and [32]), the provision of training data and corresponding desired outputs allows machines or computers using machine learning algorithms to identify relationships and patterns within the data and learn how to compute the desired output, creating their own processing logic (cf. [31], p. 210). Once the learning process has been successfully completed and the machine learning algorithm or system has been trained, it can be used to process new data and analyze it according to the learned logic (see Fig. 2.9).

In Machine Learning, the learning process is defined by specific algorithms, which can be categorized into three different types based on their approaches. These types will be explained in the following section:

2.6.1 Supervised Learning

Supervised Learning is the most common machine learning approach and consists of datasets that already include known output values, referred to as labeled data or labeled datasets. In these datasets, the inputs X and outputs Y are known, but the function f that maps the inputs to the outputs is unknown and needs to be learned by the machine learning algorithm. This relationship can be expressed as (cf. [33], p. 21):

$$f: X \rightarrow Y \quad \text{or} \quad f(X) = Y \quad (1)$$

The labeled data acts as a supervisor or teacher, enabling the machine learning algorithm to learn how inputs and outputs relate to each other and how to distinguish between correct and incorrect labels through a sufficient amount of data. Supervised learning consists of two main categories: classification and regression.

Classification

Classification describes the process of matching inputs, such as images or time series data, to discrete classes or categories. The objective of a classifier is to determine the class of an unknown input from a known set of classes. For example, it can be used to distinguish between e-mails and spam-mails or to categorize pictures of domestic animals into classes such as cats, dogs, birds, and fishes. Classification can be further divided into the following three subcategories:

1. Binary Classification: If the algorithm only needs to distinguish between two categories, such as e-mails and spam-mails, it is called binary classification.
2. Multi-Class Classification: If the classification task consists of three or more classes, such as differentiating between pictures of cats, dogs, birds, and fishes, it is referred to as multi-class classification.
3. Multi-Label Classification: If a single input can be matched to more than one category at once, such as a picture containing both a cat and a dog or a bird, cat and a fish, it is called multi-label classification (cf. [33], p. 203).

In general, the classification-problem can be described as following in accordance with Eqn. (1):

$$c: X \rightarrow C \quad (2)$$

with c being an unknown classification function that matches a set of features X to a set of classes C , and given a known subset of labeled data D (cf. [33], p. 23):

$$D = \{(x_1, c(x_1)), (x_2, c(x_2)), \dots, (x_n, c(x_n))\} \subseteq X \times C \quad (3)$$

Depending on the classification problem, C can be represented as follows:

$$1. \ C = \{0, 1\} \text{ or } \{-1, +1\} \quad \text{for binary classification} \quad (4)$$

$$2. \ C = \{1, \dots, k\} \quad \text{for multi-class classification} \quad (5)$$

$$3. \ C = \{-1, +1\}^k \quad \text{for multi-label classification} \quad (6)$$

with 3, where positive components display membership to a class (cf. [35], p. 11 and p. 183). The classification problem is solved by obtaining the function c through supervised learning with the labeled dataset D (cf. [33], p. 23).

Regression

Regression works similarly to classification, where a set of inputs X and outputs Y are provided. However, the given labels don't represent classes, but instead numerical values, with:

$$X \subseteq \mathbb{R}^n \quad \text{and} \quad Y \subseteq \mathbb{R}^n, \quad (7)$$

and

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^n, y^n)\} \subseteq X \times Y \quad (8)$$

as the provided dataset (cf. [33], p. 23). Regression involves predicting numerical values, such as the price of a domestic cat, based on its specific features. These features, such as the breed, age, or fur and eye color of the cat, are also referred to as predictors. The learning process results in the same function as in Eqn. (1), with the ability to predict numerical values Y based on inputs X . Regression algorithms can also be used for classification problems, by providing a probability measure of belonging to a specific class or category (cf. [33], p. 23; [34], pp. 10 and 295). However, since regression algorithms predict a numerical value, their success is not evaluated based on whether they correctly predict a class, but rather on how close the prediction is to the actual or desired value (cf. [33], p. 23 and [35], p. 2).

In terms of predictive maintenance, classification algorithms allow to differentiate between healthy and unhealthy states of a machine or piece of equipment, such as a broken drill bit holder, a broken drill bit, or a blunt drill bit due to a development of temperatures or vibrations. Regression algorithms, on the other hand, can estimate specific numeric values, such as a specific temperature, degree of degradation or probability of failure. Two supervised learning algorithms that will be discussed within the scope of this thesis are the support vector machine (see Section 2.6.5) and neural network (see Section 2.6.7).

2.6.2 Unsupervised Learning

As the name indicates, unsupervised learning does not receive a teacher or supervision in the form of labeled data. Instead, the computer or algorithm tries to learn solely from unlabeled data without prior knowledge of the output.

Unsupervised learning is used to find unknown similarities, distributions, or structures within data. It is therefore applied in analytic tasks such as clustering and determining of distributions or associations, all based on the nature, characteristics, and features of the data itself (cf. [31], p. 235, [33], pp. 25-27 and [34], pp. 11-14). For example, it could cluster cats, dogs, birds, and fishes into two groups based on whether they live above or under water, or cluster them into three groups based on the number of legs the animals possess.

Another use-case of unsupervised learning is anomaly or outlier detection. The algorithm is trained on normal or ordinary data in order to perform a so-called one-class or unsupervised classification. It learns to recognize the characteristics and features of the data it has been trained with and uses that knowledge to identify if new data belongs to the same group or not (see Fig. 2.10). The goal is to identify unwanted issues, such as equipment faults or malfunctions, production errors, signs of fraud or medical issues, by questions such as: “Does the equipment operate as intended?”, “Is the patients CT scan unremarkable?” or “Do the last transactions correspond to the previous purchasing behavior?” (cf. [31], p. 235; [33], pp. 25-27; [34], pp. 11-14 and [40]).

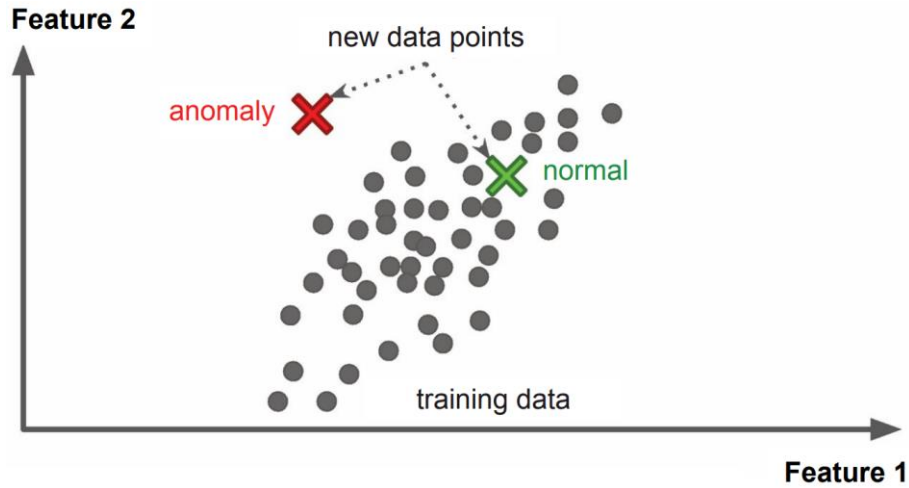


Figure 2.10: Identification of anomalies (adapted from [34], p. 14)

Unsupervised learning algorithms that will be discussed within this thesis are the one-class support vector machine (see Section 2.6.5), the isolation forest (see Section 2.6.6), and a neural network in form of an autoencoder (see Section 2.6.8).

2.6.3 Reinforcement Learning

Reinforcement Learning differs from the other two learning processes. It entails a so-called software agent that develops a strategy by performing actions based on an observed environment or a given situation. These actions receive feedback in the form of rewards or punishments, also called negative rewards, which in turn lead to an adaptation and change of the strategy. This is an iterative process, where the system needs to find the optimal strategy on its own, in an effort to maximize the number of positive rewards (cf. [31], p. 307; [33], p. 24; [34], p. 15; [35], p. 8 and [36], p. 3). Common examples of reinforcement learning are teaching machine learning algorithms to play a board game, where the strategy that leads to a win, as the ultimate reward, is learned by playing and analyzing previous strategies (cf. [34], p. 16 and [36], p. 3).

2.6.4 Machine Learning Algorithms

A machine learning algorithm is a specific mathematical or statistical process used to learn from data and create a corresponding machine learning model [49]. It determines how the data is processed, analyzed, and optimized in order to recognize patterns and perform predictions. The learning process is guided by the previously described learning techniques,

depending on the category and desired use case of the machine learning algorithm. In the following sections, machine learning algorithms in the form of the support vector machine, isolation forest and neural network are further explained:

2.6.5 Support Vector Machines

Support vector machines, or short SVMs, are machine learning algorithms that are able to solve both linear and nonlinear classification problems and can be used for unsupervised classification, such as anomaly detection, as well as regression tasks [34]. The goal is to find a hyperplane that can separate a given n -dimensional feature space into two classes, ensuring that the datapoints of one class lie on one side of the hyperplane, while the data points of the other class lie on the opposite side (see Fig. 2.13 and cf. [37]).

Linear classification

Given data points in a feature space, the challenge is to find an optimal hyperplane that separates two groups of data for accurate classification. Figure 2.11 shows two groups of data points (red and blue) that can be separated by a straight line, representing a hyperplane in a two-dimensional space.

For the purposes of demonstration and generalization, upcoming Figures (see Fig. 2.11 to 2.13) do not include class, feature, or axis descriptions.

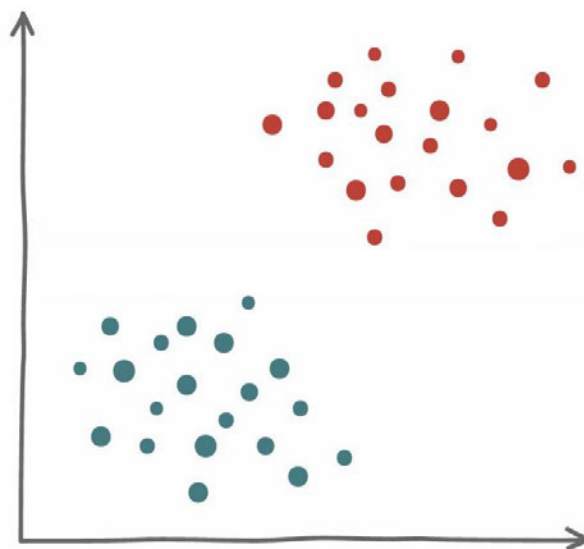


Figure 2.11: Two groups of data points in a feature space (adapted from [38])

Visually, the space can be separated by multiple instances of hyperplanes to classify the data accordingly (see Fig. 2.11). Figures 2.12 and 2.13 show four examples and one exception of these different possibilities. A support vector machine needs to determine the optimal hyperplane out of infinite possibilities in order to separate future inputs of unknown data successfully and accurately. The following describes the structure and reasoning behind the hyperplane of a SVM for linear classification.

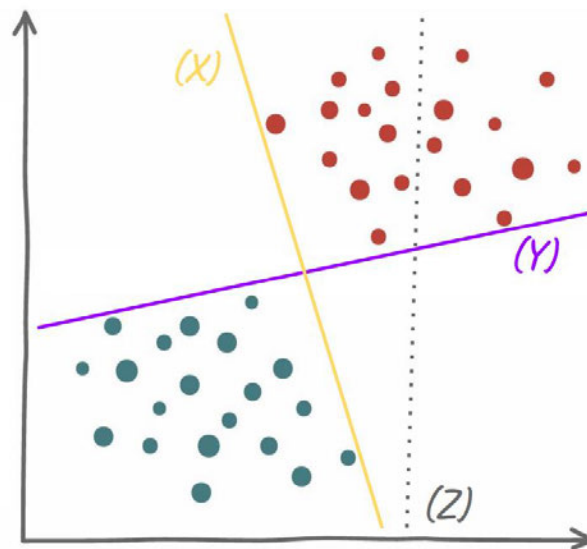


Figure 2.12: Example of insufficient hyperplanes (adapted from [38])

Figure 2.12 shows three hyperplanes of linear classifiers, (X), (Y) and (Z). Classifier (Z) is hereby not able to separate the data points correctly, due to the location of its hyperplane. The hyperplanes of classifiers (X) and (Y) can both separate the data entirely, but are located closely to data points of either group, which indicates that they won't perform well on new data (cf. [34], p. 155). For example, data points on the opposite side of the hyperplane, but in close proximity to the group they belong to, would be misclassified.

Figure 2.13 shows this instance with hyperplanes (A) and (B), which have a greater distance to the groups of data points in comparison to hyperplanes (X) and (Y).

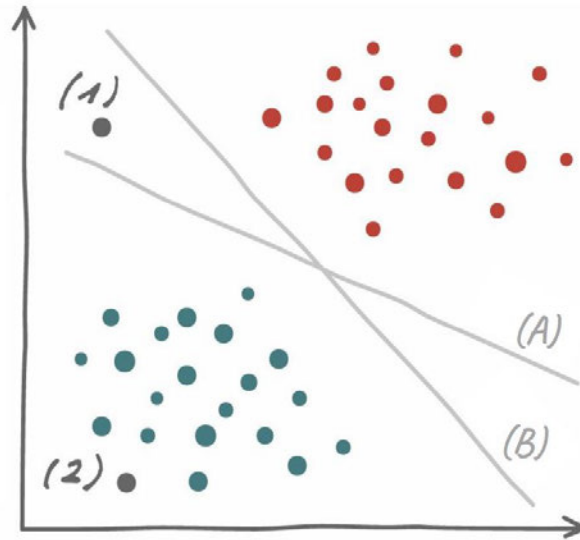


Figure 2.13: Hyperplanes with additional datapoints (adapted from [38])

Further, it shows two additional datapoints, (1) and (2). Depending on how the hyperplane is located or tilted (see Fig. 2.13, (A) and (B)), point (1) is classified as green or red data points. Point (2) however, which has the greatest distance to the hyperplanes, can be matched safely to the class of green data points. The greater the distance of the data points is to the hyperplane, the safer is the classification (see Fig. 2.13 and 2.14; cf. [38] and [39]).

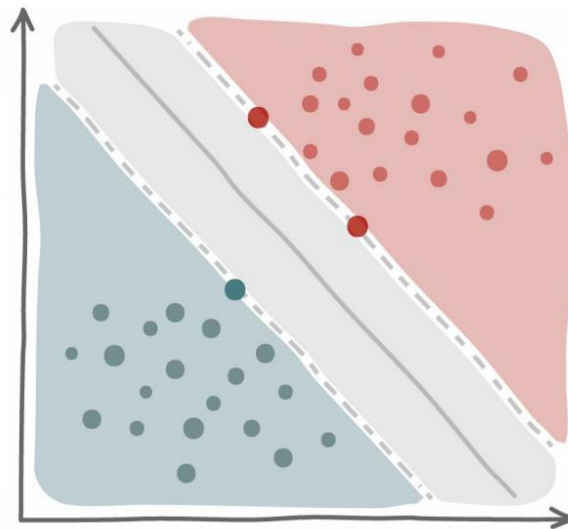


Figure 2.14: Optimal hyperplane with support vectors (adapted from [38])

Figure 2.14 shows the optimal hyperplane of a SVM, maximizing the distance to the closest datapoints of each group. The goal of a SVM is to find this hyperplane by maximizing the mentioned distance, also known as margin. The closest data points to the hyperplane,

highlighted in Fig. 2.14, are referred to as support vectors. These support vectors define the marginal hyperplanes, visualized by the dashed lines, as well as the hyperplane itself (see Fig. 2.14, [33], [34], [35] and [38]).

In general, a hyperplane is defined as follows (cf. [35], p. 64):

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad (9)$$

It represents a separating or decision boundary without intersecting datapoints of a given set of training data D (see Eqn. (3) and Fig. 2.14). Therefore, b and \mathbf{w} can be scaled, so that the distance from the hyperplane to the closest data point in the training data is 1 (cf. [35], p. 64 f.):

$$\min_{(\mathbf{x}, c) \in D} |\mathbf{w} \cdot \mathbf{x} + b| = 1 \quad (10)$$

The marginal hyperplanes, which run parallel to the hyperplane (see Fig. 2.14), possess the same normal vector \mathbf{w} and intersect the support vectors as the closest data points to the hyperplane, can therefore be described as:

$$\mathbf{w} \cdot \mathbf{x} + b = \pm 1 \quad (11)$$

with

$$\mathbf{w} \cdot \mathbf{x} + b = +1 \quad \text{for the nearest positive points}$$

and

$$\mathbf{w} \cdot \mathbf{x} + b = -1 \quad \text{for the nearest negative points}$$

relative to the hyperplane (cf. [35], p. 65). With a training dataset that entails labeled outputs $C = \{-1, +1\}$ representing each class for binary classification (see Section 2.6.1 and Eqn. (4)), a classification of a data point \mathbf{x}_i is successful when the expression on the left side of Eqn. (11) matches the sign of c_i (cf. [35], p. 65).

With ρ as the definition of the margin (cf. [35], p. 65)

$$\rho = \min_{(\mathbf{x}, c) \in D} \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}, \quad (12)$$

a maximized margin and therefore a solution to the classification problem, can be achieved by “minimizing $\|\mathbf{w}\|$ or $\frac{1}{2} \|\mathbf{w}\|^2$ ” ([35], p. 65) as follows:

$$\min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2 \quad (13)$$

$$\text{subject to: } c_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i \in [1, m],$$

under the condition, that all data points are classified properly (cf. [35], p. 65).

The training data of an SVM algorithm serves as the basis for determining the hyperplane of a classification model. Hereby, only the data points within the training data, that act as the support vectors, influence the outcome of the learning process. Therefore, altering, adding or removing data points behind the marginal hyperplanes has no effect on the outcome. A change in the support vectors on the other hand, can impact the entire outcome of the resulting model (cf. [33], p. 408 and [38], p. 100).

A classifier with a large margin reduces the risk of misclassification for new data points that might be near the hyperplane and enhances its ability to generalize (cf. [34], p. 156 and [39]). A SVM trained for linear classification while maximizing its margin and correctly classifying all training data, as illustrated in Fig. 2.14 and Eqn. (13), is referred to as hard-margin-classifier or, in general, as hard-margin-classification. However, in cases where the training data contains outliers and/or the feature space is not linearly separable (see Fig. 2.15), resulting in either a very narrow margin or a feature space that cannot be separated by a hyperplane, a flexible model allowing exceptions within the marginal hyperplanes can be used, known as a soft-margin-classifier or soft-margin-classification (cf. [34], p. 156 f.).

Soft-margin-classification

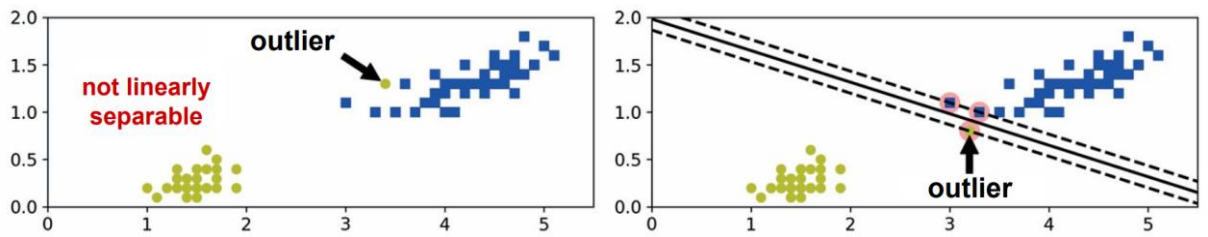


Figure 2.15: Hard-margin-classification and outliers (adapted from [34], p. 157)

Figure 2.15 shows an example of training data, that includes outliers and therefore limits the size of the margin (Fig. 2.15, right) or cannot be separated through hard-margin-classification (Fig. 2.15, left). In this case, a slack variable ξ_i is introduced, that allows exceptions

within as well as outside of the margin and includes the distance of a point x_i to its according marginal hyperplane (see Fig. 2.16). Hereby, all data points x_i with $\xi_i > 0$ are considered outliers, as illustrated below (cf. [35], p. 71).

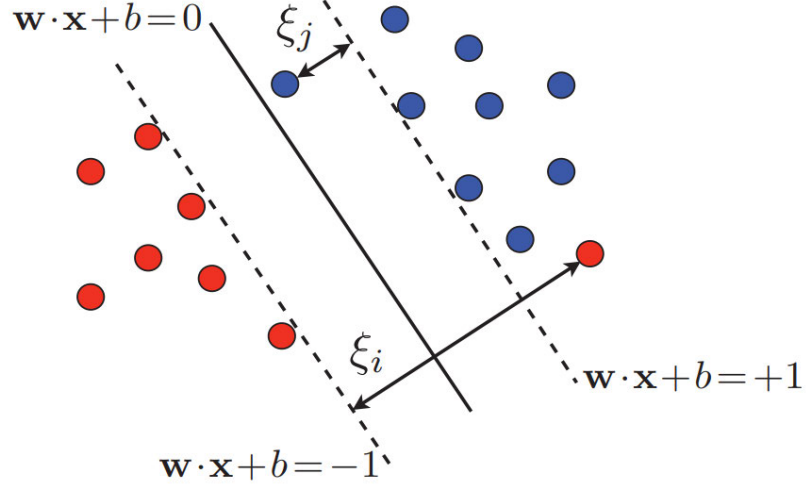


Figure 2.16: Soft-margin-classification (adapted from [35], p. 71)

The classification problem is solved through a compromise between minimizing the allowed sum of exceptions and maximizing the margin (cf. [33], p. 411; [34], p. 157; [35], p. 71 and [36], p. 332):

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i \quad (14)$$

$$\text{subject to: } c_i(w \cdot x_i + b) \geq 1 - \xi_i \wedge \xi_i \geq 0, i \in [1, m]$$

with an adjustable parameter $C \geq 0$, that influences the number of allowed exceptions.

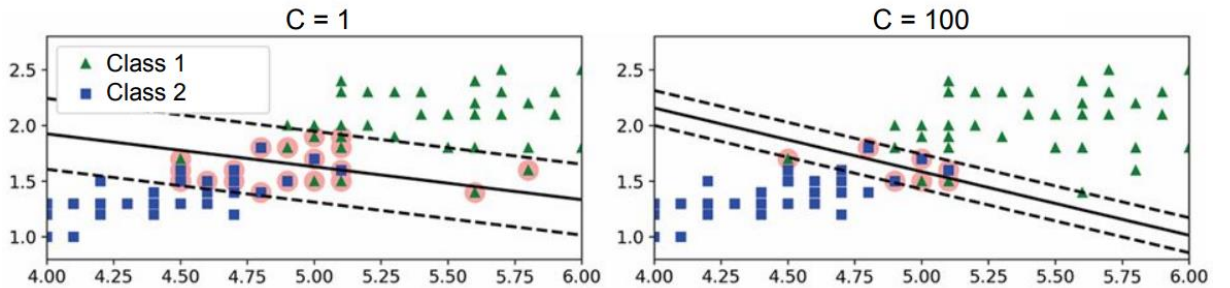


Figure 2.17: Soft-margin-classification and parameter C (adapted from [34], p. 157)

Figure 2.17 shows the margin of a classifier with $C = 100$, allowing few exceptions and narrowing the margin (Fig. 2.17, right) and $C = 1$, allowing more exceptions with a wider

margin (Fig. 2.17, left). This method is more applicable to real circumstances, as datasets mostly subject to non-ideal conditions and contain outliers that prevent a linear separation through a straight line or hard-margin-classification.

Non-linear classification

Aside from outliers, in practical applications, datasets often consist of or include non-linear data that cannot be divided into two classes through linear separation, as illustrated by the left diagram in Fig. 2.18 below:

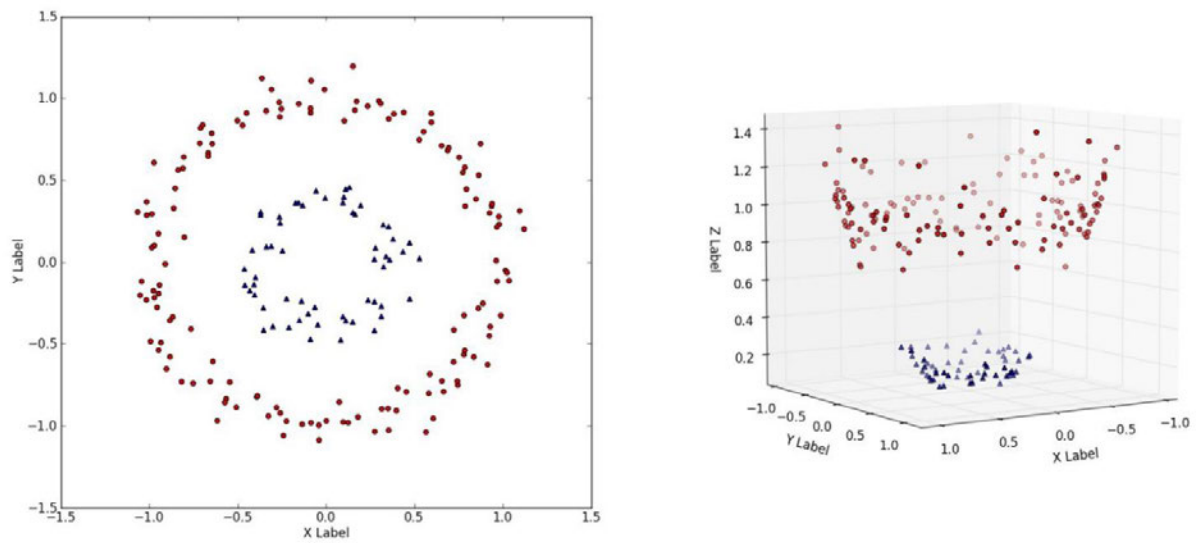


Figure 2.18: Adding another feature dimension (adapted from [40])

By extending the feature dimension, here for example, by adding polynomial features and transforming the two-dimensional feature space into a three-dimensional feature space (see Fig. 2.18), i.e. transferring the data into another coordinate system of a higher dimension (cf. [38], p. 103), a distribution of data points within a dataset can be created, that enables linear separation with a hyperplane and therefore the application of SVMs (see Fig 2.18, right and Fig. 2.19, left).

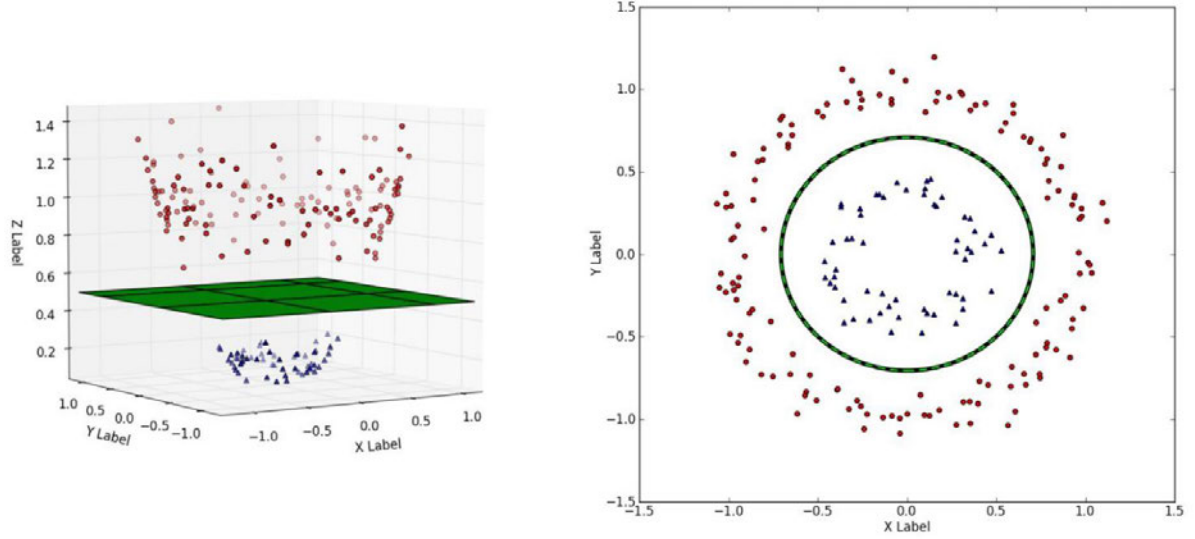


Figure 2.19: Separation by hyperplane and back transformation (adapted from [40])

After transforming the data and separating the feature space with a linear hyperplane, as demonstrated by the left diagram in Fig. 2.19, the data along with the hyperplane are transformed back into their original n -dimensional feature space, separating the data through a non-linear boundary (see Fig. 2.19, right). Using this boundary, non-linear classification of new data points can be performed (cf. [34], p. 159 and [38], pp. 102-103). This approach, utilized by SVMs, is referred to as the so called “kernel trick” (cf. [38], p.101) and will be explained in the following:

A function ϕ that transforms an input space X to a higher feature space \mathbb{H} can be depicted as (cf. [35], p. 91):

$$\phi: X \rightarrow \mathbb{H} \quad (15)$$

A kernel is a function K that, for any two points x_1 and x_2 as elements of the input space X , can compute the inner product of vectors $\phi(x_1)$ and $\phi(x_2)$, without transforming them, under the condition that ϕ exists (cf. [34], p. 171 ff. and [35], pp. 89 ff.):

$$\forall x_1, x_2 \in X, K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle = \phi(x_1)^T \phi(x_2) \quad (16)$$

The output of the kernel K hereby represents the measure of similarity of two points within X , equivalent to the inner product of the vectors (cf. [35], p. 91). The kernel trick is therefore computationally efficient, as all points x_1 and x_2 in X (i.e., the input space) can be used

without computationally entering a higher feature dimension (cf. [33], p. 415; [34], p. 172 and [35], p. 91).

The following kernels are generally used for non-linear classification with SVM (cf. [34], p. 172 and [37]):

Table 2.1: Generally used kernels

SVM	Kernel
Linear	$K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{x}_2$
Polynomial	$K(\mathbf{x}_1, \mathbf{x}_2) = (\gamma \mathbf{x}_1^T \mathbf{x}_2 + r)^p$
Gaussian Radial Basis Function	$K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\gamma \ \mathbf{x}_1 - \mathbf{x}_2\ ^2)$
Sigmoid	$K(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma \mathbf{x}_1^T \mathbf{x}_2 + r)$

One-class classification

One-class-SVMs for anomaly detection function similarly to SVMs utilizing kernel functions, which were described in the previous section. In this instance, instead of separating two classes with a non-linear hyperplane, the hyperplane aims to encapsulate the training dataset, leading to any future data point on the opposite side of the hyperplane being detected or classified as an anomaly (cf. [34], pp. 276-277).

Multi-class classification

SVMs are binary classifiers with the directive to distinguish between two classes (see (4)). To use SVMs for multi-class classification tasks, multiple instances of SVMs need to be trained in order to perform one of two strategies:

- **One-vs-one:** This method trains $n = \frac{k(k-1)}{2}$ SVMs to distinguish between every pair of k classes (see Eqn. (5)), where each classification counts as a win for the corresponding class. The class with the highest number of wins is chosen as the final class for the data point.
- **One-vs-all:** In this strategy, k classifiers for k classes are trained, each used to distinguish between one class and all other classes combined. During classification, the distance of the corresponding data point to the hyperplane is determined for each

classifier. The class with the greatest distance between the data point and the hyperplane is chosen as the final class for that data point.

One-vs-all is considered less accurate, when an imbalanced distribution of data points across all classes exists, as the decision is based on the greatest distance of a data point to the hyperplane. However, except when $k = 3$, one-vs-one requires more computational resources due to the higher number of classifiers that need to be trained, compared to one-vs-all (cf. [33], pp. 416-417). All in all, both strategies, along with all previous SVM-related classification techniques combined, enable SVMs to be utilized for a wide range of machine learning applications.

2.6.6 Isolation Forest

The isolation forest, short IF or iForest, is a machine learning algorithm primarily used for anomaly detection and works on the premise that anomalies are easier to isolate than their normal counterparts.

The isolation forest, comparable to a real or natural forest, consists of a community of trees, referred to as iTrees. Based on the premise that “anomalies are ‘few and different’” ([41], p. 414), the iTrees, generated randomly, recursively separate a given dataset by choosing a random feature and a random threshold, until all datapoints are isolated (see Fig. 2.20).

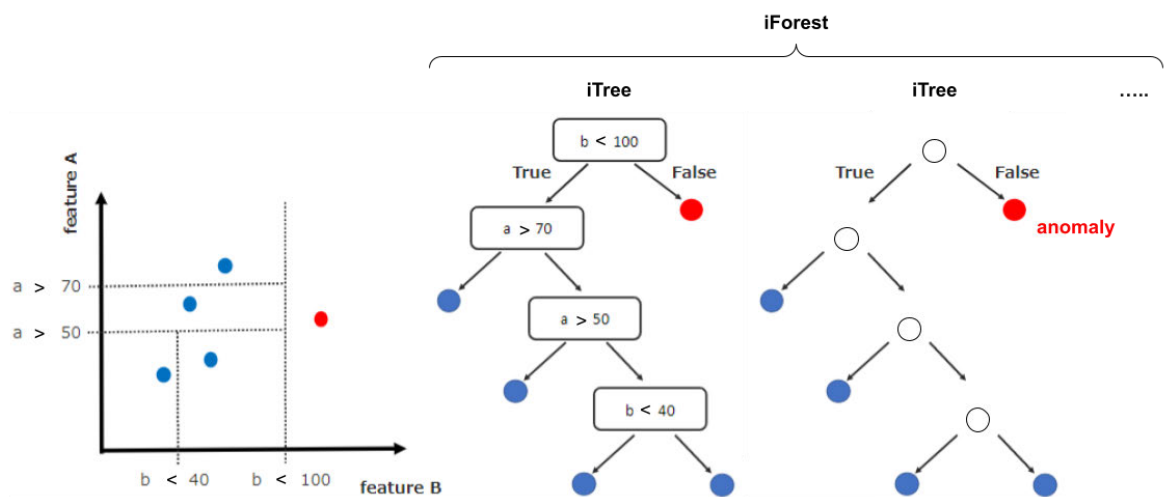


Figure 2.20: Visualization of an iForest (adapted from [42])

The datapoints with a closer proximity to the root or origin of the iTree, as illustrated in Fig. 2.20, are classified as anomalies, since their isolation can be achieved with fewer cuts or separations, representing a shorter path length within its structure, compared to data points considered to be normal (see Fig. 2.20, left and right).

The iTree hereby functions as a binary tree, where each node T possesses either two or zero daughter nodes T_l and T_r (left and right). Each tree receives a sample of data $X = \{x_1, \dots, x_n\}$ to grow, by randomly picking a feature q as well as a threshold p and dividing the data points by $q < p$ (see Fig. 2.20). This process is repeated until one of the following conditions is met: X consists of one remaining data point, X consists of data points that all share the same value, or the trees height limit is met (cf. [41], p. 415). The iForest model is trained when a community of iTrees is grown. After training, the iForest is tested by moving a labeled test dataset through the iTrees, determining a corresponding anomaly score for each data point [41]. Anomalies are indicated by an anomaly score very close to 1, representing a high certainty. A data point is considered normal with a score much smaller than 0.5. If an anomaly score of $s \approx 0,5$ applies to all data points of a dataset, it suggests that no individual data point stands out as an anomaly (cf. [41], p. 415). The anomaly score s can be obtained through:

$$s(x, n) = 2^{\frac{E(h(x))}{c(n)}} \quad (17)$$

with $h(x)$ as the number of edges a data point x passes through, corresponding to the path length used to differentiate between anomalies and normal data points, and $E(h(x))$ as its average across the iTrees. $c(n)$ represents the “average path length of unsuccessful search” ([41], p. 415), equivalent to the “average $h(x)$ for external node terminations” ([41], p. 415):

$$c(n) = 2H(n - 1) - (2(n - 1)/n) \quad (18)$$

with n as the number of external nodes, i.e., nodes that have zero daughter nodes, and the harmonic number $H(i)$, estimated by Eulers constant $\ln(i) + 0.5772156649$. $n - 1$ represents the number of internal nodes, dividing the datapoints as illustrated in Fig 2.20, shown as white circles and rounded rectangles, resulting in:

$$2n - 1 \quad (19)$$

as the overall number of nodes in an iTree [41]. The iForest possesses two hyperparameters: the number of iTrees t and the subsampling size ψ , which in turn determines the height limit l of an iTree automatically:

$$l = \text{ceiling}(\log_2 \psi), \quad (20)$$

The ceiling function hereby returns the nearest integer rounded up. All trees within the total of t trees in the iForest receive the set number of random subsamples ψ without replacement as their input during training. Empirically, the initial paper [41] proposes $t = 100$ and $\psi = 256$ as good values for a wide range of applications. Through subsampling, several trees within the entirety of the iForest identify various anomalies, as each subsample differs in its composition. Hereby, the isolation of every individual, and generally mostly normal, data point within a dataset is neither desired nor necessary (cf. [41], p. 416), as “anomalies are ‘few and different’” ([41], p. 414) and therefore easier to isolate through their shorter path length, resulting in the tree height limit.

In summary, iForests consist of a fixed number of iTrees that grow proportionally with the dataset size. By utilizing subsampling and implementing a tree height limit, iForests randomly split data points to isolate and identify anomalies through partial models and their respective path lengths. This allows them to handle large datasets efficiently, as the memory usage grows linearly with the overall number of nodes (see equation (19)). Additionally, iForests can effectively detect anomalies since each model operates on a small sample size, enhancing the visibility of anomalies and making the algorithm applicable to a wide range of data compositions [41].

2.6.7 Artificial Neural networks

Artificial neural networks, also referred to as ANN, neural networks or NN, are networks consisting of interconnected artificial neurons inspired by the human brain. Neural networks are a subset of machine learning, used for automatic information and data processing. The foundation, structure and learning process of neural networks will be explained in the following paragraphs:

As the name indicates, the foundation of a neural network are its neurons. The neurons are organized in layers and every neural network generally consists of three different types of

layers: one input and one output layer as well as one or multiple layers between them, referred to as hidden layers. Every layer consists of multiple neurons and is connected to neurons of another layer (see Fig. 2.23). One exception of the described structure is represented by the so-called perceptron, a neural network consisting of a single neuron, which will be briefly explained in the following to describe its basic functionality:

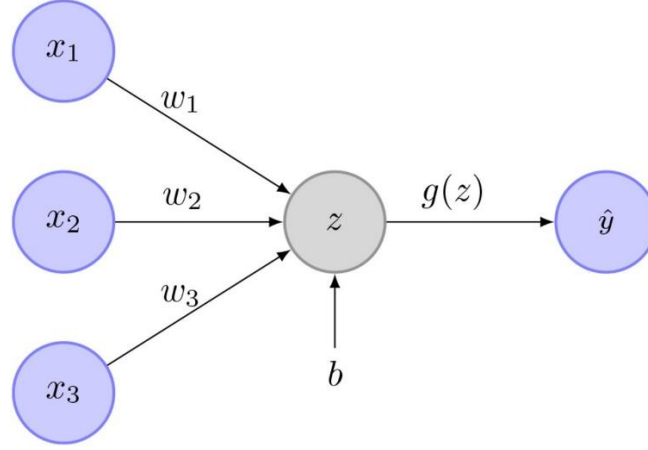


Figure 2.21: Perceptron [43]

Fig. 2.21 shows an example of a perceptron, consisting of a single neuron z . The perceptron receives three inputs, x_1, x_2 and x_3 , which are multiplied by their respective weights w_1, w_2 and w_3 . The neuron z adds up all the weighted inputs and includes a bias b . $g(z)$ represents the activation function, which describes how the neuron reacts to the weighted input signal with the output signal \hat{y} as follows:

$$\hat{y} = g(\mathbf{w} \cdot \mathbf{x} + b) \quad (21)$$

In the case of a perceptron, the output is defined by a threshold θ , making the activation function a step function, also known as Heaviside function, where:

$$\hat{y} = \begin{cases} 1, & \text{if } \sum_{i=1}^n \mathbf{w}_i \mathbf{x}_i + b > 0 \\ 0, & \text{if } \sum_{i=1}^n \mathbf{w}_i \mathbf{x}_i + b \leq 0 \end{cases} \quad (22)$$

with $b = -\theta$ (cf. [33], pp. 169-170; [34], p. 286; [43]; [44], p. 19 and [45], p. 13). Perceptron's are used for binary classification of linearly separable data points [33], similar to binary SVMs (see Section 2.6.5). The perceptron learns by randomly initializing its weights

and bias, then optimizing them in order to achieve the desired outputs. During training the weights and bias are updated, based on the errors in predictions. A perceptron is trained, when it finalizes the weights and bias information, either by correctly classifying all training examples or reaching a specified number of iterations (cf. [33], [34], [38] and [44], p. 22).

Other popular and widely used activation functions for neural networks are depicted in Fig. 2.22 below (cf. [34] and [46]):

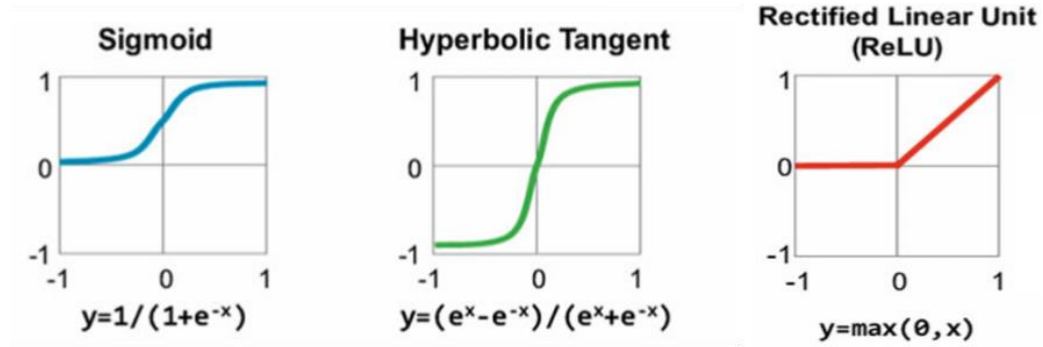


Figure 2.22: Sigmoid, tanh and ReLU activation function (adapted from [46], p. 36)

A sequence of neurons is essentially a sequence of linear transformations. Without nonlinearities, introduced by activation functions $g(z)$ within a neural network, it is unable to recognize or learn complex and nonlinear relationships (cf. [34], p. 293 and [51], p. 72), as illustrated by Eqn. (21). The sigmoid function does not represent a sudden change in its output depending on a threshold. Instead, it represents a continuous and differentiable function, where small changes in the weights w_i and bias b result in small changes in the output \hat{y} (cf. [34] and [50]). The hyperbolic tangent, also known as the tanh activation function, has a similar S-shaped form, but ranges from -1 to 1 . This allows outputs to be centered around 0 at the beginning of the training, as well as the production of negative outputs in general. However, both possess saturation at their highest and lowest output values [33] [34].

The ReLU activation function, short for rectified linear unit, computes the value x it receives, provided it is greater than 0, and returns a zero for inputs ≤ 0 , therefore not differentiating between 0 and negative values. The function is continuous and does not possess saturation for inputs > 0 , and, due to its linearity, is fast to compute. It is therefore considered the most popular among activation functions within neural networks, as it offers good results in practice (cf. [34], p. 293; [50] and [51], p. 72). Further functions, as well as an additional

activation function, are explained in the context of training neural networks, after an overview of their structure:

The neurons of a neural network function analogously to the neurons of a perceptron. A neural network consists of multiple interconnected neurons organized into layers. Each layer of neurons is connected to the subsequent layer, with the output of the previous layer serving as the input for the following. Neural networks can contain any desired number n of neurons as well as any number l of hidden layers in order to fulfill their desired purpose, if technically and computationally possible (cf. [43] and [44]). Networks where all neurons in one layer are connected to all neurons of the following layer are called fully connected networks, as shown in Fig. 2.23:

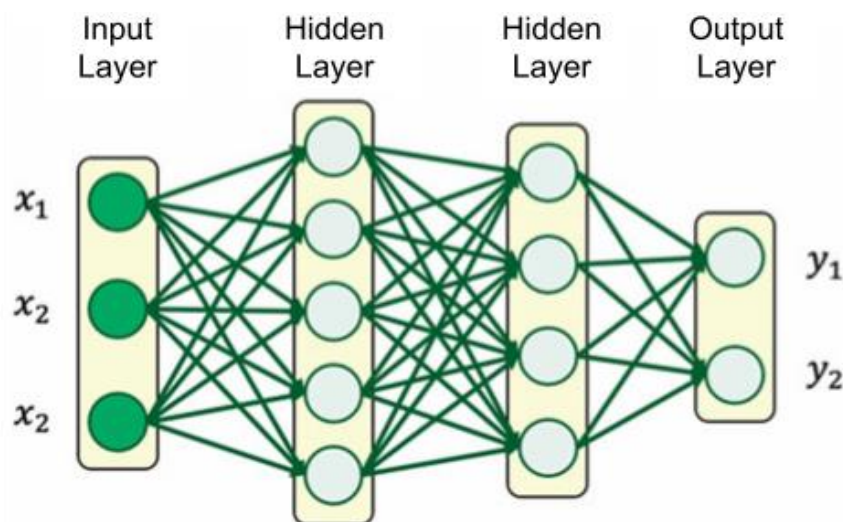


Figure 2.23: Fully connected feedforward neural network (adapted from [46])

Figure 2.23 shows a fully connected neural network with one input layer, two hidden layers as well as an output layer. Based on the direction of connections between neurons and layers, neural networks can be classified as either feedforward neural networks (FNNs) or recurrent neural networks (RNNs). Both types can be either fully connected or not.

The network shown in Fig. 2.23 is an example of a feed forward neural network, where all connections, starting from the input layer, are directed towards the output layer. If a network has connections within its layers or feedback loops from one layer to a previous layer, it is referred to as a recurrent neural network. RNNs can obtain and consider previous processing states and are able to learn relationships, patterns and dependencies within sequential data, as text or audio data, but in return, are more complex to train (cf. [17]; [31], pp. 230-233 and

[44]). FNNs are the most common and widely used architecture, as they possess efficient training methods, a wide range of applications and are proven to be able to approximate any function or behavior, given a hidden layer with sigmoid activation functions and a finite number of neurons (cf. [17], [44], [47] and [48]).

Training of neural networks

In order to train a neural network, aside from specifying the number of hidden layers, their number of neurons as well as their respective activation functions, as the number of input and output neurons is often given by the task or classification problem automatically (cf. [34], p. 327), a loss function and an optimization function need to be defined.

Loss function

The loss function, also known as the cost function, compares the desired output of a neural network, using the labeled training data, to its predictions. It measures the corresponding cost, error, or distance to the desired output, which will be used to minimize the discrepancy during optimization [47]. The loss function of a perceptron, for example, can be defined as the difference between the desired output and the predicted output, either 1 or 0. The classification is optimized by adjusting the perceptron's weights and bias if the loss function's output is not zero [44]. Commonly used loss functions are the mean squared error (MSE) and the cross-entropy (cf. [33], p. 240).

The MSE is used for both regression and classification tasks, but commonly for regression (cf. [33] and [51], p. 60). It measures the mean squared error between the desired values y and predicted values y_p of N samples as follows (cf. [33] and [47]):

$$\frac{1}{N} \sum_{n=1}^N \sum_i \left(y_i^{(n)} - y_{p_i}^{(n)} \right)^2 \quad (23)$$

Cross-entropy is the default activation function used for classification (cf. [33], [43] and [51], p. 60). It measures the difference or degree of deviation between the probability distributions of y and y_p over N . It is defined as:

$$\frac{1}{N} \sum_{n=1}^N D\left(y^{(n)}, y_p^{(n)}\right) \quad \text{where} \quad D(y, y_p) = - \sum_i y_i \log(y_p) \quad (24)$$

for multi-class classification, also referred to as categorical cross-entropy, and:

$$-\frac{1}{N} \sum_{n=1}^N y^{(n)} \log(y_{p_i}^{(n)}) + (1 - y^{(n)}) \log(1 - y_{p_i}^{(n)}) \quad (25)$$

for binary cross-entropy [33]. In order to use cross-entropy, the softmax activation function is required (cf. [33], p. 240)). This activation function is solely used for classification within the output layer of a neural network. Its output represents the probabilities of belonging to one of k classes, where the sum of all output values, i.e., the probability scores, is 1. It is defined as:

$$\sigma_j(z) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (26)$$

for FNNs and $j = 1 \dots n$ [33]. The output with the highest probability score, represents the class to which the input most likely belongs. For multi-class classification problems, categorical cross-entropy combined with a softmax output layer is the commonly proposed solution (cf. [51], pp. 60 and 84).

Optimization function

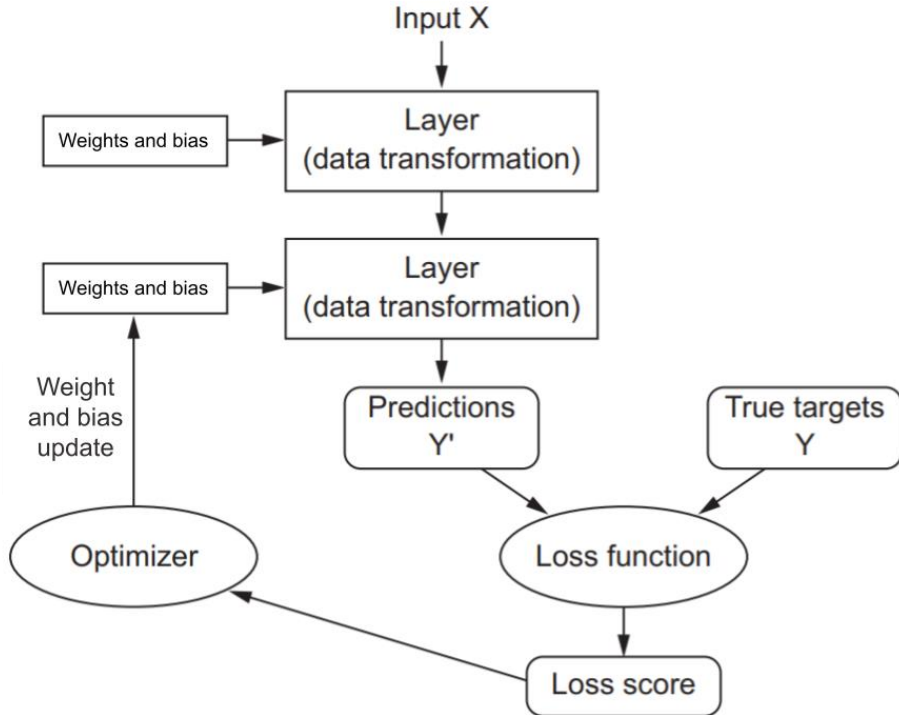


Figure 2.24: Optimizing through loss score (adapted from [51], p. 11)

Figure 2.24 illustrates the learning process of a neural network through the loss function and an optimizer. The optimizer updates the parameters (weights and biases), which are initialized randomly, based on feedback from the loss function, aiming to minimize the loss score (see Fig. 2.24). This process is achieved through gradient descent in combination with back-propagation, two fundamental and complementary techniques used to train and optimize neural networks, which will be explained in the following:

Gradient descent

The derivative of a function represents its slope with respect to a specific variable. A gradient represents a multidimensional slope for functions with multidimensional inputs, such as a vector or matrix, in the form of partial derivatives for each of its parameters (cf. [33], p. 183; [34], p. 124; [51], pp. 47-48 and [53]). Gradient descent is an optimization algorithm that aims to minimize the output of the loss function by iteratively updating the parameters of a neural network. It calculates the gradient of the loss function based on the current set of parameters (weights and biases) and uses the gradient to adjust the parameters gradually towards its descent, therefore minimizing the loss (see Fig. 2.25).

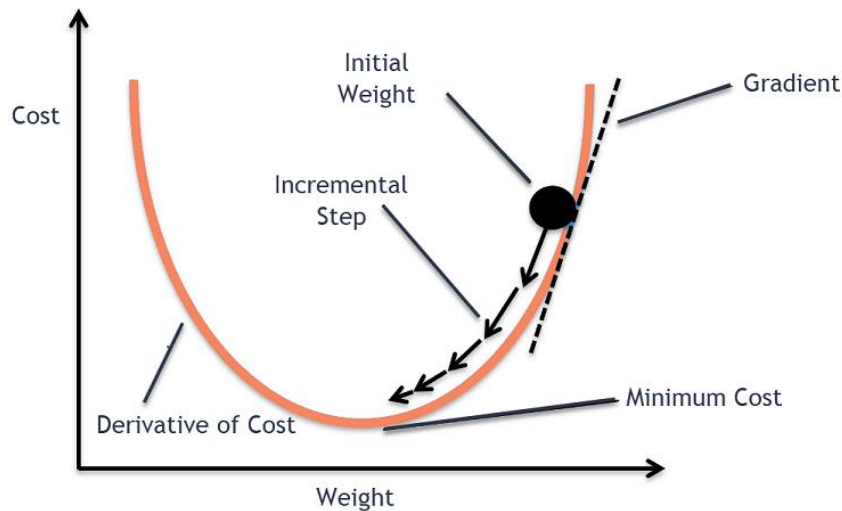


Figure 2.25: Gradient descent on a one-dimensional curve [52]

Figure 2.25 illustrates this with a simplified gradient in a two-dimensional space with a one-dimensional curve. As the gradient ∇f of a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ indicates the direction of the steepest ascent, the negative gradient $(-\nabla f)$ is used to move the parameters towards the direction of the steepest descent, gradually and iteratively determining the parameters for minimized loss or cost (cf. [31], p. 81 and [33], pp. 183-184). “Gradually and iteratively”

hereby refer to the steps visualized in Fig. 2.25. The size of these steps is determined by the so-called learning rate. The learning rate η is a hyperparameter that defines the size or dimension of the parameter adjustments per iteration of gradient descent, representing the distance to the next position on the curve (see Fig. 2.25). The position in terms of the new parameters is defined as follows:

$$W_{new} = W_{old} - \eta \nabla J(W_{old}) \quad (27)$$

$$B_{new} = B_{old} - \eta \nabla J(B_{old}) \quad (28)$$

with W_{new} and B_{new} as the new weight and bias parameters, and the distance is represented by the step size η , multiplied by the gradient of the loss function ∇J with current parameters W_{old} and B_{old} . If the gradient is equal to zero, a minimum has been reached, indicating that no further steps towards the steepest descent, and therefore, no further steps towards decreasing the loss, can be made (cf. [33], [34], [51], [52] and [53]).

Depending on the chosen learning rate η , the following scenarios can occur (see Fig. 2.26 and 2.27):

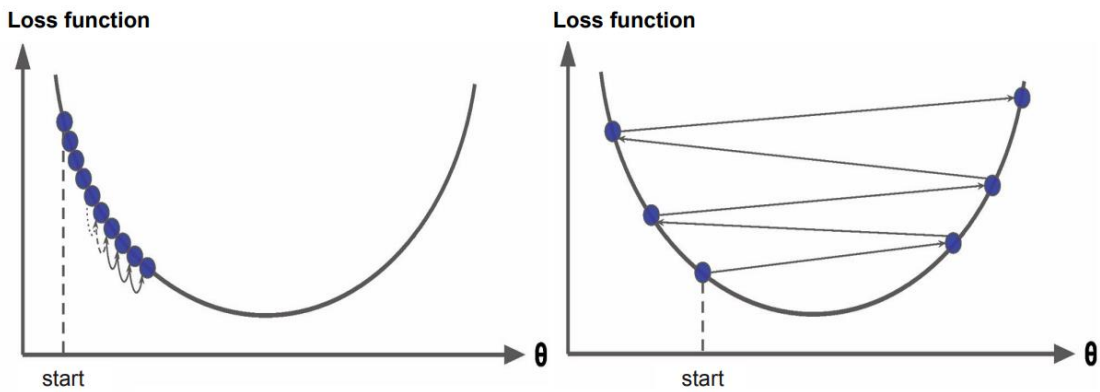


Figure 2.26: Learning rate, too low vs too high (adapted from [34], p. 122)

As the next parameters, and therefore the next position on the curve, are estimated or approximated by the learning rate in combination with the function's gradient, the size of the learning rate has a significant impact on the optimization result (cf. [51], p. 48). Figure 2.26 shows two diagrams of the same loss function with different learning rates. If η is too low, as visualized by the left diagram in Fig. 2.26, the training process progresses very slowly, as it requires a large number of parameter adjustments. If η is too high, on the other hand, as

illustrated by the right diagram in Fig. 2.26, it can overshoot the minimum and cause oscillations during the training process, which, in the latter case, can lead to a failure to converge (cf. [34], p. 122 and [44], pp. 87-90).

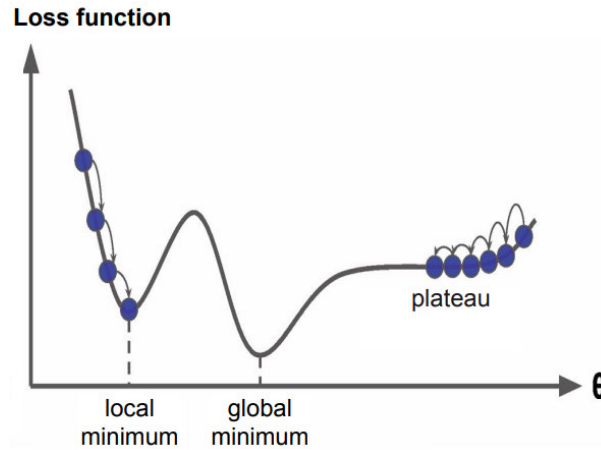


Figure 2.27: Plateau, local and global minimum within loss function (adapted from [34])

Applying these scenarios to a learning function that is not convex and possesses local minima or plateaus, as shown in Fig. 2.27, shows that the learning rate can have a significant impact on the effectiveness of gradient descent. If the learning rate is too low, the optimization process may never reach the global minimum, either by getting trapped in a local minimum, as the loss would increase when adjusting the parameters in both directions, or by plateauing, as it would take too many iterations to leave the plateau. Therefore, the learning rate must be large enough to overcome local minima or plateaus, but small enough to successfully reach the global minimum without overshooting it, independent of how or where the parameters are initialized (cf. [34], p. 123).

Stochastic Gradient Descent

Instead of calculating the gradient for the entire batch of training data, which is referred to as batch gradient descent, the stochastic gradient descent (SGD) uses a random sample of the training data to determine the gradient of the loss function and adjust the parameters accordingly. This enables the training of networks on large datasets, as it reduces the computational burden, allowing the algorithm to update the parameters more frequently, one data point at a time. While this makes the algorithm faster compared to the batch gradient descent, it can also be more irregular or erratic due to its random nature, causing fluctuations between different positions or adjustment values in the optimization process. This variability can be

both beneficial and detrimental, as it helps the algorithm to overcome local minima and increases the likelihood to find the global minimum, but it may also lead to overshooting the optimal point, resulting in suboptimal parameter values (cf. [34], p. 127).

A compromise between the two algorithms is the so-called mini-batch SGD, which uses a small adjustable batch of randomly chosen data points to compute the gradient within every iteration. It combines the benefits of both algorithms and is the most commonly used algorithm to optimize neural networks (cf. [52] and [54]). Other popular and widely used algorithms are based on the SGD, such as Adagrad, Adam and RMSprop (cf. [33], [34], [54] and [51], p. 60).

Backpropagation

During optimization, the neural network iterates through datapoints or batches of the training dataset and predicts an output. The predicted output is then compared to the desired output by the loss function, and a corresponding loss score is calculated (see Fig. 2.24). During this process, the data propagates forward through every layer of the neural network, where the weights and biases of the neurons contribute to the output. The backpropagation algorithm uses the loss score and propagates it back through the network, calculating the contribution of every layer and neuron connection to the loss score, starting from the output. It calculates the gradient of the loss function for all parameters contributing to the discrepancy and adjusts the weights and biases through gradient descent to minimize the loss. One iteration through the entire dataset is referred to as one epoch. During optimization, a neural network is trained over multiple epochs (cf. [34] and [54]).

Data Partitioning

In order to assess the performance of a neural network after its training, the training data set does not consist of the entirety of all data but is partitioned accordingly beforehand. The data is typically divided into three different datasets: a training dataset, a validation dataset, and a test dataset. The training dataset usually consists of 70-80% of the total data (cf. [33], [34] and [44]). As previously mentioned in Section 2.6, the training dataset is used to train the network based on its content. The validation dataset is used to validate the performance of the trained network and adjust its hyperparameters, presenting a set of previously unseen data to the network. Hyperparameters are parameters, that are provided externally to a neural network, unlike the weights and biases that a network learns through backpropagation itself

[51]. The adjustment of the hyperparameters is a highly iterative process, where the entire training is repeated, and the validation dataset is used after each hyperparameter adjustment to validate the resulting performance of the neural network. The performance is evaluated using defined quality measures, such as the accuracy or overall cost. Once this iterative process, also known as hyperparameter tuning, is completed and the desired performance is achieved, the network is tested with the test data set, a final set of data completely unknown to the network. The test data set is used to determine how the neural network performs under real-world conditions. This is necessary to evaluate if the training has been successful and if the network will perform well on new data, as the entire neural network was optimized using the validation dataset (cf. [34] and [51], pp. 97-98).

Instead of holding out a percentage of the data as a validation dataset, which is referred to as hold-out validation, another approach exists that further partitions the validation data into k -folds, known as k -fold cross-validation. In this method, the data is partitioned into k equally sized folds in order to subsequently perform training and validation k times. During cross-validation, one of the k partitions is used for validation, while the remaining $k - 1$ partitions are used for training, as shown in Fig. 2.28. The resulting validation performance is represented by the average across all folds [51].

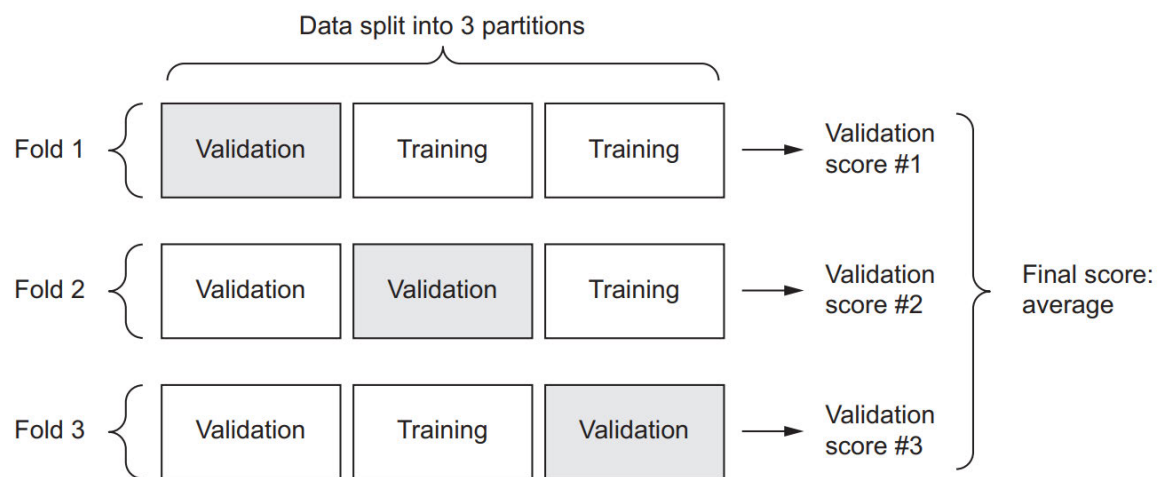


Figure 2.28: 3-fold cross-validation [51]

The main advantage of k -fold cross-validation is that it demonstrates how well a neural network or machine learning model with a certain set of hyperparameters is able to generalize by validating across different subsets of data while training with the remaining, varying

training sets. Additionally, if only a small set of data is available, it maximizes its use and ensures that the performance is independent of a certain partitioning of the data. The performance is measured as the average across all folds and therefore includes the standard deviation, which presents additional useful information regarding the model's abilities (cf. [33], p. pp. 97-98; [34], 76 and [51], p. 99). This approach is also beneficial for identifying overfitting, which will be explained in the following section:

Overfitting and Underfitting

The desired outcome of training and optimizing a neural network is for it to generalize well and apply its learned knowledge to new data as effectively as it does to the data it was trained on. While training can be controlled and adjusted, generalization can only be observed through validation as a result of the optimization process [51].

Underfitting occurs when a neural network suffers from a low capacity. Its performance reaches a peak and cannot improve further as the data is too complex for it to learn from. Measures against underfitting include increasing the model's size to enhance its resources, analytic capabilities, and overall capacity (cf. [51] and [54]).

Overfitting occurs when a neural network performs well on known data but poorly or less effectively on new and unseen data. This indicates that the network has memorized the training data, learning patterns that apply only to it specifically. In general, it means that the neural network has high learning capacities but not enough data to learn from effectively, leading it to learn misrepresentative or misleading patterns within the data by using its excess capacity. The solution is to provide more and relevant data, so the network has more training examples to learn from, or, if that is not possible, limit or restrict its capacities through regularization [51]. This can be achieved by reducing the networks size, using dropout layers or perform early stopping. Overfitting can occur in every area of machine learning applications. If a SVM overfits for example, the parameter C can be reduced to regularize it accordingly (see Section 2.6.5 and cf. [34], p. 157).

Dropout layers are layers within a neural network where a certain percentage of neurons, defined by the dropout rate, randomly turn off during training. During each training step, different neurons turn off according to the dropout rate, and neurons that were turned off before can be turned back on. This forces the active neurons to adapt, making them robust

against overfitting and therefore enhancing the network's ability to generalize. During testing, all neurons are turned on (cf. [34], pp. 368-371; [51], pp. 109-110 and [54]).

Early stopping monitors the training and validation progress during optimization and stops the training automatically as soon as the validation loss reaches a minimum and does not further improve. This method allows to identify if and when the neural network starts to overfit, returning to the point where the minimum and therefore the best performance is reached (cf. [34], p. 143).

A method to provide additional data without necessarily collecting it, is called data augmentation. It involves creating new data from already existing data by duplicating data points and modifying them slightly. This can prevent overfitting and enhance the performance of the neural network, as the network needs to learn from a wider range of slightly differing data, under the condition, that the data contains learnable features and conditions (cf. [34] and [51], p. 130).

All in all, the ability of a network to learn and achieve the desired performance is highly dependent on its composition and overall hyperparameters. Essential aspects of defining these hyperparameters are explained in the following:

Hyperparameters

As previously mentioned, hyperparameters are parameters provided externally to a neural network, unlike the weights and biases that a network learns through backpropagation itself [51]. The most important hyperparameters are summarized below:

- Training Data: While the training data itself is not a hyperparameter, its volume, the applied preprocessing, including the selection of suitable features as well as normalization or scaling measures, significantly influence the training results and must be considered, depending on the data composition and whether neural networks perform these automatically or not.
- Number of hidden layers: The number of hidden layers in a neural network increases its overall capacity and allows it to recognize complex patterns. However, this increases the risk of overfitting and its computational expense. The smallest network

that can fulfill the desired task, is usually the most practical, as it has less computational effort and a shorter processing time than large networks. In addition, smaller networks can achieve better generalization with limited data, provided no early stopping is used. A good approach is to start with one or two hidden layers and gradually increase the number (cf. [34], p. 326; [48] and [51]).

- Number of neurons per hidden layer: In general, two approaches are proposed for aligning the number of neurons per hidden layer [33] [34]:
 1. **Pyramidal Approach**: In this method, the number of neurons decreases from the input layer to the output layer, resembling a pyramid. This means that neurons must compress the information they receive, thereby reducing its complexity.
 2. **Cuboidal Approach**: In this method, layers are structured like a cuboid, with each layer containing the same number of neurons. This approach has proven effective in practice and allows for only one hyperparameter to be adjusted, as this hyperparameter simultaneously determines the number of neurons in all layers.

A good approach is to gradually increase the number of neurons per hidden layer until the network overfits. Additionally, [34] proposes defining a higher number of hidden layers and neurons than needed and using early stopping, as this approach is simpler and more efficient (cf. [33], p. 203; [34], p. 327; and [51]).

- Activation function: The activation function determines how the neuron reacts to the input signal with its output signal. In general, ReLU is proposed as the default activation function for hidden layers [34]. Regarding the last layer, it specifies the output of the neural network and establishes limits or restrictions, such as softmax for example [51].
- Loss function: The loss function usually results from the task or application of the neural network. The loss function is generally chosen as follows [51]:
 - Regression: MSE
 - Binary classification: Binary cross-entropy
 - Multi-class classification: categorical cross-entropy

- Optimization: The optimization process involves the learning rate in combination with the learning function, or so-called optimizer. [51] proposes RMSprop (Root Mean Squared Propagation) with a default learning rate of 0.001 as suitable for most applications.
- Number of iterations or epochs: The number of training epochs is often not specified due to the use of early stopping (cf. [34], p. 328).

Hyperparameter tuning is an iterative process where the optimal parameter set is often determined through trial and error. Techniques, such as grid search or random search allow to optimize this process by searching through a defined or random grid of parameters (cf. [51], [53] and [54]).

2.6.8 Deep Learning

Networks that possess two or more hidden layers are referred to as deep neural networks or deep learning (cf. [34] and [50]). Since all deep learning algorithms are neural networks, all previously discussed contents in Section 2.6.7 apply to them as well. A specific deep learning model that will be used within this thesis is the so-called Autoencoder (AE), which will be explained in the following:

Autoencoder

Autoencoders are among the most widely used deep neural networks for anomaly detection within the context of predictive maintenance (cf. [17]). As part of unsupervised learning methods (see Section 2.6.2), the autoencoder learns features representative for normal data in order to identify anomalies. During training, the Autoencoder compresses the normal data and learns its compressed representation, also referred to as the latent representation (see Fig. 2.29). Afterwards, the autoencoder tries to reconstruct the learned information and compares it to its input, measuring the discrepancy or deviation. The autoencoder consists of two main components, an encoder and a decoder, responsible for the processes of compression and reconstruction, respectively (cf. [17], [33], [34] and [55]). Therefore, both components are structured like a pyramid, with the input and output as their respective bases (see Fig. 2.29).

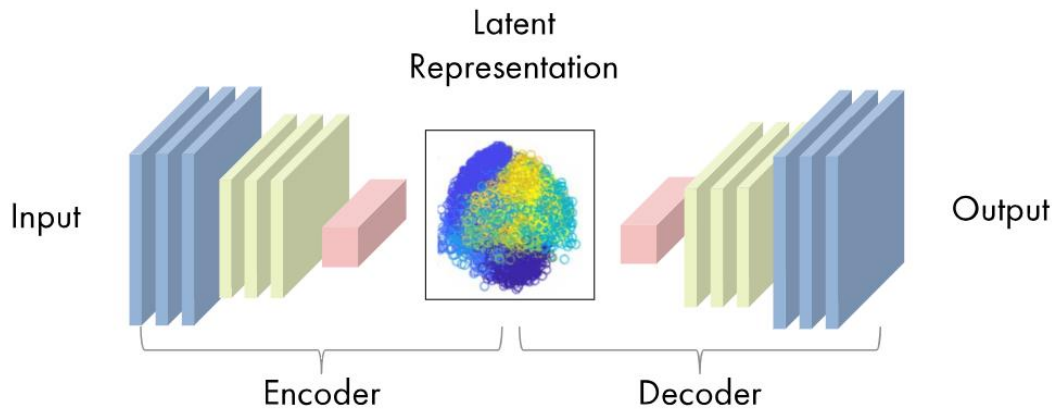


Figure 2.29: Structure of an Autoencoder [54]

After training, the Autoencoder can be used to predict whether given data points represent anomalies by reconstructing the input. If it is able to reconstruct the input successfully, it is regarded as normal. Otherwise, the input is declared an anomaly. The decision is usually made based on a calculated error between input and output, similar to a loss function, and using an iteratively determined threshold value [56].

3 The Industry 4.0 Production Plant

Within this chapter, the to be enhanced industry 4.0 production plant is presented (see Fig. 3.1). It is located at the Shanghai-Hamburg College at the University of Shanghai for Science and Technology and was manufactured by Lucas Nuelle.

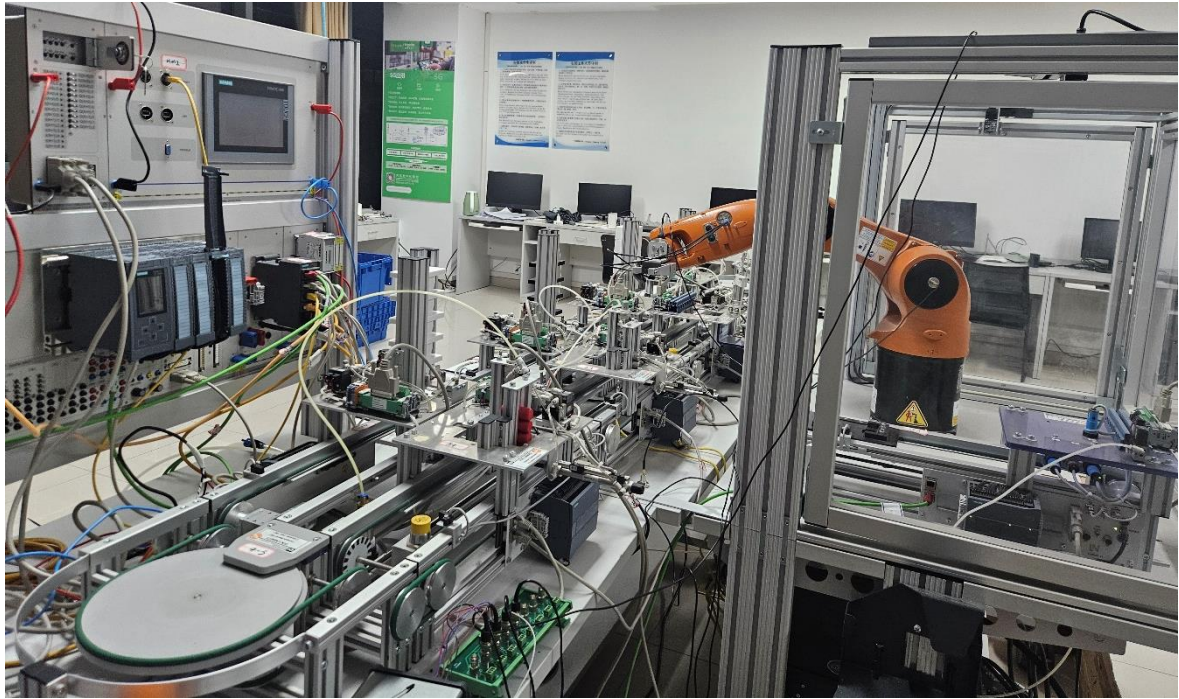


Figure 3.1: Industry 4.0 production plant [67]

In the following, the manufactured products within the production plant, its production process as well as its components will be explained:

3.1 Products and production process

The industry 4.0 production plant represents an automated and interconnected block production system, producing an assembled block of so-called workpieces. In the following, workpieces and block are often used synonymously to refer to the individual components as well as the final product, in form of the assembled block or workpiece. The individual workpieces are presented in Fig. 3.2:



Figure 3.2: Individual workpieces (adapted from [57])

The production plant features an OpenCart web shop (see Section 3.2.3 and Fig. 3.15), where a user or customer can decide which product he wants to purchase or order, and therefore, which product the production plant should produce. The customer can hereby freely decide, which combination of individual workpieces the final product should entail, as long as they consist of an upper and a lower piece (see Fig 3.2).

The lower workpieces are shown in the lower left cornered area of Fig. 3.2, numbered as 1. The upper workpieces are shown in the area above, numbered as 2. Each component consists of at least, and at most, one of each of these two workpieces, independent of their color, fully to the desire of the customer. Both workpieces are combined by integrating an upper workpiece into a lower workpiece. The connection or combination of both workpieces can then be optionally enhanced, using of two pins, shown in the right cornered area of Fig. 3.2,

numbered 3. The numbers in this case, also refer to the assembly sequence of the workpieces. The assembly is performed completely automatically by eight individual workstations, as well two extensions:

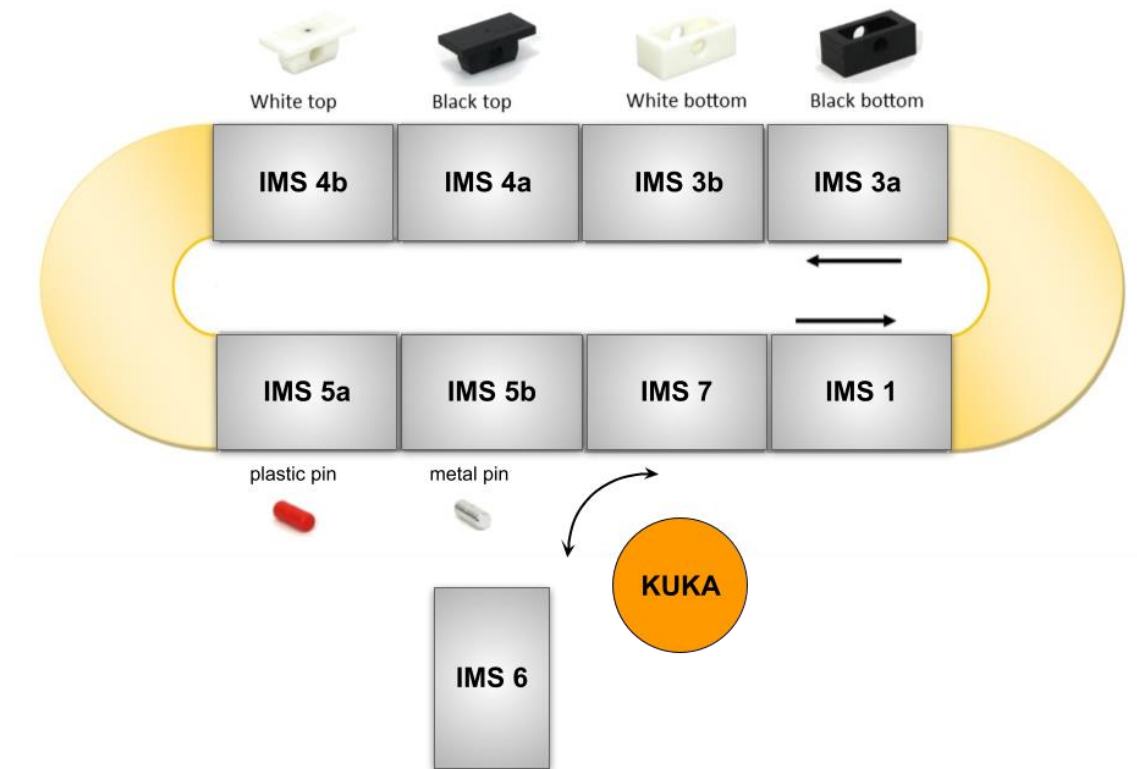


Figure 3.3: Schematic plant overview (adapted from [58])

The production plant consists of nine industrial mechatronic systems, abbreviated IMS, as well as one industrial robot manufactured by KUKA. Eight IMS Stations, arranged in a circulating and closed-loop conveyor system, as shown in Fig. 3.3, are responsible for the production of the workpieces. The IMS Station outside the depicted and closed-loop conveyor system (see Fig. 3.3, IMS 6) is a testing station, to which the KUKA robot enables the transfer from and to the production loop. The entire production process, starting with an order, is as follows:

After an order is placed through a customer on the web shop, it is forwarded to the manufacturing execution system database. The manufacturing execution system, or MES, manages the orders within the database and triggers them, if the production plant is idle and capable of production.

In its idle state, three empty workpiece carriers, equipped with RFID tags, are located within the production plant. The initial position of these carriers, for the sake of simplicity referred to as carrier 1 (C1), carrier 2 (C2) and carrier 3 (C3), is illustrated in a simplified manner in Fig. 3.4.

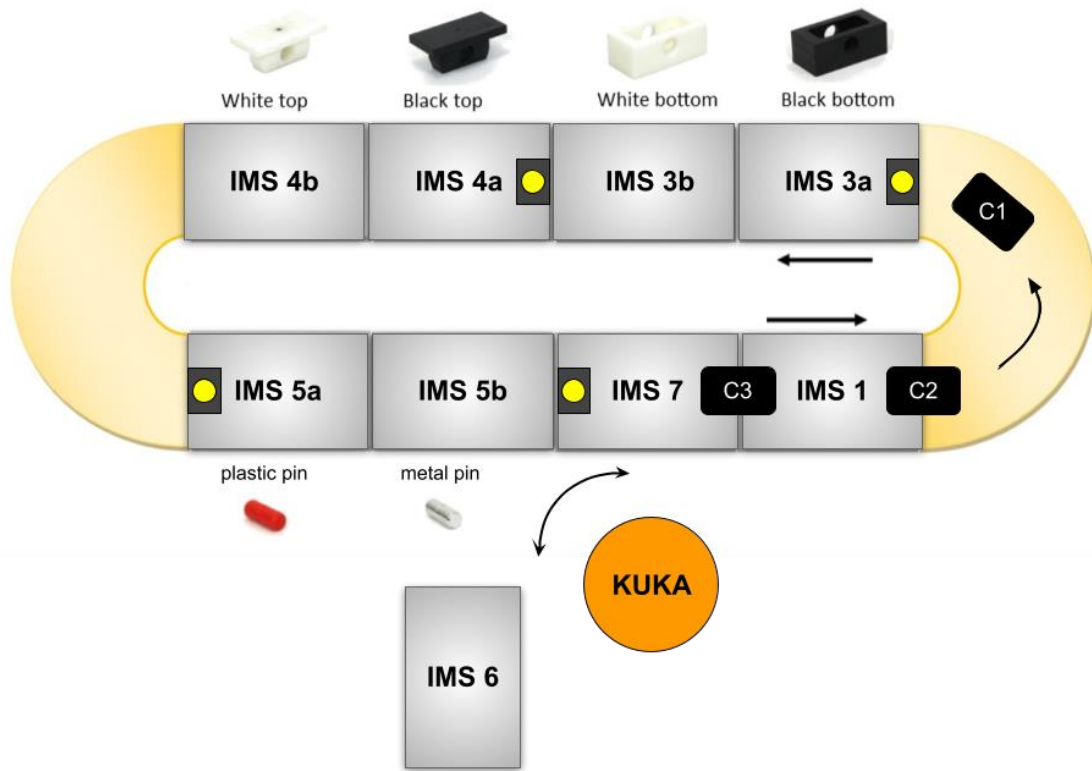


Figure 3.4: RFID read and write units and carrier flow (adapted from [58])

Carriers C3 and C2 are hereby located at the front and rear end of IMS Station 1. Figure 3.4 shows C1 in a transfer state, with its idle position across carrier 2, in front of IMS 3a. The production plant is equipped with four IMS Stations featuring RFID read and write units, depicted as rectangles with a single yellow dot at the beginning of each of these stations in Fig. 3.4. With the RFID tags and RFID read and write units, each carrier, which contains an individual and recognizable identification number, is assigned the product information of the workpiece it is carrying and needs to produce. From the beginning of the production process, each carrier possesses knowledge of the final product and is aware of each individual station or step required to achieve this final state. This also applies to the overall production system, which reads, writes and updates the information on the RFID tags after each

production step. According to the definition of smart objects within Industry 4.0 (see Section 2.5.2), the carriers can be considered smart carriers or, in relation to the workpieces they transport, can also be referred to as smart workpieces.

After an order is triggered and production is initiated by the MES, the carrier, provided it is in the idle position of C1, moves over the RFID read and write unit at IMS Station 3a and receives the corresponding product information. The other two carriers shift positions, moving from Carrier position C3 to C2 and from C2 to C1, assuming that no carrier is currently within the production process and that production starts from the previously described idle state (see Fig. 3.4).

During production, the carrier moves through each of the circularly arranged IMS Stations, starting from 3a, as indicated by the arrows in Fig. 3.4. Depending on the customer's preference and the corresponding production information the carrier possesses, the carrier will stop at the respective stations in order to complete the relevant production steps. At IMS Stations 3a and 3b, the lower workpieces, also referred to as bottom parts, are assembled (see Fig. 3.4). The carrier will stop at IMS Station 3a for a black bottom part or at 3b for a white bottom part, depending on the content of the order. Similarly, IMS Station 4a handles the assembly of black upper or top parts, while 4b manages the assembly of white upper or top parts (see Fig. 3.4), following the same procedure.

After passing the first four stations, the carrier stops at IMS Station 5a for a red pin or at Station 5b for a metal pin. If the desired product does not include a pin, the carrier continues moving and passes through IMS Station 5a and 5b without stopping, proceeding directly to IMS Station 7 as intended. At this point, after IMS Station 5b and right before IMS Station 7, one of two scenarios may occur. Through an HMI touch panel, it can be selected whether the KUKA robot and, consequently, the testing station IMS 6 should be integrated into the manufacturing process, determining if the product should undergo a final non-destructive test regarding its composition. If the KUKA robot is selected or enabled, it transfers the workpiece to the IMS Station 6 and its own separate carrier. The previous smart carrier remains at the same location within the circulating closed-loop conveyor system, waiting for the part to be returned after testing. During testing, the IMS Station 6, which features four individual sensors, checks if the final product matches the desired specifications. If all the individual components match the original order, the robot returns the workpiece to the carrier

waiting between IMS Stations 5b and 7. If not, it transfers the workpiece to a separate area, consisting of three individual slots for sorted-out workpieces, and the smart carrier remaining in its waiting position returns to an available idle position. As a result, the order is automatically reinitiated, and the production of the entire workpiece is repeated. The outcome of the non-destructive testing is displayed on the HMI.

After a successful test and the workpiece is returned to the smart carrier, it proceeds to IMS Station 7, the final stage of production. Here, the workpiece is grabbed by a handling device equipped with vacuum suction cups and transferred out of the production plant onto a designated platform. The carrier then moves to IMS Station 1, into an available idle position, completing one production cycle (see Fig. 3.4).

3.2 Components

Within this section, the previously described IMS Stations and their components, along with the KUKA robot and the production, planning and execution system, as well as the underlying logic and architecture will be explained.

3.2.1 IMS Stations

The circulating closed-loop conveyor system consists of multiple individual conveyor belts (see Fig 3.4 and 3.5). These conveyor belts, controlled by a Siemens S7-1200 PLC, also form the basis of the IMS Stations, with the IMS Station 1 representing its initial form or structure (see Fig 3.5). The IMS Stations, which are responsible for assembling the same type of workpiece and therefore share the same number, are presented as one type of station within this section.

IMS Station 1

The conveyor belt features a 24V DC motor capable of bidirectional movement and uses a dual-belt configuration to transport the smart workpiece carriers (see Fig. 3.5). It includes two magnetic-field end-limit sensors that detect when a carrier - equipped with integrated magnets identifiable by the end-limit sensors, in addition to the previously mentioned RFID

tag - reaches the front or rear end of the conveyor belt. These sensors signal the system to deactivate the previous belt and activate the next one.

The ends of the conveyor belts at each station are positioned adjacent to one another, allowing the workpieces to be transported throughout the entire circular production system. 180° conveyor belt turntable segments, connected to the motors of the conveyor belts at IMS Station 3a and 5, transfer the workpieces to the following stations on the opposite side and moving direction, as illustrated by the yellow 180° connection lines in Fig. 3.3 and 3.4.



Figure 3.5: Conveyor belt IMS Station 1 [67]

The I/O signals of the PLC, connected to the sensors and the motor of the conveyor belt, are presented below (cf. [57], [58] and [59]):

Table 3.1: Conveyor belt [59]

Description	ID	I/O
End position sensor - left	B1	%I1.3
End position sensor - right	B2	%I1.4
Conveyor belt - right		%Q1.0
Conveyor belt - left		%Q1.1

IMS Stations 3a and 3b

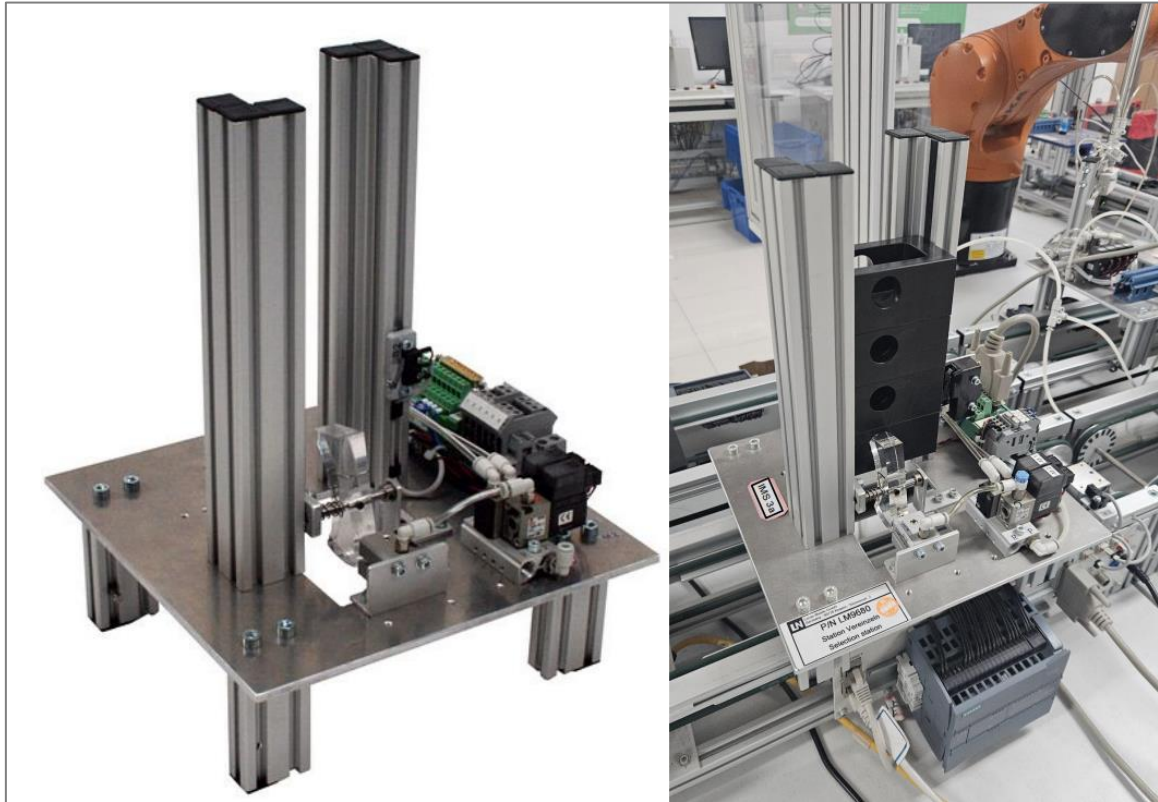


Figure 3.6: IMS Station 3a (adapted from [57] (left) and [67] (right))

The IMS Stations 3a and 3b are responsible for assembling the bottom part. They use the conveyor belt as a base and are positioned above it (see Fig. 3.6). If a carrier moves through the station and needs to receive a bottom part, the station extends a stopper that prevents the carrier from proceeding, positioning it under the workpieces stock tower, shown in Fig. 3.6, in order to receive the workpiece. Shortly after, the conveyor belt is deactivated and the station, equipped with pneumatic valves and a corresponding mechanism that briefly releases, allows the workpiece to drop onto the carrier. Once the carrier has received the workpiece, the stopper retracts, the conveyor belt is activated, and the carrier moves on to the next station.

In addition to the sensors and motor of the conveyor belt, IMS Stations 3a and 3b are equipped with a pneumatic valve block with two valves that control the stopper and the dropping mechanism, as well as a tactile or mechanical switch to monitor the magazine level, and a magnetic sensor to detect the position of the stopper. Their I/O signals, excluding those

already shown for the conveyor belts as they share the same address and ID (see Table 2), are presented below (cf. [57], [58] and [59]):

Table 3.2: IMS Station 3a and 3b [59]

Description	ID	I/O
Magnetic sensor – stopper at top	B3	%I0.2
Switch – magazine occupancy	B4	%I0.3
Lower stopper	M1	%Q0.0
Activate sort cylinder	M2	%Q0.2

IMS Stations 4a and 4b

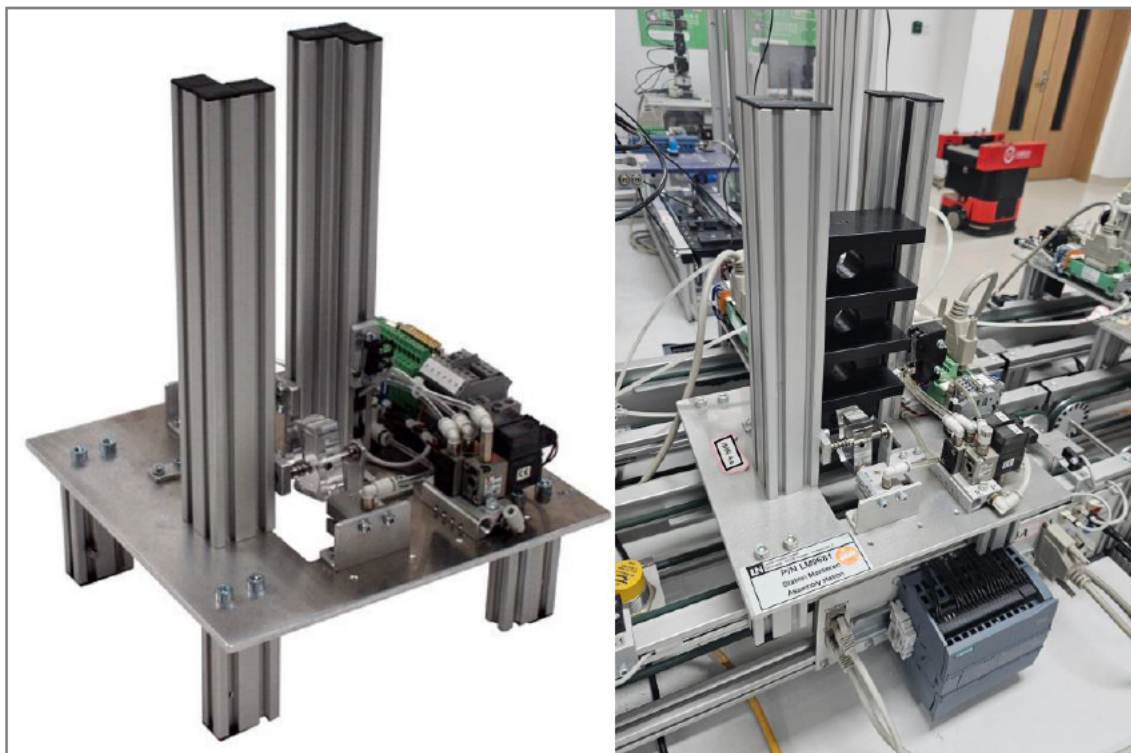


Figure 3.7: IMS Station 4a (adapted from [57] (left) and [67] (right))

The IMS Stations 4a and 4b have the same setup as their counterparts responsible for the assembly of the bottom parts (see Fig. 3.7). Only the mechanical device or mechanism responsible for holding and releasing the workpieces has been adapted to the geometry of the upper parts. Therefore, the I/O signals for these stations are the same as those shown in Table 3.2, in addition to those shown in Table 3.1 for the conveyor belt (cf. [57], [58] and [59]).

IMS Stations 5a and 5b

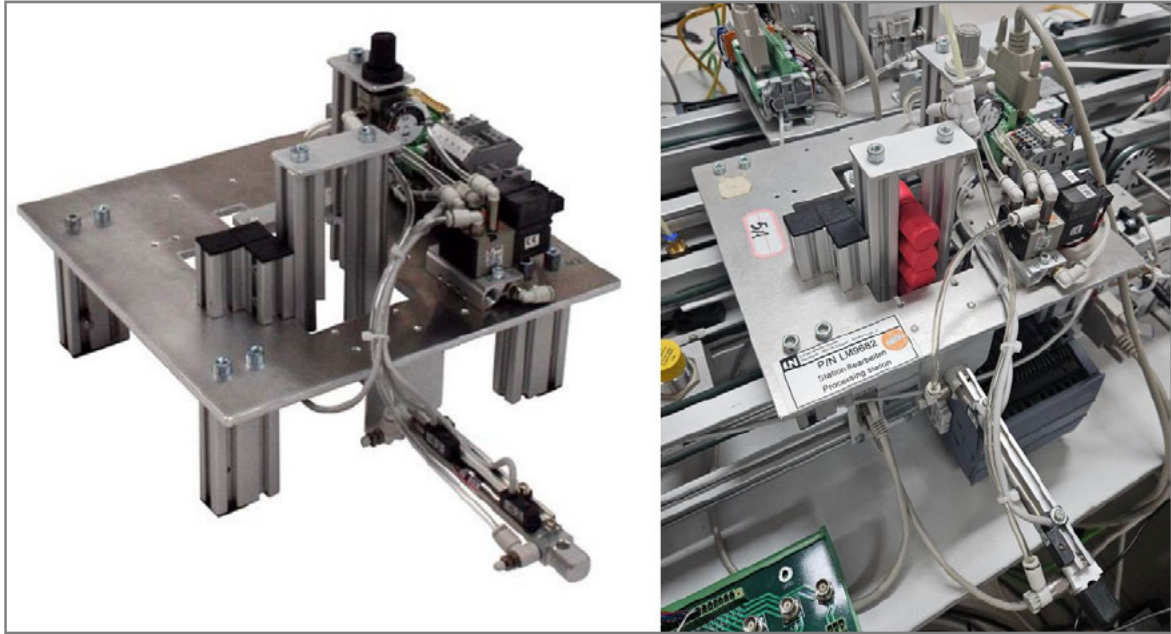


Figure 3.8: IMS Station 4a (adapted from [57] (left) and [67] (right))

The IMS Stations 5a and 5b enhance the connection of the upper and lower workpieces by pinning them together, using a pneumatic cylinder to insert one of two different pins into the workpieces and their co-aligning recesses (see Fig. 3.2). These stations are also equipped with a stopper and use the conveyor belt as a base, being positioned above it as well (see Fig. 3.7). Additionally, the stations feature a light barrier and a reflector to monitor the magazine level, replacing the mechanical switch, and the pneumatic cylinder is equipped with two magnetic end-limit sensors, resulting in the following I/O signals, excluding those for the conveyor belts (cf. [57], [58] and [59]):

Table 3.3: IMS Station 5a and 5b [59]

Description	ID	I/O
Magnetic sensor – stopper at top	B3	%I0.2
Sensor – magazine occupancy	B4	%I0.3
Magnetic sensor – pressing cylinder not actuated	B5	%I0.4
Magnetic sensor – pressing cylinder actuated	B6	%I0.5
Lower stopper	M1	%Q0.0
Activate sort cylinder	M2	%Q0.2

IMS Station 6

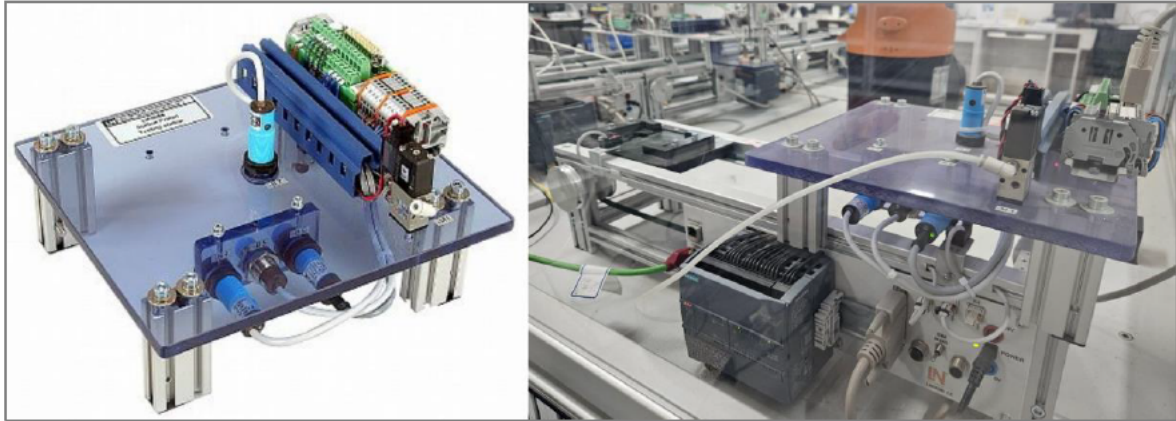


Figure 3.9: IMS Station 6 (adapted from [57] (left) and [67] (right))

The IMS Station 6 moves the workpiece, along with its own carrier, after receiving it from the robot (see Section 3.1) and positions it under the sensor using its stopper at the rear end of its conveyor belt (see Fig. 3.9). The station is equipped with four different sensors: a capacitive sensor, an inductive sensor, and two optical sensors. The optical sensors detect the color of a workpiece. One sensor is positioned directly on top of the station for the upper part (see Fig. 3.9), and the other is positioned on the side of the station for the lower part. These optical sensors return a binary value: 0 for black and 1 for white. The capacitive sensor and the inductive sensor are also mounted on the side of the station (see Fig. 3.9). The capacitive sensor determines whether the workpiece has a pin, while the inductive sensor identifies whether the pin is metal or plastic.

All sensors are oriented towards the workpiece, with the workpiece being positioned underneath them. During the non-destructive testing, the workpiece remains briefly within the station before being transported back to the start of the conveyor belt (see Fig. 3.9), where it can be picked up by the robot. The I/O list for IMS Station 6 is as follows, excluding the signals for the conveyor belt (cf. [57], [58] and [59]):

Table 3.4: IMS Station 6 [59]

Description	ID	I/O
Magnetic sensor – stopper at top	B3	%I0.2
Optical sensor- bottom part	B4	%I0.3

Inductive sensor	B5	%I0.4
Capacitive sensor	B6	%I0.5
Optical sensor- top part	B7	%I0.6

IMS Station 7

The IMS Station 7 features a designated platform for completely assembled and manufactured workpieces, marking the final stage of production (see Fig. 3.10). The platform can hereby hold two workpieces at a time. The station includes a swivel table that rotates 90° from its initial position above the platform to its handling position above the opening in the center of the station to pick up the workpiece (see Fig. 3.10).

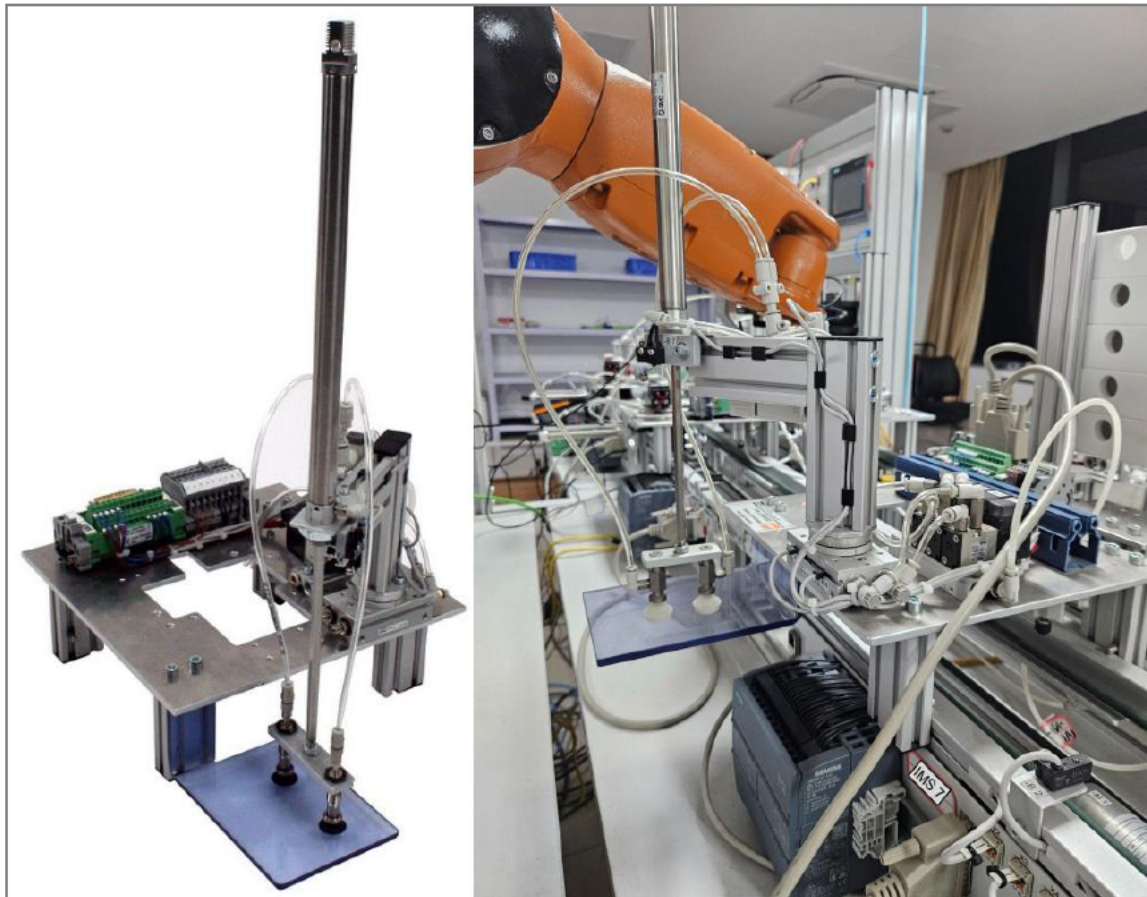


Figure 3.10: IMS Station 7 (adapted from [57] (left) and [67] (right))

A carrier with a fully assembled workpiece arrives at IMS Station 7 and is stopped by its stopper to be positioned under the opening in the center of the station. The swivel table is

equipped with a vertical cylinder that includes a vacuum lifter. This lifter descends through the opening, uses vacuum suction cups to grasp the workpiece, and then returns to its elevated position above the opening. From there, the swivel table rotates 90° back to its original position above the platform. The vertical cylinder then lowers to place the workpiece onto the platform. Finally, the vacuum is turned off, and the vertical cylinder returns to its starting position, leaving the workpiece on the designated platform.

The swivel table is equipped with two position sensors for the described positions, as well as a vacuum sensor (see Table 3.5). Like the previous stations, it uses the conveyor belt as a base, being positioned on top of it. The signals for IMS Station 7 are presented below, excluding the signals for the conveyor belt (cf. [57], [58] and [59]):

Table 3.5: IMS Station 7

Description	ID	I/O
Magnetic sensor – stopper at top	B3	%I0.2
Magnetic sensor- swivel table 0°	B4	%I0.3
Magnetic sensor- swivel table 90°	B5	%I0.4
Vacuum monitoring	B6	%I0.5
Switch- lift cylinder at top	B7	%Q0.6
Swivel table from 0 to 90	M1	%Q0.0
Lower stopper	M2	%Q0.1
Lower cylinder	M3	%Q0.2
Vacuum on	M4	%Q0.3

3.2.2 KUKA Robot

The industrial KUKA robot, which transports workpieces between the closed-loop conveyor system and the testing station, is a 6-axis KR6 R700, along with a KRC4 controller and a smartPAD (see Fig. 3.1). During production, the robot remains in its pickup position within the free space above the adjacent conveyor belts of IMS Stations 5b and 7 (see Fig. 3.11). When a carrier stops underneath the robot and a workpiece needs to be transported to the

testing station, the robot moves towards the smart carrier, grips the workpiece with its end-effector, and transports it to the carrier at IMS Station 6 (see Fig. 3.11).

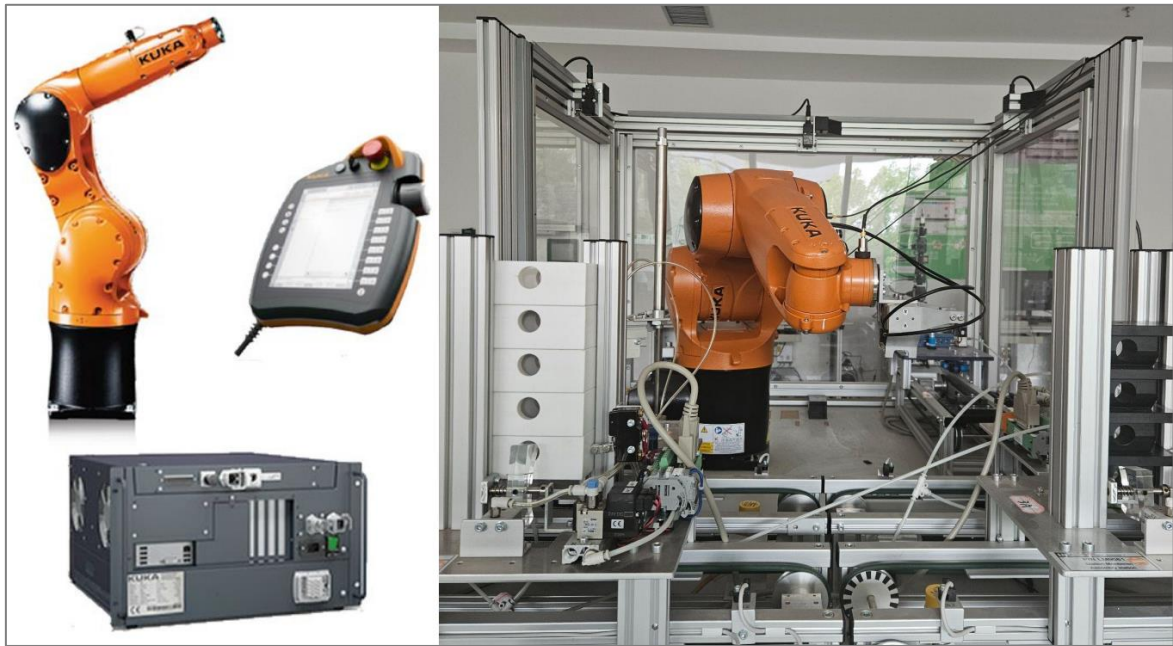


Figure 3.11: KUKA robot KR6 R700 (adapted from [57] (left) and [67] (right))

During the test, the robot stays in a waiting position above the initial position of the testing stations at the beginning of the station's conveyor belt (see Fig. 3.9). If the non-destructive testing of the workpiece is successful and the workpiece matches the desired specifications, the robot picks it up with its end-effector and transports it back to the closed-loop conveyor system, where it is placed on the waiting smart carrier (see Section 3.1).

If the test is unsuccessful and the workpiece does not match the desired composition, the robot transports the workpiece to one of three designated sort-out slots, depending on which is available (see Fig. 3.11). If all the slots are occupied, the robot will wait in a designated position and will only pick up the workpiece once the operator confirms via the HMI touch panel that the slots have been cleared. After picking up and placing the workpiece, the robot returns to its original idle position, fixed at a certain height between IMS Stations 5b and 7 (see Fig. 3.11 and cf. [57], [58] and [59]).

3.2.3 ERP-Lab

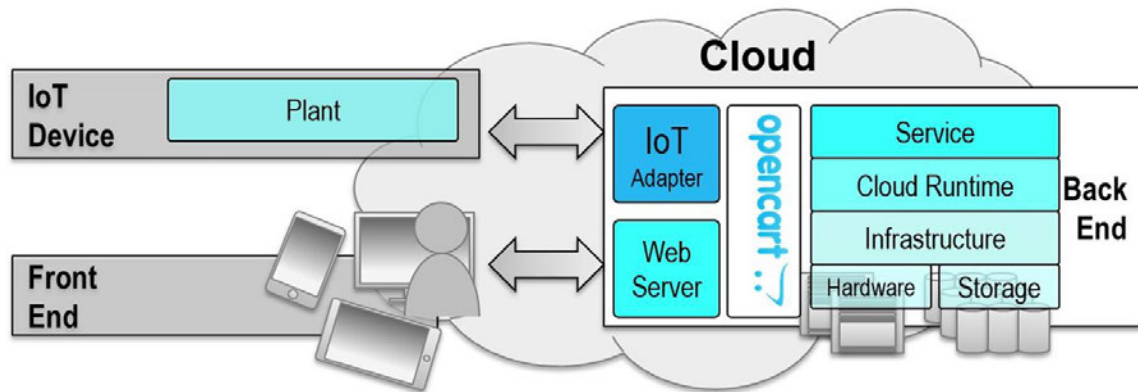


Figure 3.12: Design and structure of ERP-Lab [58]

The ERP-Lab, short for Enterprise Resource Planning Laboratory, runs as a cloud system within the production plant's own intranet (see Fig. 3.12). It includes administrative functionalities, managing applications and equipment relevant for production, resource planning, and manufacturing execution. The ERP-Lab's functionalities, logic, and data are provided to the production plant, representing an IoT device, via an IoT adapter, which connects the cloud with its PLCs (see Fig. 3.12). The IoT adapter is also connected to the OpenCart web shop, which is realized with a web server, providing a user interface. The user interface can be accessed via a browser, regardless of the device or operating system (see Fig. 3.12).

The cloud's required hardware and storage are provided by a Windows PC that includes the cloud runtime system as a virtual Linux machine. The ERP-Lab communicates with the PLCs using the resource-efficient CoAP (Constrained Application Protocol) interface, responding to queries from the PLCs. This interface utilizes the User Datagram Protocol (UDP) to facilitate communication and data exchange. The content on the user interface is realized through HTML (Hypertext Markup Language) data exchanges with the web server (cf. [57], [58] and [59]). A view of the web server's user interface, as well as its contents, are presented in Fig. 3.13:

ERP-Lab v2.1.0

Order Stock SCADA MES ERP Shop Analytics

ID	Order	Product	Duration	Finished	Waited																																						
#1939	oc-1131	In-wp-wb	49.0 s	19.06.24 22:26	28.0 s																																						
#1938	oc-1130	<table><tr><th>Step</th><th>Segment</th><th>Duration</th><th>Finished</th></tr><tr><td>#3</td><td>IMS3A</td><td>4.2 s</td><td>19.06.24 22:25:47</td></tr><tr><td>#1934</td><td></td><td></td><td>ID: 5879</td></tr><tr><td>#1936 #4</td><td>IMS4A</td><td>5.3 s</td><td>19.06.24 22:25:56</td></tr><tr><td>#1935</td><td></td><td></td><td>ID: 5882</td></tr><tr><td>#1933 #5</td><td>IMS5A</td><td>8.2 s</td><td>19.06.24 22:26:07</td></tr><tr><td>#1932</td><td></td><td></td><td>ID: 5895</td></tr><tr><td>#1931 #6</td><td>IMS5B</td><td>17.8 s</td><td>19.06.24 22:26:26</td></tr><tr><td>#1930 #7</td><td>IMS7</td><td>4.7 s</td><td>19.06.24 22:26:32</td></tr></table>				Step	Segment	Duration	Finished	#3	IMS3A	4.2 s	19.06.24 22:25:47	#1934			ID: 5879	#1936 #4	IMS4A	5.3 s	19.06.24 22:25:56	#1935			ID: 5882	#1933 #5	IMS5A	8.2 s	19.06.24 22:26:07	#1932			ID: 5895	#1931 #6	IMS5B	17.8 s	19.06.24 22:26:26	#1930 #7	IMS7	4.7 s	19.06.24 22:26:32		
Step	Segment	Duration	Finished																																								
#3	IMS3A	4.2 s	19.06.24 22:25:47																																								
#1934			ID: 5879																																								
#1936 #4	IMS4A	5.3 s	19.06.24 22:25:56																																								
#1935			ID: 5882																																								
#1933 #5	IMS5A	8.2 s	19.06.24 22:26:07																																								
#1932			ID: 5895																																								
#1931 #6	IMS5B	17.8 s	19.06.24 22:26:26																																								
#1930 #7	IMS7	4.7 s	19.06.24 22:26:32																																								
#1929	oc-1130	In-wp-wb	50.4 s	19.06.24 22:00	1.6 s																																						

Figure 3.13: ERP-Lab v2.1.0 (Order view) [67]

Figure 3.13 shows a view of the ERP-Lab, the Order view, in particular. Alongside the order tab, it includes six additional subsites or tabs, which will be explained from left to right, excluding the Analytics view, as it currently possesses no useful or beneficial functionalities, in the following (see Fig. 3.13 and cf. [58] and [59]):

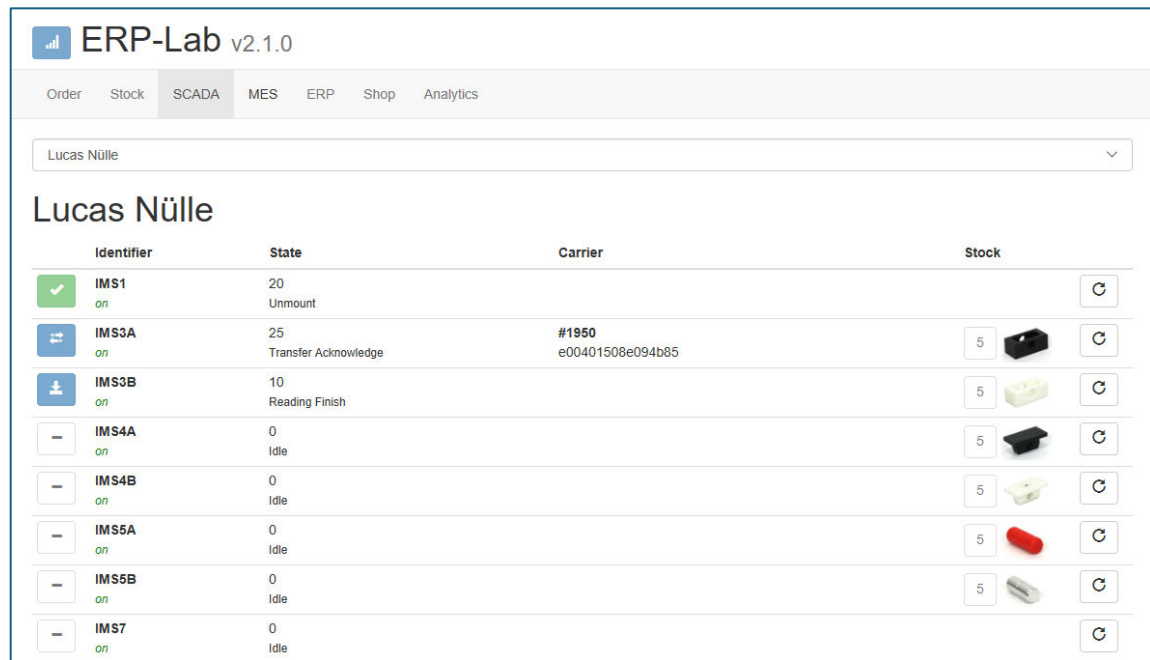
Order

The order view includes a list of all orders received through the OpenCart web shop, therefore including a list of products previously assembled, currently in production, or waiting for production, as well as their corresponding order ID, production or waiting duration, and a timestamp, marking the end of their production (see Fig. 3.13). The subsite also includes a more detailed view of the individual components of a workpiece, available by clicking on the magnifier-icon, illustrated by the red lines and arrow in Fig. 3.13. The additional view includes the production step as well as a timestamp marking its end, the involved station and duration of each individual component (see Fig. 3.13).

Stock

This view includes the current magazine or stock level of each station and can be manually increased to a number of five pieces per station. This functionality is also included within the following subsite.

SCADA



The screenshot shows the SCADA interface of ERP-Lab v2.1.0. At the top, there is a navigation bar with tabs for Order, Stock, SCADA (selected), MES, ERP, Shop, and Analytics. Below the navigation bar, a search bar contains the text 'Lucas Nülle'. The main content area is titled 'Lucas Nülle' and displays a table with the following columns: Identifier, State, Carrier, and Stock. Each row represents a workpiece station with its current state and a stock counter.

Identifier	State	Carrier	Stock
IMS1 <i>on</i>	20 Unmount		5
IMS3A <i>on</i>	25 Transfer Acknowledge	#1950 e00401508e094b85	5
IMS3B <i>on</i>	10 Reading Finish		5
IMS4A <i>on</i>	0 Idle		5
IMS4B <i>on</i>	0 Idle		5
IMS5A <i>on</i>	0 Idle		5
IMS5B <i>on</i>	0 Idle		5
IMS7 <i>on</i>	0 Idle		5

Figure 3.14: ERP-Lab v2.1.0 (SCADA) [67]

The SCADA subsite, allowing Supervisory Control And Data Acquisition, includes a view of all stations, their current state as well as the ID of the carrier, currently at a station (see Fig. 3.14). The stock counter can be increased by clicking on the numbered button in front of the workpiece icons. If an error occurs, it is also portrayed within the stations state column and can be resolved by clicking on the refresh button behind the workpiece icons (see Fig. 3.14).

MES

The MES tab includes all administrative settings of the production plant. Here, the connections of the stations and PLCs, carriers and carrier IDs, stocks, upper stock limits and individual workpieces, as well as the stations states, state machines and actions are defined, allowing to be edited, removed or added. The logic of the entire system and the assignment of the individual workpieces to the corresponding IMS Stations is defined here. Incoming orders can be viewed and deleted from the system as well, if necessary.

ERP

The design of the web shop, the product catalogue, as well as the application of possible marketing measures, represent functionalities and contents, that are available, modifiable

and customizable within the ERP tab. In addition, sales statistics, reports and customer profiles can be reviewed.

Shop

By clicking on “Shop” within the tab bar, the OpenCart web shop opens as a new tab. It is structured like a regular web shop, where a customer can click on a product, set a desired quantity and add it to his shopping cart (see Fig 3.15). By checking out, the workpieces within the shopping cart are ordered, and transmitted to the MES database in order to be produced (see Section 3.1).

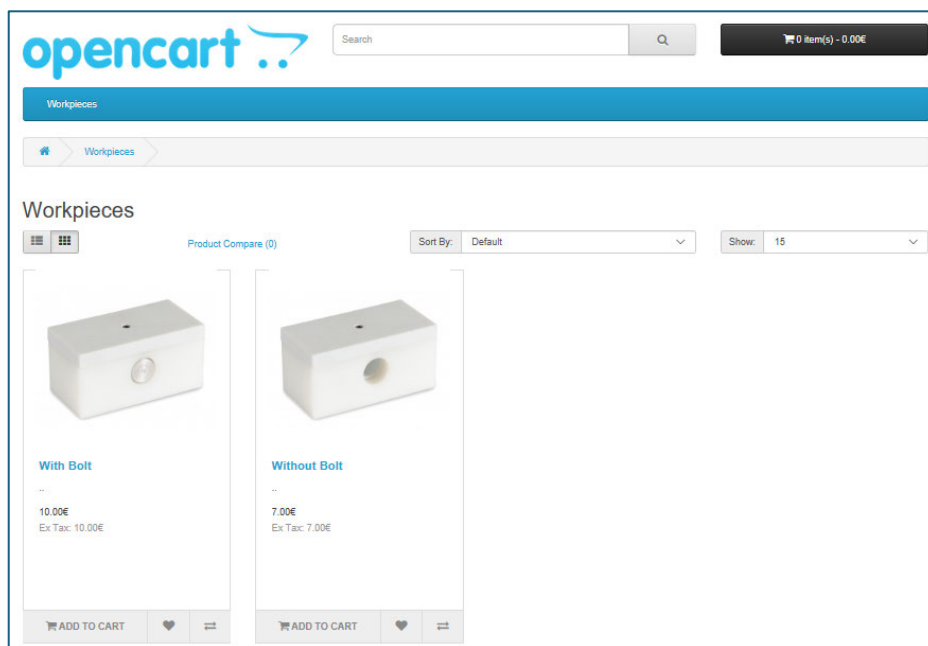


Figure 3.15: OpenCart web shop [67]

3.2.4 The production plant in terms of Industry 4.0

Each IMS Station within the production plant represents a CPS, consisting of mechatronic and network components, able to interact with the physical and digital world through their actuators, sensors, software and information technology (see Section 2.5.1). The IMS Stations are physically and digitally connected, communicating over an Ethernet connection between their PLCs, and virtually represented within the ERP-Labs cloud system, networking the production equipment as their IoT (see Section 2.5.3). The carriers represent smart objects, aware of the workpiece they are carrying, their current state as well as the necessary steps in order to achieve the desired outcome in form of the final product (see Section 2.5.2).

Therefore, the production plant is able to derive actions based on the information available within the IoT and provided by its CPS. Based on this information, the production plant queues and prioritizes certain incoming orders according to the stock count of each individual station, representative of the availability and capability of a station for production or assembly of the desired workpieces and product composition. The production plant is aware of the individual and customized orders from web shops customers, as well as the stock count of its stations, and can derive actions autonomously, organizing and optimizing the production order without manual or human oversight (cf. [57], [58] and [59]). As a result, the production plant can be considered a Smart Factory, conscious and capable of delivering information about its operations (see Section 3.2.3).

In the context of Industry 4.0, all production-relevant data is collected and available, however, no data or information representative of the plant's condition or the condition of its components, necessary for condition-based and predictive maintenance, is gathered or available. The aggregation and availability of this data were achieved in a previous master thesis [59] and are summarized in form of the controller network and data blocks within the TIA Portal (Totally Integrated Automation Portal), which is used to program the PLCs, in Section 3.2.5 below:

3.2.5 Controller network and data blocks

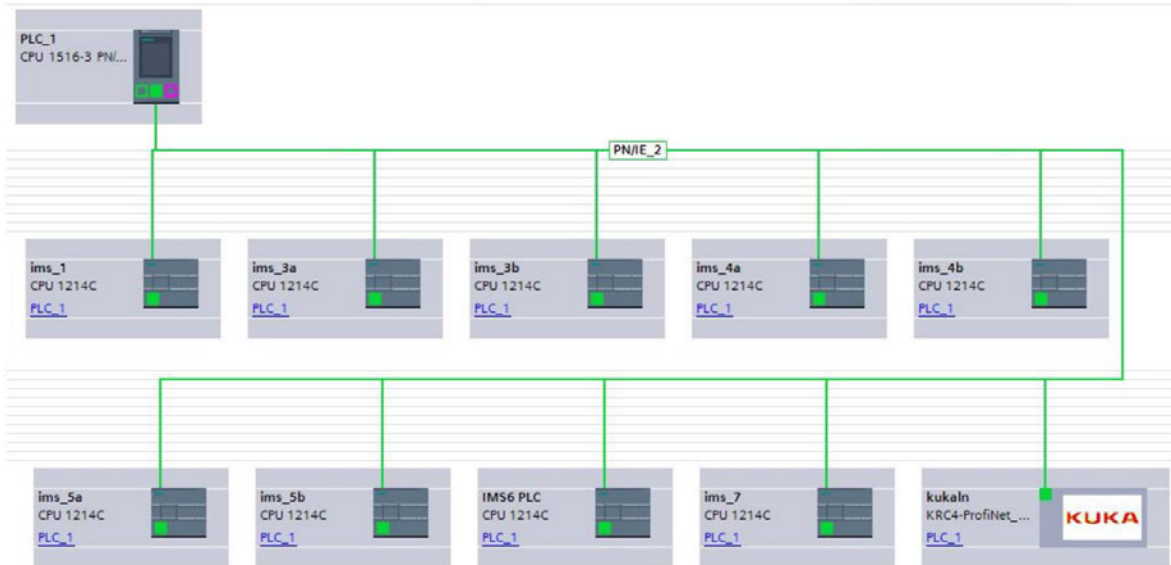


Figure 3.16: TIA Portal network overview showing PLC and KRC4 connections [67]

As previously mentioned, each IMS Station has its own Siemens S7-1200 PLC, which is connected to its sensors and actuators. Additionally, the production plant is equipped with a Siemens S7-1500 PLC, referred to as PLC_1, consisting of ten PLCs in total (see Fig. 3.16). Connected to the PLCs of the IMS Stations as well as the controller of the KUKA robot via Profinet, an Ethernet-based communication protocol, it assumes an overarching role, receiving and aggregating all the data generated by the production plant. Figure 3.17 shows a cropped view of the data blocks, containing the mentioned data, within the TIA Portal:

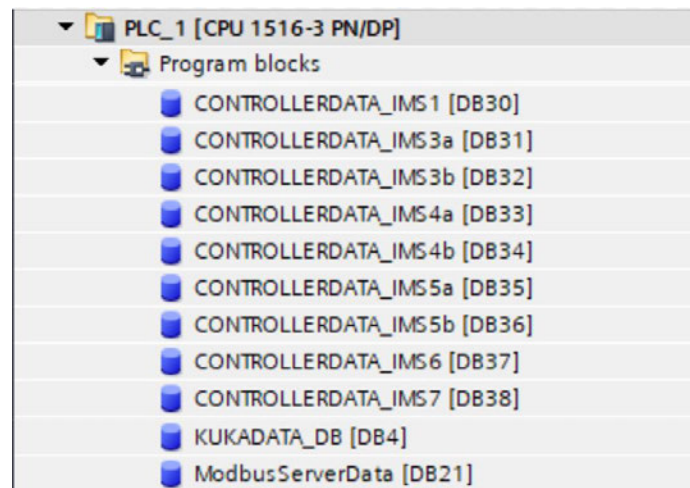


Figure 3.17: Cropped view of PLC_1's data blocks [67]

The data consists of information generated both during the production and the idle states of the individual IMS Stations and the KUKA robot.

Data IMS Stations

For the IMS Stations, this data is generated by the PLCs as a result of the I/O signals shown in Section 3.1.1 (see Tables 3.1 to 3.5), as well as power consumption measurements of each conveyor belt motor. Counters track how often a sensor or actuator is triggered according to the I/O signals, incrementing the count over time with each activation or actuation.

Regarding the power consumption of the conveyor belt motors, the current and voltage are measured, and the resulting power is calculated. Additionally, the runtime of the motors is measured based on their activation. The individual data and the number of the IMS Stations generating it, are presented in Table 3.6:

Table 3.6: Data IMS Stations (adapted from [59])

Name	Content	IMS Stations
MotorCurrent	analog current value [A]	1, 3, 4, 5, 6, 7
MotorVoltage	analog voltage value [V]	1, 3, 4, 5, 6, 7
MotorPower	calculated power [W]	1, 3, 4, 5, 6, 7
motorOperatingHours	motor runtime [min]	1, 3, 4, 5, 6, 7
countSenBeltLeft	counter sensor belt left	1, 3, 4, 5, 6, 7
countSenBeltRight	counter sensor belt left	1, 3, 4, 5, 6, 7
countMotorIsActive	counter belt motor activations	1, 3, 4, 5, 6, 7
countCylStopper	counter stopper cylinder	3, 4, 5, 6, 7
countCylSort	counter sort cylinder	3, 4
countSenStopperTop	counter sensor stopper top	3, 4, 5, 6, 7
countSenMagazine	counter sensor magazine	3, 4, 5, 6, 7
countCylPressing	counter pressing cylinder	5
countSenPressCylNotActuated	counter sensor pressing not actuated	5
countSenPressCylActuated	counter sensor pressing actuated	5
countSenOptBottom	counter optical sensor bottom part	6
countSenOptTop	counter Optical sensor top part	6
countSenInd	counter inductive sensor	6
countSenKap	counter capacitive sensor	6
countCylSwivelTable	counter swivel table cylinder	7
countCylLift	counter lift-cylinder	7
countVacuumValve	counter vacuum valve	7
countSenSwivelTable0	counter sensor swivel table 0°	7
countSenSwivelTable90	counter sensor swivel table 90°	7
countSenVacuumMonitoring	counter sensor vacuum monitoring	7
countSwitchLiftTop	counter switch lift-cylinder top	7

The variables within each data block (see Fig. 3.17) of the individual IMS stations are derived from the names shown in Table 3.6. They use either the suffix “IMSxy” for names in the first three rows of the table or the prefix “IMSxy_” for the remaining rows, where “x” represents the number and “y” the letter of each respective station. For example, *MotorPowerIMS3a* or *IMS3a_countMotorIsActive* for IMS Station 3a.

Data KUKA robot

The data from the KUKA robot represents the current state of each axis and includes measurements of current, motor temperature, torque, and velocity, as shown in Table 3.7 below:

Table 3.7: Data KUKA robot (adapted from [59])

Name	Content
CURR_ACT_Axis	Current [A]
MOT_TEMP_Axis	Motor Temperature [°C]
TORQUE_ACT_Axis	Torque [Nm]
VEL_ACT_Axis	Velocity [rpm]

The KUKA robot's data block consists of 24 parameter variables in total, with each axis number from 1 to six as a suffix to the names listed in Table 3.7.

Combined data block

The last data block shown in Fig. 3.17, *ModbusServerData*, contains all the previously described information, aggregated and reassigned by PLC_1. The variables within this data block are cyclically updated and accessible via OPC UA, enabled by a built-in OPC UA server of the S7-1500 PLC [59].

The described communication and data exchange between PLC_1 and the individual controllers, as well as the creation of the data blocks, variables, and resulting aggregation within *ModbusServerData*, were achieved in [59] by D. Löffler. Although the data is currently available within the described data block and is updated cyclically, there is no recipient or system that receives, stores, monitors, or analyzes this data. The previous approach in [59] involved uploading the data to Machine Advisor, a digital cloud-based service platform by Schneider Electric [60], every 15 minutes. This interval is not suitable for real-time applications and the immediate identification of changing conditions necessary for condition-based and predictive maintenance. Furthermore, the equipment required for this periodic upload is no longer available, both resulting in the development of a new concept or solution.

The previously described situation serves as the starting point for this thesis and will be used to develop a concept for implementing a condition-based and predictive maintenance solution for the Industry 4.0 production plant.

4 Concept

In this chapter, the concept for implementing condition-based and predictive maintenance as well as the underlying necessary requirements and functionalities will be described. First, a functional requirement analysis will be performed, followed by the selection of relevant and necessary tools to achieve the functional requirements or desired functionalities.

4.1 Functional requirements analysis

In order to determine what kind of tools are necessary within the condition-based and predictive maintenance solution, the desired functionalities or functional requirements need to be defined.

At this point, the system is essentially a black box that consists of a collection of different and currently unknown tools providing specific and desired functionalities. These functionalities represent the output of the system or black box and will define its content, with the data from the production plant serving as its input, as illustrated in Fig. 4.1:

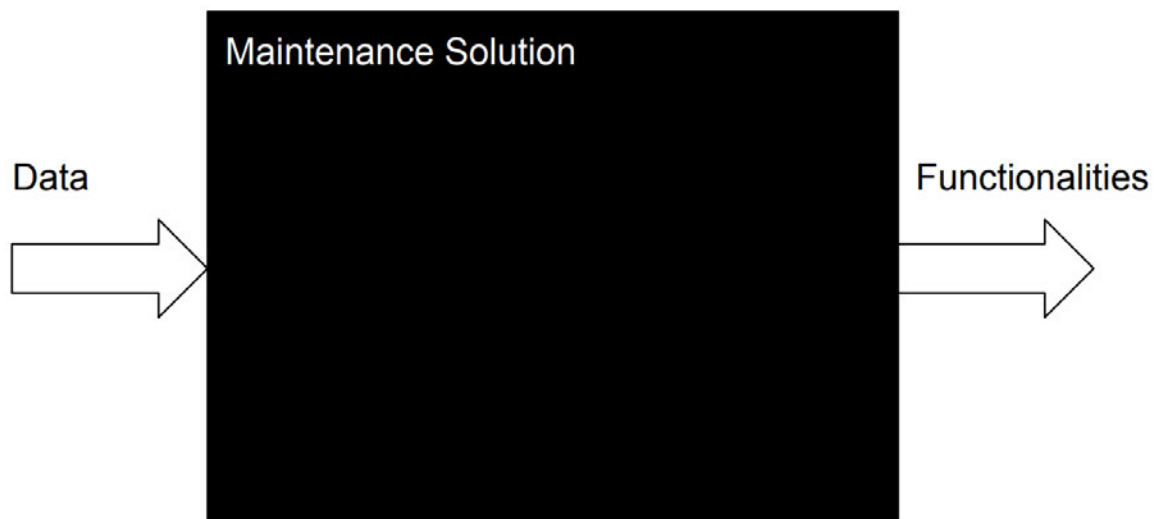


Figure 4.1: Black box of the condition-based and predictive maintenance solution [67]

In order to implement an effective condition-based and predictive maintenance solution, the following functionalities and necessary functional requirements are defined:

1. Accessibility:
 - 1.1 Historical and real-time data
 - 1.2 Independent of physical proximity to the production plant
2. Automated processes:
 - 2.1 Condition monitoring using thresholds
 - 2.2 Data analysis and classification through machine learning
3. Display of:
 - 3.1 Data visualization
 - 3.2 Conditions exceeding thresholds
 - 3.3 Machine Learning output
 - 3.4 Maintenance Alerts
4. Maintenance:
 - 4.1 Alert messages and notifications
 - 4.2 Log data exceeding thresholds
 - 4.3 Log classification

The aim is to provide a general overview of the production plant and its components, enabling operators and maintenance personnel to gain insight into its current state and the condition of its equipment. Operators should be able to inspect data in real-time and, if needed, examine specific parameters of individual stations and components, as well as certain time periods. Simultaneously, the data should be monitored based on specific and defined thresholds, triggering automatic maintenance alerts when necessary. Additionally, relevant data should be analyzed by appropriate machine learning models and classified accordingly. By generating alert messages and notifications when parameters, features or characteristics deviate from normal operation, responsible maintenance personnel should be notified to address potential issues before they lead to failures. Moreover, if maintenance alerts are triggered, either due to parameters exceeding their thresholds or predictive maintenance models identifying deviations, the corresponding data should be logged and retained to maintain a detailed record of all events in case they need to be reexamined at a certain point in the future.

All information should be accessible through an interface and displayed accordingly, independently of one's physical proximity to the production plant, ensuring comprehensive accessibility as well as a detailed overview, if needed. Maintenance personnel should be able to receive notifications, both with and without accessing the interface. In addition, stakeholders and operators should be able to enter and access the interface without needing to be physically present at the production plant.

4.2 Selection of necessary tools

The functionalities, defined in the previous chapter, result in the following tools, necessary for their implementation (see Figure 4.2):

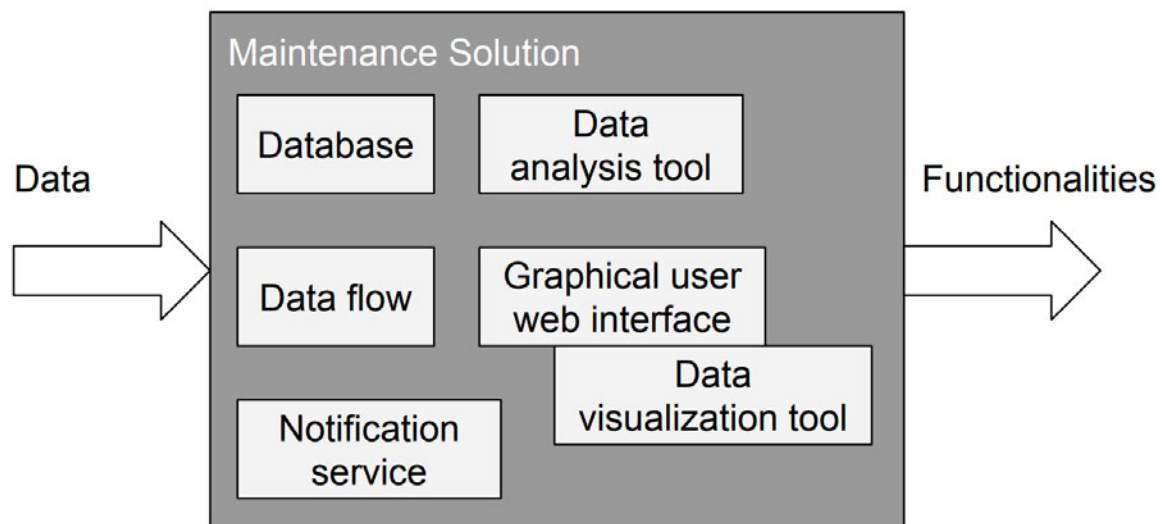


Figure 4.2: Grey box including the necessary tools [67]

To provide insights through historical data, the data generated by the production plant first needs to be transferred from the PLC_1 to a database and stored appropriately, in order to be accessed afterwards at a certain point of time. Additionally, a data analysis tool is required to monitor and process the collected data, in order to provide additional and relevant information necessary for maintenance, such as insights through condition monitoring and machine learning models. Furthermore, to display the information and the collected data effectively and in general, a user interface is necessary. This should be combined with a data visualization tool to create and generate appropriate graphs and provide comprehensive overviews of past and present conditions. As the interface should be accessible independently of

physical proximity to the production plant, a web interface should be implemented. This will ensure that users can access the system from any location, enabling remote monitoring and management, as long as they are connected to the production plants network or intranet.

Moreover, a notification service is necessary, that will notify operators, maintenance personnel or stakeholder both with and without actively accessing the interface. Connecting all previously mentioned tools, an appropriate data flow tool is necessary to access the data and distribute it accordingly. In the following paragraph, appropriate tools are presented and chosen according to their suitability.

Table 4.1: Tool Decision Matrix

Number	MySQL	InfluxDB	Telegraf	Node-RED	TensorFlow	MATLAB	Grafana
1.1	X	X					
1.2		X		X			X
2.1		X		X		X	
2.2					X	X	
3.1		(X)		X		X	X
3.2		(X)		X		X	X
3.3		(X)		X		X	X
3.4				X			X
4.1		X		X			X
4.2	X	X					
4.3	X	X					
5			X	X			

Table 4.1 shows a decision matrix with numbers on the left representing the functional requirements (see Section 4.1) and seven columns representing different tools considered for the application. In addition to the defined functional requirements in Section 4.1, an additional requirement, number 5, was added to represent the ability to manage the necessary data flow. A cross marks a tool that meets the requirement. The columns highlighted in blue

indicate the chosen tools, deemed suitable for this application. In the following paragraph, the decisions will be explained, based on the benefits of each tool, how they complement each other and considering expandability and modularity for implementing additional functionalities as well as an open system architecture allowing integration of new tools and technologies as an additional factor.

Database

Constantly monitoring and collecting data from the Industry 4.0 production plant will result in a large volume of time series data. MySQL and InfluxDB hereby both represent databases capable of receiving and storing data. MySQL is a relational database that uses SQL (Structured Query Language) and requires structured data storage. It can be used for a wide range of applications, including time series data. In contrast, InfluxDB is specifically designed and optimized for handling time series data, able to store the data in columns without requiring complex relational structures and capable of handling high data rates, making it easy and predestined to store a continuous series of time series data for this application. Additionally, all database functionalities can be accessed via an HTTP API (Hypertext Transfer Protocol Application Programming Interface), enabling data access with various software tools [61]. InfluxDB also features a graphical web user interface, allowing the creation of graphs and dashboards, as well as monitoring functionalities for setting thresholds and generating corresponding alarm messages.

Data flow

Telegraf is a server-based agent for collecting and sending data, created by InfluxDB, and able to use OPC UA plug-ins, among others, to communicate and receive data from PLCs [62]. It can be used to transfer the data from PLC_1 to the database but provides no additional functionalities. A better option for managing data flows is Node-RED. Node-RED is a visual and browser-based flow editor, enabling the management of flows and the implementation of functions by connecting and arranging nodes via drag and drop [63].

In addition, Node-RED offers a variety of functionalities by extending its node palette through a library of extensions. These functionalities include creating custom dashboards, displaying notifications and real-time data graphs, sending e-mails, and integrating with various APIs, services, and protocols, such as OPC UA and InfluxDB, allowing different applications to communicate with each other. Furthermore, custom function nodes can be created

with a text editor and JavaScript, enabling users to interact with, manage, and customize data as well as the data flow, and create actions based on custom rules and preferences.

Therefore, the data flow can be automated, and monitoring tasks, as well as the logging of data exceeding thresholds, can be executed in parallel. Node-RED, along with its customizable dashboard as a user interface, can be accessed through a browser and configured as a web interface with additional tabs, views, tabular displays, pop-ups, and buttons, enabling a flexible, comprehensive, and interactive user experience. This makes it suitable for creating maintenance dashboards, where real-time data visualization, alerts, and detailed monitoring are important. Moreover, Node-RED possesses an open system architecture and can be easily extended by adding and integrating further communication protocols, functions, dashboards, and other elements for maintenance or additional applications within the production plant.

Data analysis tool

As a data analysis tool, MATLAB, a platform for numerical calculations, programming, and data analysis, is chosen [64]. In comparison to TensorFlow, a platform for machine learning, MATLAB offers a variety of toolboxes for data preprocessing, feature extraction, and machine learning models, while still retaining the ability to be used for other applications, such as simple data monitoring and visualization tasks or creating complex real-time graphics, models and digital twins for example. This flexibility maintains the potential for expanding condition-based and predictive maintenance solutions.

Data visualization with Grafana

Grafana, a data visualization tool for creating dashboards, offers and provides more advanced visualization possibilities than InfluxDB or Node-RED by themselves. Grafana hereby does not need data to be transferred or separately stored in order to be visualized and can query the data directly from InfluxDB using Flux, InfluxDBs functional data scripting language. Grafana provides a web interface and enables the creation of dashboards, organizing parameters based on categories such as power consumption or according to the stations they belong to. It offers dynamic and customizable visualization options, particularly for historical, real-time, and time series data, as well as additional metrics and filtering options [65].

While Grafana is not essential for the realization of the condition-based and predictive maintenance solution (see Table 4.1), it provides additional functionalities to create visually pleasing and detailed dashboards for examining graphs and trends of data. Therefore, it is used as an additional complement for the application.

Combined maintenance solution

The combined condition-based and predictive maintenance concept for the Industry 4.0 production plant is defined as follows, with the resulting combination of tools summarized in Fig. 4.3:

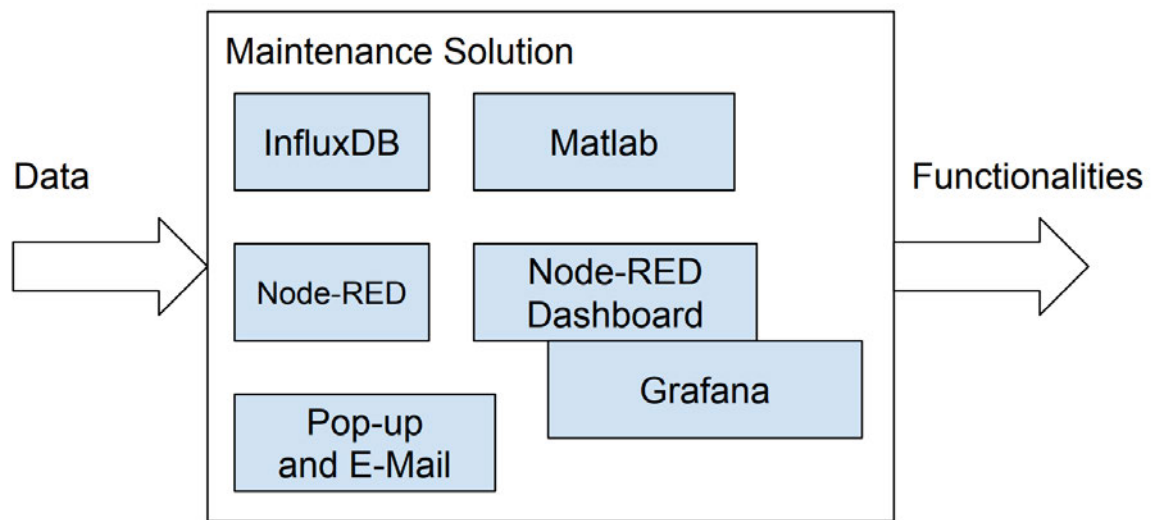


Figure 4.3: White box including the specific combination of tools [67]

Node-RED will be used as the center of the maintenance application, managing the data flow from the production plant to InfluxDB via OPC UA (see Fig. 4.4). As it manages the data flow and already has access to the data before transferring it into the database, thresholds for condition monitoring will be implemented directly within Node-RED, enabling the monitoring and storing of data in parallel. The dashboard of Node-RED will be configured as a user interface to display the data and corresponding maintenance alerts. For notifications, pop-ups within the user interface will be created, and appropriate e-mail nodes will be used to additionally notify maintenance personnel in case they are not currently using the interface (see Fig. 4.3). If notifications need to be disabled, a corresponding function can be implemented.

The data within the database can be accessed by MATLAB and Grafana to analyze and visualize its content (see Fig. 4.4). If maintenance alerts occur, the corresponding data, whether it exceeds a threshold or is the outcome of the predictive maintenance model, will be logged into the database with a corresponding timestamp. As Grafana and InfluxDB also possess web interfaces, they will be linked to Node-RED, and all services will be configured to be ready at startup and accessible through the production plant's network. The relationships between the individual tools are outlined in Fig. 4.4. Unless otherwise specified, the arrows represent data flows or connections established through Node-RED and corresponding nodes:

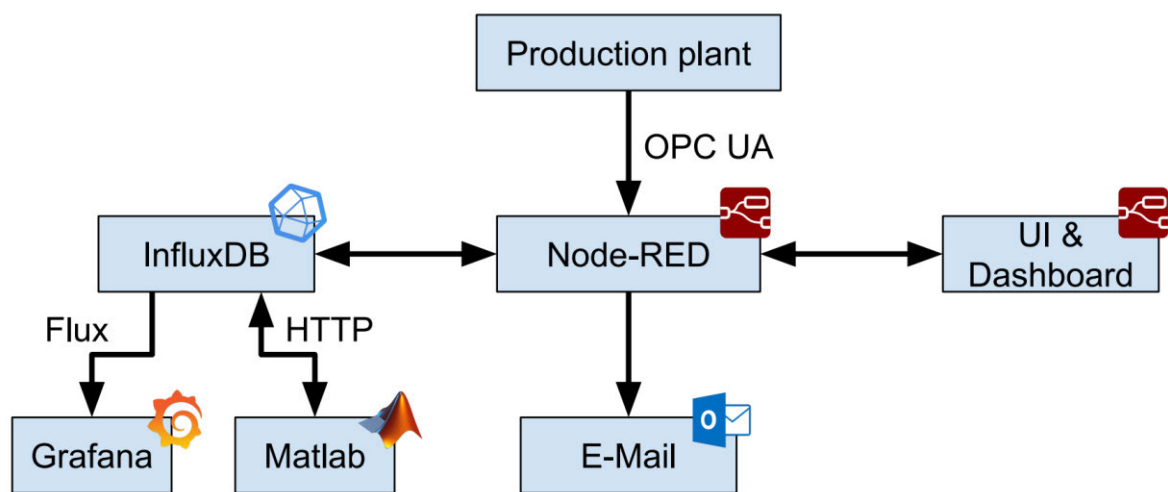


Figure 4.4: Outline of the tool's architecture and data flow [67]

5 Implementation and deployment

In the following chapter, the implementation and deployment of the condition-based and predictive maintenance solution will be described. First, the implementation of the communication and data flow between the production plant, Node-RED, InfluxDB and Grafana will be described, followed by the creation of the user interface and the associated dashboards in Grafana and Node-RED. Afterwards, the condition monitoring and creation of maintenance alert messages and notifications will be presented. Finally, the collected data will be processed in MATLAB, appropriate machine learning algorithms will be trained, and the deployment of a resulting model, as well as the associated data flows and functionalities, will be described.

5.1 Communication and data flow

5.1.1 Prerequisites

First, InfluxDB (v2.7.5), Node-RED (v3.1.7), and Grafana (v10.4.0) are installed to run locally on the Windows PC of the production plant, making them accessible through its network, similarly to the ERP-Lab (see Section 3.2.3). InfluxDB and Node-RED are configured as tasks within the Windows Task Manager to be triggered and initiated at system startup, ensuring that both are automatically online and able to receive and process data when the production plant is active and running. Grafana, which already runs as a Windows service by default, is also active at startup. The addresses of tool each are presented below:

Table 5.1: Local and network addresses of InfluxDB, Node-RED and Grafana [67]

Tool	Local address	Network address	Port
InfluxDB	http://localhost:8086	http://192.168.2.220:8086	8086
Node-RED	http://localhost:1880	http://192.168.2.220:1880	1880
Grafana	http://localhost:3000	http://192.168.2.220:3000	3000

In InfluxDB, the data can be organized in buckets, with each bucket representing a different set of stored and columnized data. After the installation of InfluxDB a separate bucket for

each station of the production plant and the KUKA robot is created to store and access the data in a defined and organized manner.

The nodes in Node-RED are connected through wires, transmitting messages between them. The messages consist of different components or aspects, such as a payload, which mostly consists of parameter values, or additionally assigned names and topics. The nodes react to an incoming message based on its content and the node's specific configuration, processing the data and generating their own messages to pass along the flow. Multiple nodes are organized within flows, with each project, such as this maintenance solution or application, consisting of multiple flows.

5.1.2 Data flow within Node-RED

In the following section, the data flow between the production plant, specifically PLC_1, through and within Node-RED to InfluxDB will be described. An overview of the initial part of the corresponding configuration is shown in Fig. 5.1 below:

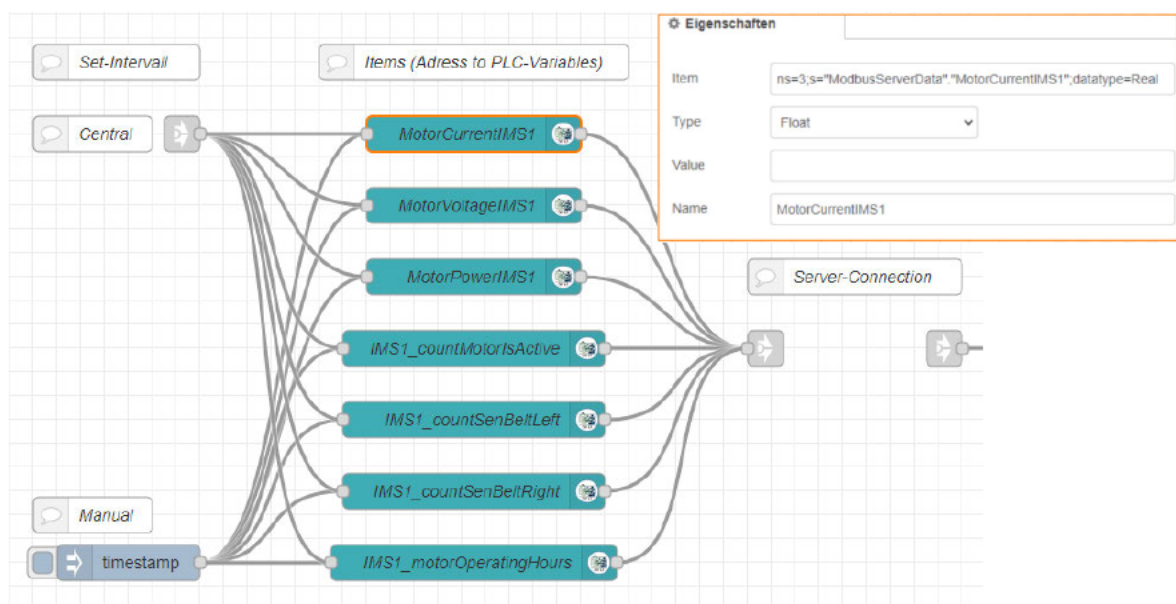


Figure 5.1: OPC UA Item nodes within Node-RED [67]

Figure 5.1 shows a selection of nodes within Node-RED. The majority of these nodes, shown in the middle of Fig. 5.1, are so-called OPC UA item nodes. The square grey nodes represent link nodes, linking nodes between different flows in Node-RED, organized as tabs within its editor workspace. The mostly white nodes with a speech bubble icon represent comments,

providing descriptions or notes within the flow. The node shown at the bottom left, labeled "timestamp," is an inject node. This node periodically or manually injects a timestamp as a payload to the nodes it is connected or wired to, causing a reaction, such as accessing and reading an OPC UA item. In this example, each OPC UA item node represents one parameter of the data generated by IMS Station 1. The parameters and their underlying information can be accessed by providing their respective and individual addresses within the data block of PLC_1, as shown in the upper right window in Fig. 5.1.

The addresses represent a combination of the data block name and the name of the specific parameter to be accessed or read. For the production plant or PLC_1, these consist of the parameter names within the data block *ModbusServerData*, with all parameters previously summarized in Section 3.2.5 and listed in Tables 3.6 and 3.7. In this example, it represents the address of the parameter *MotorCurrentIMS1*, as shown in Fig. 5.1, with its node and properties window both highlighted in orange.

The address is entered in the "Item" field of the properties window (see Fig. 5.1). First, the namespace (ns) is defined, which is 3 for all items within this data block. Afterwards, separated by a semicolon, the string (s) with the full address of the item is entered. At last, the data type is specified, which in this case, represents a "Real" for the motor current of IMS Station 1. If the data type is specified within the "Item" field, the content of the field "Type" below is not considered by the node. Additionally, a name for the node can be assigned, in this case, it is defined as the name of the corresponding variable (see Fig. 5.1).

Each OPC UA item node is wired a link node on the upper left, connected with a central timestamp inject node, and a link node on the right, connected to a central OPC UA client (see Fig. 5.1 and Fig. 5.2). Both nodes are located in another flow, which is referred to as "central", and shown in Fig. 5.2:

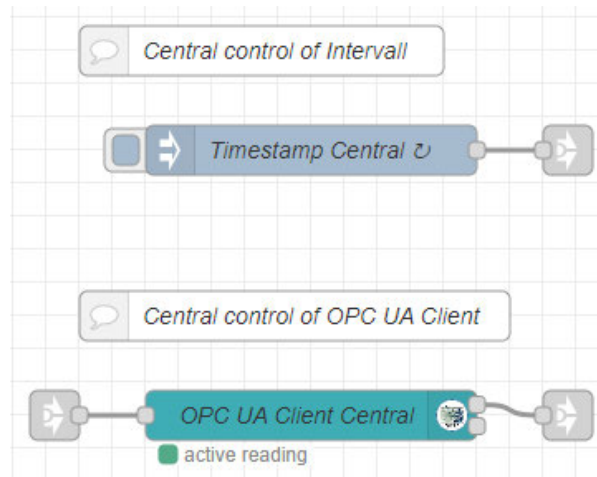


Figure 5.2: Central timestamp and OPC UA Client [67]

The central timestamp controls the periodically injected payload for all item nodes. The central OPC UA client is connected to the built-in OPC UA server of PLC_1 and enables access to its items, representing the individual parameters within *ModbusServerData*. The settings of the client are shown below (see Fig. 5.3):

Figure 5.3: Settings of OPC UA Client Central [67]

The endpoint represents the address of the OPC UA server and matches the corresponding settings within the TIA Portal (see Fig. 5.4):

Figure 5.4: Addresses of the OPC UA server [67]

The client is set to “READ” (see Fig. 5.3), therefore reading each variable periodically according to the interval set by the central timestamp. It is set to one second in order to meet the requirements of real-time data accessibility but can be adapted if necessary. To actively read the data (see Fig. 5.2), the specification of a particular certificate is not necessary (see Fig. 5.3).

The output of the central timestamp node represents the input for the OPC UA items, while the output of these items represents the input for the central OPC UA client (see Fig. 5.1 and 5.2). The server's output, which represents the read data, is connected to the link node via an outgoing wire (see Fig. 5.2). This link node's input forwards the accessed information, through an invisible wire, to the output of the link node within the data flow of IMS Station 1, located on the far right of Fig. 5.1, with a single outgoing wire.

Subsequently, this link node's output is wired to a switch that assigns the parameters to the corresponding buckets of their respective stations based on their names (see Fig. 5.5). These buckets, previously described in Section 5.1.1, are represented by beige-colored “InfluxDB-out” nodes with the InfluxDB icon on their right end (see Fig. 4.4 and 5.5). The names of these nodes consist of the local InfluxDB address as well as the respective parameter names.

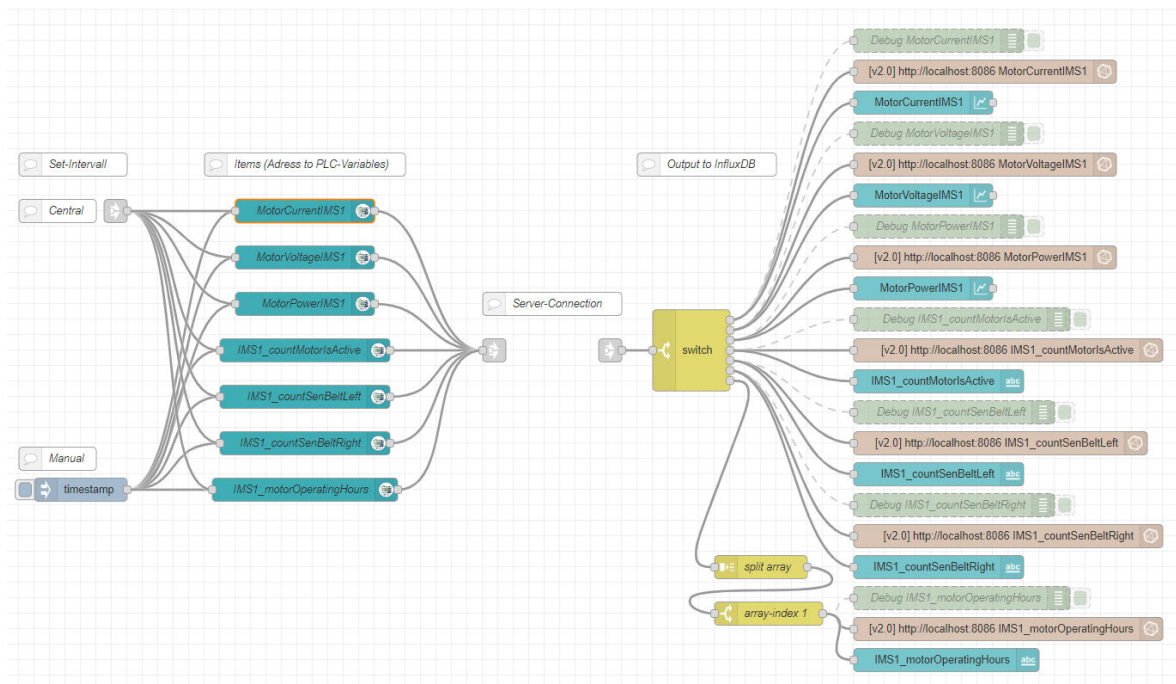


Figure 5.5: Data flow of IMS Station 1 [67]

Figure 5.5 shows an overview of the entire data flow for IMS Station 1, with the previously described OPC UA items on the left and the switch along with the InfluxDB-out nodes on the right. The OPC UA client (see Fig. 5.2) can be imagined as bridging the gap between the two link nodes in the middle of the flow, labeled "Server-Connection." The green nodes are debug nodes, while the blue nodes positioned between the debug nodes and the InfluxDB-out nodes are part of the user interface, representing dashboard UI nodes. These will be further explained in Section 5.3. The parameter *IMS1_motorOperatingHours* is interpreted as an array, with the desired information located at index 1. The data is split, and the information at array index 1 is assigned to the appropriate InfluxDB-out node of the corresponding bucket accordingly (see Fig. 5.5).

The InfluxDB-out nodes are configured as follows:

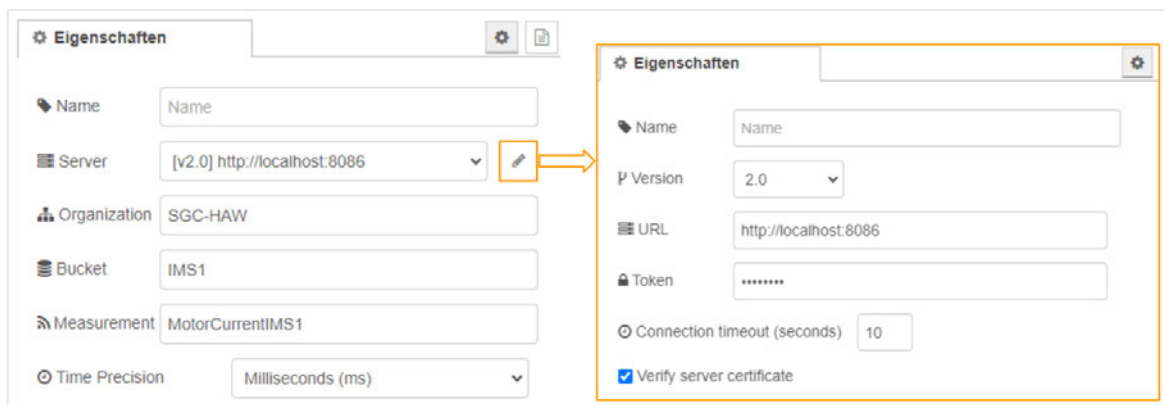


Figure 5.6: Settings of the InfluxDB-out node [67]

The connection to the server can be configured using the icon on the right of the corresponding field. In the associated subwindow, outlined in orange (see Fig. 5.6), the InfluxDB version and its address can be specified. To establish the connection and finalize the settings, a specific API token, which can be generated via the InfluxDB web interface, is assigned. Afterwards, the subwindow can be saved and closed, and the settings can be continued within the initial window.

Following the server configuration, the organization where the data will be stored is specified. Organizations represent different databases within InfluxDB, which can be accessed separately and contain their own set of buckets. For this application, all buckets are located within the same organization. After specifying the organization (see Fig. 5.6), the bucket

where the parameter will be stored is indicated, as well as the specific measurement it refers to, allowing for later access to the parameters under their specific measurement name within their bucket. When each individual measurement of the time series data is saved, a timestamp is assigned to it. The time precision, which indicates how precise the timestamp will be from seconds to nanoseconds, is set to milliseconds (see Fig. 5.6).

Completing these settings allows the data, accessed and read via OPC UA from the production plant, to be directed through the flow depicted in Fig. 5.5 into the InfluxDB database and its corresponding bucket for IMS Station 1.

Analogous to the process described for implementing the data flow for IMS Station 1, separate flows are created for each of the other IMS Stations as well as the KUKA robot (see Appendix A.1). All the data is then stored and can be viewed within InfluxDB as follows:

5.1.3 Storing the data in InfluxDB

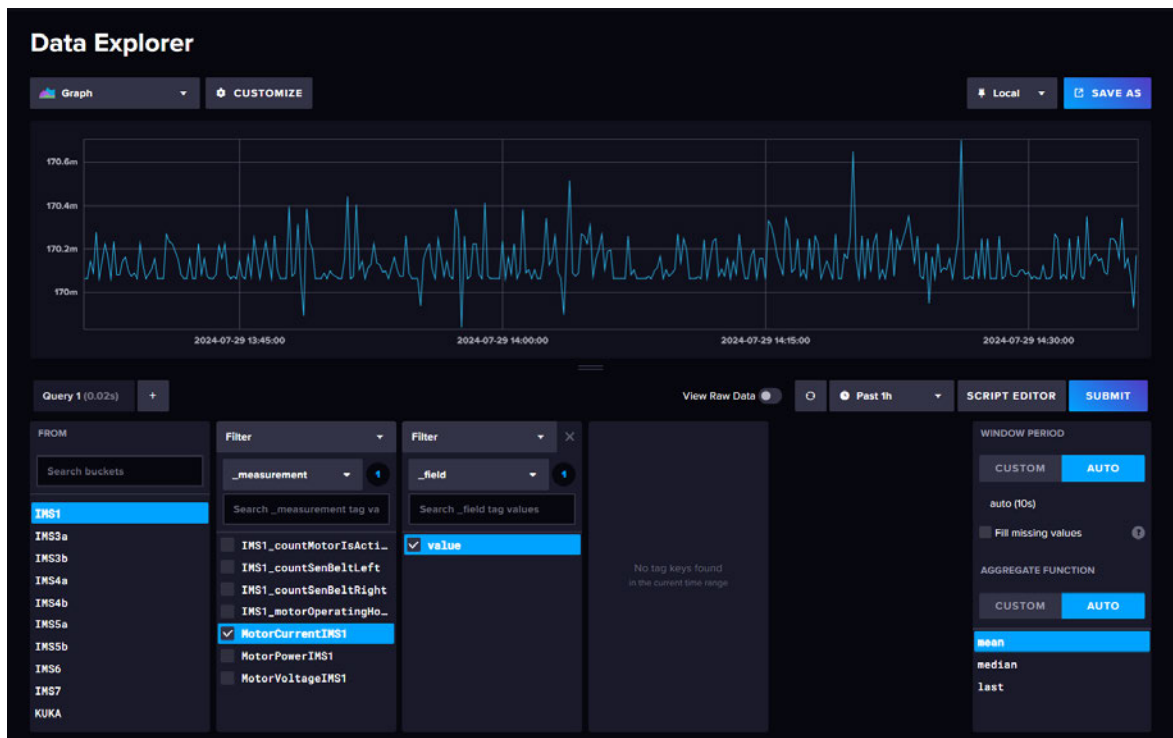


Figure 5.7: Stored data within InfluxDB [67]

Figure 5.7 shows the Data Explorer view of InfluxDB's web interface, which allows users to explore the stored data within its respective buckets over a selected period. In the first

selection list, located at the bottom left of the Data Explorer, all the buckets from various stations, including the KUKA robot, are listed and can be selected. The second selection list enables users to choose one or multiple measurements from the selected bucket to be displayed on the screen above. If the measurements contain multiple fields, they can be selected from the third list of options (see Fig. 5.7).

By enabling the “View Raw Data” switch, the data can be displayed in a tabular format instead of as a graph or other form of visualization. The “Query Builder” button reveals an automatically generated Flux query for the selected bucket, measurement, and field. Both of these functionalities are illustrated in Fig. 5.8 below:

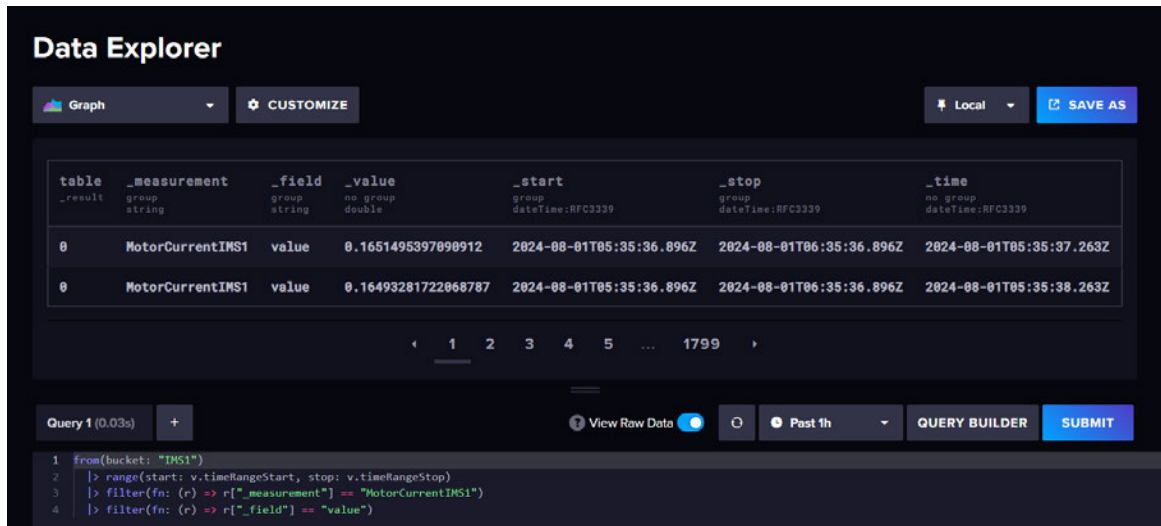


Figure 5.8: View of the tabular data and Flux query within InfluxDB [67]

If wanted or needed, the raw data can be downloaded as a CSV file for further examination and analysis using other tools, such as MATLAB, or for sharing and uploading to other platforms.

5.1.4 Accessing the data in Grafana

The function for automatically generating queries is utilized to create a series of custom Flux queries within InfluxDB, which are then transferred to Grafana. These queries are used and employed to visualize the data within Grafana's web interface, enabling the creation of more in-depth and detailed views of the individual stations, their parameters, and the data overall.

First, InfluxDB is added as a data source in Grafana by providing its local address, as shown in Fig. 5.9:

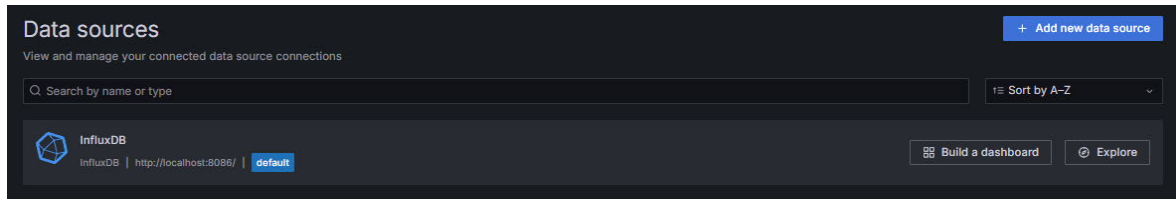


Figure 5.9: InfluxDB as a data source within Grafana [67]

Hereby, the query language is set to Flux, to guarantee that the automatically generated Flux queries from InfluxDB can be utilized and processed within Grafana (see Fig. 5.10).

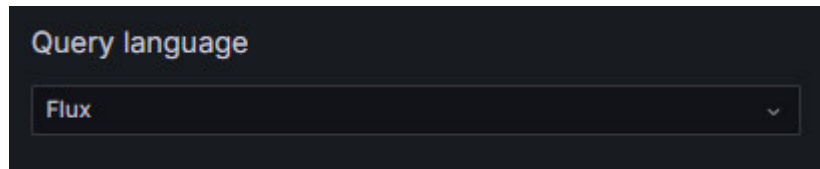


Figure 5.10: Query language setting in Grafana [67]

Within this application, InfluxDB serves as the sole data source for visualizing and organizing the data in the context of condition-based and predictive maintenance. However, it can be expanded in the future to include additional data sources. The process of implementing and creating graphs and dashboards within Grafana using the automatically generated Flux queries, will be described in Section 5.3.2.

5.2 Condition Monitoring and Alert Management

Within this section, condition monitoring using the data within Node-RED will be established, followed by the creation of alert messages and notifications. These alerts will be configured to be automatically generated and triggered by exceeding predefined thresholds to inform maintenance personnel. Additionally, the automated logging of the alert data within the database will be explained, along with the creation and management of alert tables to display the data in a structured format. Finally, a silencer function will be introduced to manage notification interruptions, allowing maintenance personnel to temporarily disable alerts directly from the user interface.

5.2.1 Condition monitoring

Through the already established data flow from PLC_1 to InfluxDB, the data already exists within the flows of Node-RED and only needs to be accessed and rerouted in parallel to establish thresholds and achieve condition monitoring. For this purpose, a new and separate flow to interact with the data is created, as shown in Fig. 5.11 below:

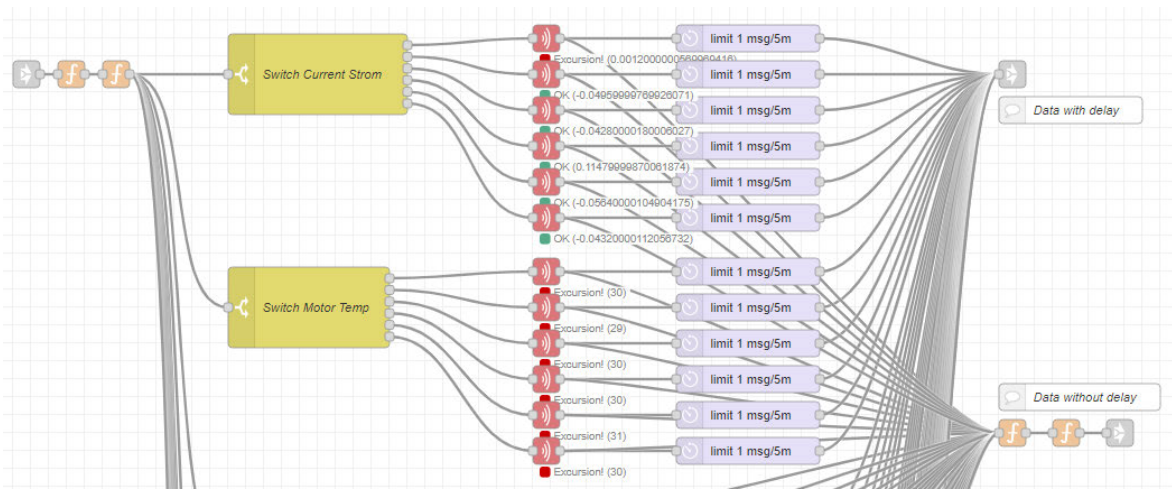


Figure 5.11: Condition monitoring flow

The data within this flow is provided by the link node shown in the upper left of Fig. 5.11. It is connected to the output of the central OPC UA client, similar to the previously described flow of the IMS Station 1 (see Section 5.1.2 and Fig. 5.5). The link node is then wired to switches that separate each variable of parameters to be monitored. Hereby, only the time series data is monitored, as it provides the most valuable information for the condition of the production plant. However, this concept can easily be expanded to include the counters of each station.

Since this process is applied to every time series parameter of the production plant, only an excerpt of this flow is depicted in Fig. 5.11, showing the switches for the axis current and temperature of the KUKA robot. Two additional switches handle the torque and velocity data of the six axes, and nine more switches are dedicated to the individual IMS Stations. The entire flow can be found in the appendix (see Appendix A.2). Figure 5.12 illustrates the switch for IMS Station 4a, serving as an example for the switches of all stations.

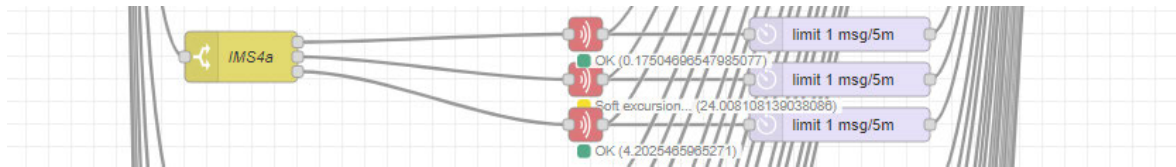


Figure 5.12: Condition monitoring of IMS Station 4a [67]

The squared nodes between the switch and link node are function nodes, positioned solely to bundle and pass through the data without altering it. These nodes are useful for inserting additional nodes between the switches and the link node without needing to renew or reconnect all wires (see Fig. 5.11).

After the variables are isolated and separated from the collective data pool by the switches, each individual parameter is routed to so-called excursion nodes (see Fig. 5.11). These nodes enable the definition of hard and soft upper and lower thresholds, as shown in Fig. 5.13 below:

Figure 5.13: Settings of an excursion node [67]

Figure 5.13 shows an excursion node for the temperature monitoring of an axis motor of the KUKA robot, configured to trigger an excursion if the temperature falls below 15°C or exceeds 35°C. The time setting in the configuration specifies the number of seconds the temperature must remain outside these soft thresholds (either minimum or maximum) before the node triggers an excursion. For hard thresholds, the excursion node reacts immediately,

regardless of the time setting. When the excursion node triggers, it forwards the value of the corresponding parameter as a payload to the following node connected to its output. This means that the messages or information distributed by the switch nodes and sent to the excursion nodes, are held and monitored there until a threshold is crossed. Once a threshold is exceeded, the relevant information, i.e., the parameter value, is forwarded through the output and wire of the excursion node.

From this point, there are two parallel and separate paths within the flow. As depicted in Fig. 5.11, one bundle of wires leads directly to a link node labeled "data without delay", while another bundle is connected to a link node labeled "data with delay". The latter includes an additional node wired between the outputs of the excursion nodes and the input of the corresponding link node. This additional node, as indicated by its label, introduces a delay that limits the message throughput. Its settings are shown below in Fig. 5.14:

The screenshot shows the 'Edit delay node' configuration window. At the top, there are 'Delete', 'Cancel', and 'Done' buttons. Below is a 'Properties' section with a settings icon. The 'Action' is set to 'Rate Limit' and 'All messages'. The 'Rate' is set to '1 msg(s) per 5 Minutes'. There is an unchecked checkbox for 'allow msg.rate (in ms) to override rate'. The 'Drop intermediate messages' option is selected. A 'Name' field is at the bottom.

Figure 5.14: Settings of the delay node [67]

The delay node is configured to limit all incoming messages, in this case, the parameter values forwarded by the triggered excursion node, to one message per five minutes while dropping all intermediate messages. This setup ensures that if an excursion condition persists and continuously sends the parameter value, alerts, and notifications (explained in Section 5.2.2) are not triggered, displayed, or sent every second. At the same time, it is important to maintain real-time data updates on the dashboard rather than updating only every five minutes. To address this, the two previously mentioned distinct data flow solutions are

established: one with a delay and one without a delay. Their link nodes, separately connected to their respective bundle of nodes and wires, forward the parameters to the central flow, where maintenance alerts and notifications will be generated according to the forwarded information. This process, along with the creation of tables for displaying threshold-exceeding data, will be explained in the following section:

5.2.2 Maintenance alerts, notifications and alert-data

The configuration of maintenance alerts and notifications is integrated within the central flow, positioned below the central control for the reading interval through the central timestamp and the central control of the OPC UA client (see Fig. 5.2), in order to allow a centralized management of notifications (see Fig. 5.15).

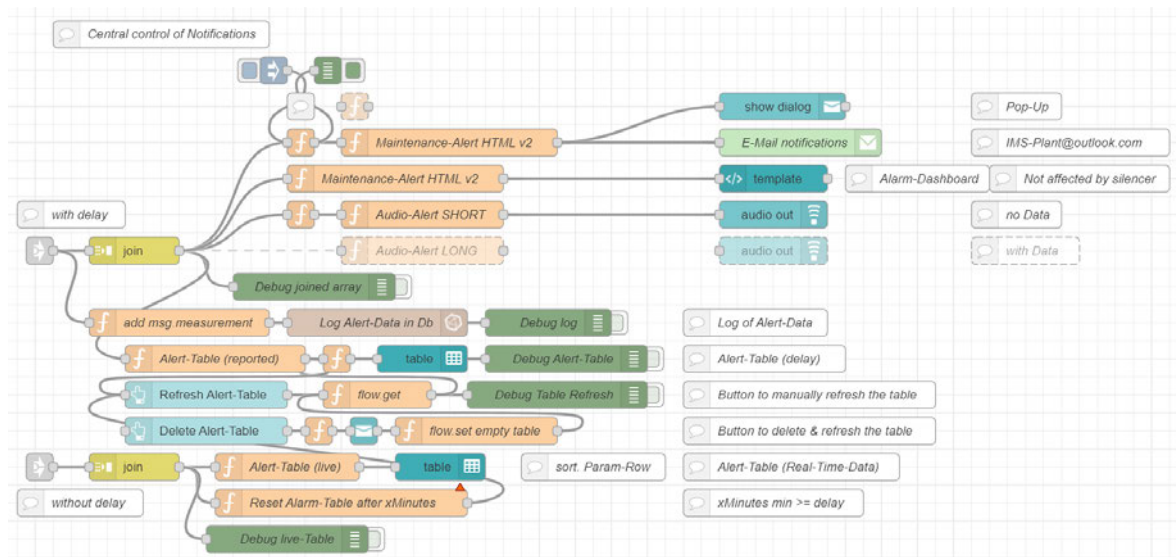


Figure 5.15: Central control of notifications [67]

The entire configuration of maintenance alerts and notifications, as well as the creation of tables for displaying threshold-exceeding data, is shown in Fig. 5.15. Since this configuration includes custom function nodes and is more complex and compact compared to the previously described flows, it will be explained in smaller, separate sections, addressing one functionality or aspect at a time.

The central control of notifications also includes functionalities related to the user interface and dashboard, represented by the different blue-colored nodes within Fig. 5.15. In this section, only the underlying functionalities regarding the necessary data flow and function

nodes for maintenance alerts and notifications are explained. The user interface and dashboards, including the integration and presentation of these functionalities, will be described in Section 5.3.

Notifications and pop-ups

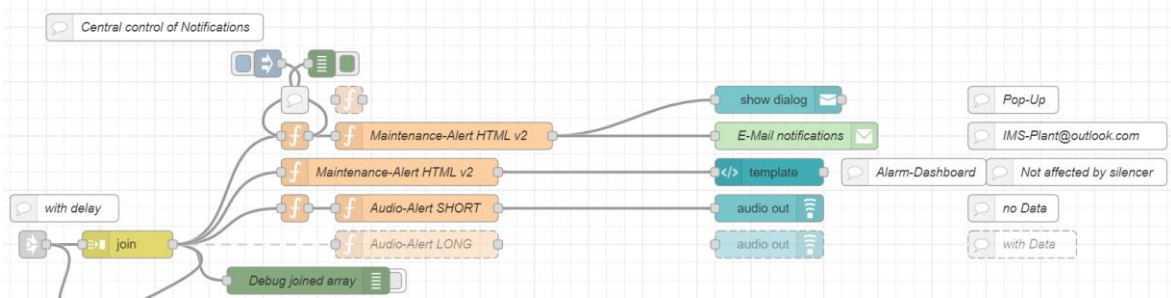


Figure 5.16: Creation of maintenance alerts and pop-up notifications [67]

The link node, shown at the beginning of the flow in Fig 5.16 (bottom left), is connected to the link node labeled "data with delay" within the condition monitoring flow, previously described in Section 5.2.1. It is wired to a join node, which merges all previously separated parameters, previously split by the switch nodes for individual threshold monitoring, (see Fig. 5.11 and 5.12), into a single array.

The purpose of merging the parameters through the join node into an array is to ensure that multiple simultaneous threshold excursions do not result in separate notifications or maintenance alerts for each parameter of every station or the KUKA robot. This approach collects all relevant information within the array, allowing a collective processing and alert generation. This helps to reduce data traffic and notification noise, streamlining the maintenance alert system. Through the output of the join node, the data is forwarded to three distinct sets of nodes, each organized in parallel linear paths, with function nodes directly following the join node, and a subsequent method of notification or display. Below these nodes, a disabled set of nodes, identifiable by the grayed-out and dashed lines, as well as a debug node for the joined array, is included (see Fig. 5.16).

The first set of horizontally arranged nodes, starting from the top, is configured to create a text message in HTML via the function node "Maintenance-Alert HTML v2" and forward it to an e-mail node and a notification node, both denoted by an envelope icon (see Fig. 5.16).

The function node contains the following logic to create an automated message based on the array of inputs from the triggered excursion nodes:

The code is hereby altered to present a short example of the automated message generation for one parameter set, specifically the axis motor temperature of the KUKA robot (see Fig. 5.17 to 5.19).

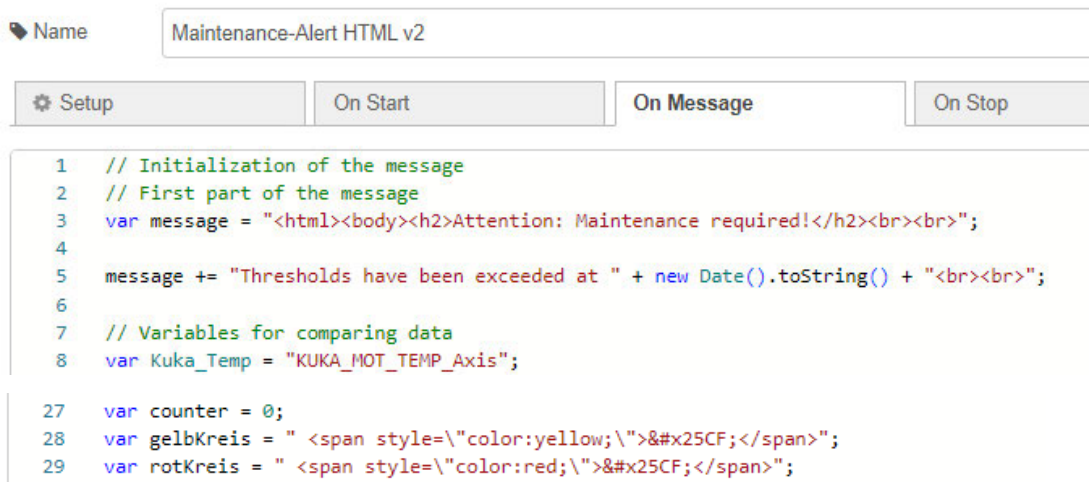


Figure 5.17: Initialization of variables [67]

Fig. 5.17 shows the settings of the function node as well as a part of its code. The function node is configured to execute this code when a message is forwarded to its input. First, the beginning of the alert message is initialized, followed by variables to be used for a later comparison, a counter, and a method to create yellow and red circles to visually differentiate hard excursions (red) from light excursions (yellow) within the messages (see Fig. 5.17).


```

37 // Iterate over each message in the array
38 msg.payload.forEach(function (msgItem) {
39
40     // Kuka Temperature
41     if (msgItem.browseName.includes(Kuka_Temp)) {
42
43         // Check if message meets conditions (hard warning)
44         if (msgItem.payload > 40 || msgItem.payload < 10) {
45
46             // Add a line break to the message when first entering the condition
47             if (counter !== 1) { message += "<br>"; }
48             counter = 1;
49
50             // Add the message to the message string
51             // Main part of the message
52             message += msgItem.browseName + ": " + msgItem.payload + " °C - hard warning" + rotKreis + "<br>";
53         } else { // light warning previously triggered by excursion-node
54
55             // Add a line break to the message when first entering the condition
56             if (counter !== 1) { message += "<br>"; }
57             counter = 1;
58
59             // Add the message to the message string
60             // Main part of the message
61             message += msgItem.browseName + ": " + msgItem.payload + " °C - light warning" + gelbKreis + "<br>";
62         }
63     }
64 }

```

Figure 5.18: Iteration over each Item within the array [67]

Afterwards, the function node iterates over each item in the array that includes the measurement or parameter names and compares it to the previously initialized variables. In the code example shown in Fig. 5.18, it checks whether the message item contains the partial name of an axis motor temperature parameter, stored in the variable *Kuka_Temp* (see Fig. 5.18, line 41). If this condition is met, the second if condition checks whether the excursion is a hard or light excursion. Subsequently, in both cases, the previously initialized counter is checked for inequality (see Fig. 5.17 and 5.18), and a line break is added to the message if it is not equal to the specific counter for the axis motor temperature. The counter is then updated to the appropriate value, ensuring that the message is organized by parameter groups (see Fig. 5.18, line 48). If the next message item also contains the temperature of an axis motor, no line break is inserted.

Then, an additional part is added to the alert message, which includes the name of the measurement, the associated value, and its unit, followed by the type of excursion and the colored circle for visual distinction (see Fig. 5.18, lines 52 and 61). The function node iterates over each item in the input array that includes the measurement or parameter names and repeats this process for all parameter categories of the robot axes and individual IMS Stations. It separates each category within the alert message by adding line breaks, for example, transitioning from axis temperature to velocity.

```

660      // Last part of the message
661      message +=
662          "<br><br>" +
663          "Please take necessary actions to address or evaluate this issue." +
664          "<br><br>" +
665          "Thank you for your attention" +
666          "<br>" +
667          "Your IMS-Station at USST" +
668          "</body></html>";
669
670      // Send Message String
671      msg = {
672          payload: message,
673          topic: "Maintenance-Alert"
674      }
675
676      if (counter !== 0) { return msg; }

```

Figure 5.19: Adding the last part of the alert message [67]

At the end of the function node, the text message is finalized with a concluding statement, and the completed message is forwarded through the function node's output (see Fig. 5.19). Connected to the function nodes output, the finalized text message is then processed by the e-mail node (see Fig. 5.20, left) and the notification node (see Fig. 5.20, right) according to their respective settings.

<div> <div>Name</div> <div>E-Mail notifications</div> </div> <div> <div>To</div> <div>wdx972@haw-hamburg.de</div> </div> <div> <div>Server</div> <div>smtp-mail.outlook.com</div> </div> <div> <div>Port</div> <div>587</div> <div><input type="checkbox"/> Use secure connection.</div> </div> <div> <div>Auth type</div> <div>Basic</div> </div> <div> <div>Userid</div> <div>IMS-Plant@outlook.com</div> </div> <div> <div>Password</div> <div>.....</div> </div> <div> <div>TLS option</div> <div><input checked="" type="checkbox"/> Check server certificate is valid</div> </div>	<div> <div>Layout</div> <div>OK / Cancel Dialog</div> </div> <div> <div>Send to all browser sessions.</div> <div><input checked="" type="checkbox"/></div> </div> <div> <div>Default action label</div> <div>OK</div> </div> <div> <div>Secondary action label</div> <div>(optional label for Cancel button)</div> </div> <div> <div>Accept raw HTML/JavaScript input in msg.payload to format popup.</div> <div><input checked="" type="checkbox"/></div> </div> <div> <div>Class</div> <div>[msg.className]</div> </div> <div> <div>Topic</div> <div>[msg.topic]</div> </div> <div> <div>Name</div> <div>Name</div> </div>
--	--

Figure 5.20: Settings of e-mail node (left) and notification node (right) [67]

For alert message notifications via e-mail, a dedicated email address for the production plant has been created in Outlook (see Fig. 5.20, Userid). The email node is configured to use Outlook's SMTP server (Simple Mail Transfer Protocol server), including its appropriate port, to send e-mails from the production plant's e-mail address to the maintenance personnel, by specifying the recipient's e-mail address (see Fig. 5.20, left). The notification node is configured to display a pop-up on the user interface that must be acknowledged in order to

disappear. It is set to accept raw HTML/JavaScript inputs to format the pop-up, enabling it to process the output from the function nodes (see Fig. 5.20, right).

The same function node is also used to display the generated text permanently in the user interface through a template node, as shown in second set of horizontally arranged nodes in Fig 5.16. The third function node below, labeled “Audio-Alert SHORT”, is used to create a short message that, together with the audio node, provides an acoustic alert to the maintenance personnel near the production plant and its PC, signaling that a threshold has been exceeded. The content of this function node is presented in Fig. 5.21.

```

2  var message =
3
4  "Attention: Thresholds have been exceeded. Please check the alert-dashboard and take necessary actions.";
5
6      msg = {
7          payload: message,
8          topic: "Maintenance-Alert"
9      };
10
11 return msg;

```

Figure 5.21: Function for audio node [67]

The specified text is then converted to speech by the audio node and played through the PC’s speakers using an automatically generated computer voice. The disabled function node, labeled “Audio-Alert LONG,” contains a longer version including details of the specific threshold-exceeding parameters. This node, along with its corresponding audio node, can be uncommented if needed (see Fig. 5.16).

Logging alert-data into the database

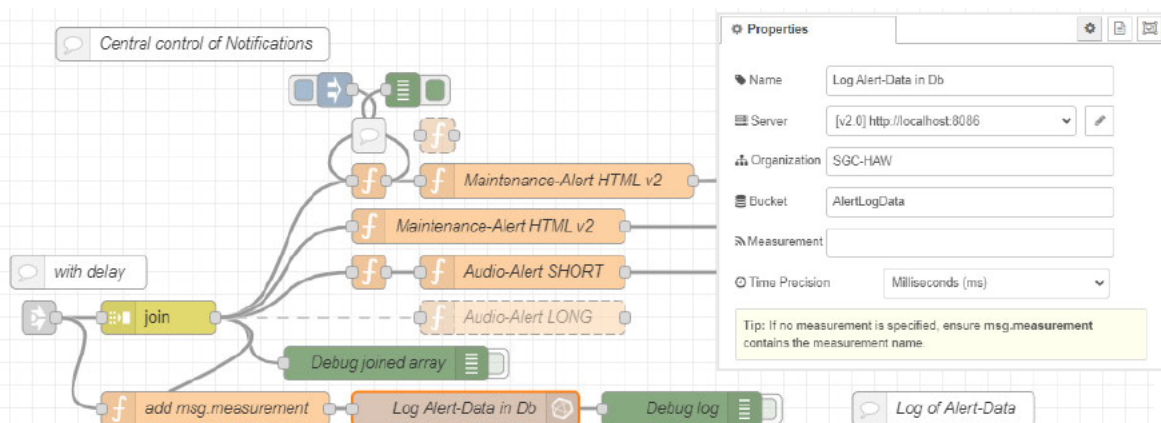


Figure 5.22: Logging alert-data into the database [67]

The next functionality implemented within the central control of notifications is the logging of alert data, triggered through the condition monitoring and highlighted in Fig. 5.22 by the orange-outlined InfluxDB node. This node receives the individual parameters from the excursion nodes at the time of the excursion without being merged. It stores the data under their measurement name, assigned by the preceding function node, within the “AlertLog-Data” bucket, specifically created for this purpose, in the InfluxDB database (see Fig. 5.22, right). This ensures that previous excursions are recorded and accessible for future reviews.

Creating alert-tables

Equally important as maintenance alerts and notifications, whether acoustic, visual, or to maintenance personnel that is not in close proximity to the workstation, is providing an overview of active maintenance alerts, to enable a quick assessment of the current state of the production plant as well as its components. Alerts and corresponding information about threshold-exceeding parameters, including the value, time, and warning level, must be visible and easily accessible to operators or maintenance personnel, allowing them to react to the current situation accordingly. The remaining function nodes implemented to create such an overview, are presented in Fig. 5.23 (cp. Fig. 5.22 and 5.23).

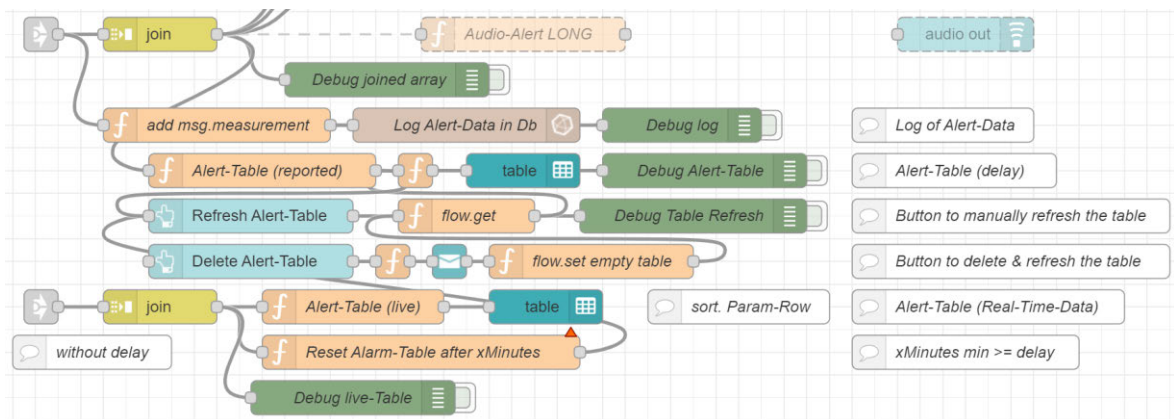
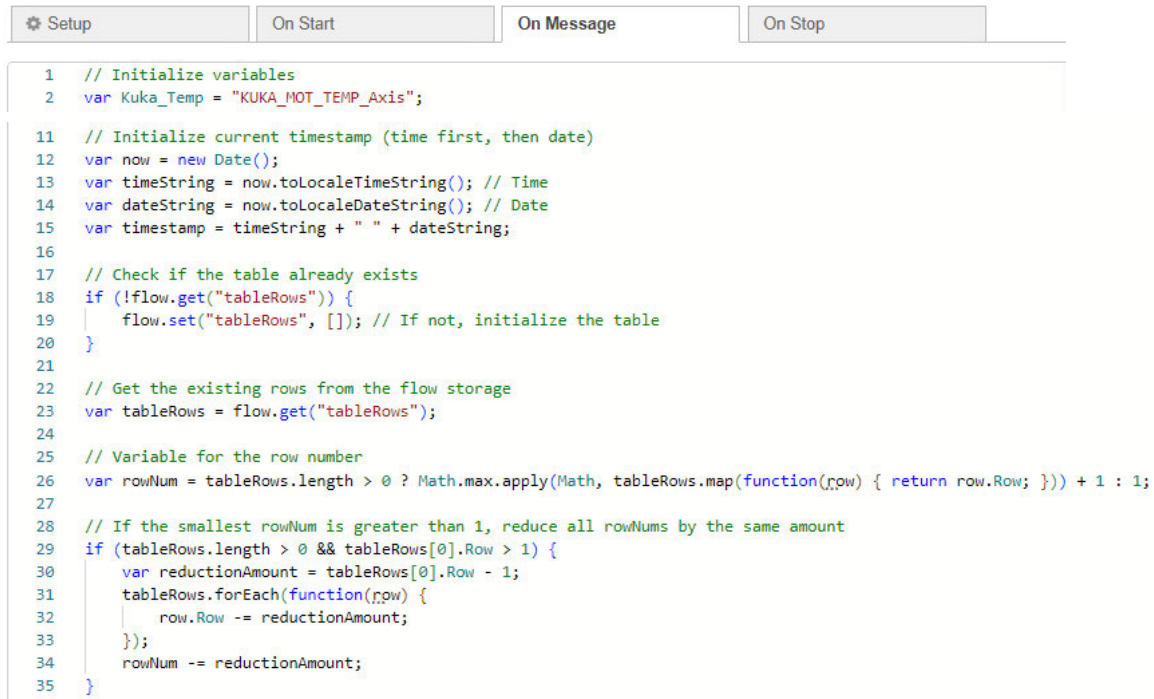


Figure 5.23: Alert-Table function nodes [67]

The function nodes, shown in Fig. 5.23, are used to create tables which will be utilized in a subsequent section to display the threshold-exceeding data within the user interface. These function nodes, exemplified by “Alert-Table (reported)”, will be explained in the following paragraph:

The function node, labeled “Alert-Table (reported)”, receives an array of threshold-exceeding parameters, recorded at the time of the excursion and after passing the delay node, from the connected join node and processes the data according to its code, shown in Fig. 5.24 and 5.25. The parameter for the KUKA axis motor temperature is used as an example for illustration purposes:



```

1 // Initialize variables
2 var Kuka_Temp = "KUKA_MOT_TEMP_Axis";

11 // Initialize current timestamp (time first, then date)
12 var now = new Date();
13 var timeString = now.toLocaleTimeString(); // Time
14 var dateString = now.toLocaleDateString(); // Date
15 var timestamp = timeString + " " + dateString;
16
17 // Check if the table already exists
18 if (!flow.get("tableRows")) {
19   flow.set("tableRows", []); // If not, initialize the table
20 }
21
22 // Get the existing rows from the flow storage
23 var tableRows = flow.get("tableRows");
24
25 // Variable for the row number
26 var rowNum = tableRows.length > 0 ? Math.max.apply(Math, tableRows.map(function(row) { return row.Row; }))) + 1 : 1;
27
28 // If the smallest rowNum is greater than 1, reduce all rowNums by the same amount
29 if (tableRows.length > 0 && tableRows[0].Row > 1) {
30   var reductionAmount = tableRows[0].Row - 1;
31   tableRows.forEach(function(row) {
32     row.Row -= reductionAmount;
33   });
34   rowNum -= reductionAmount;
35 }

```

Figure 5.24: Alert-table in central flow storage [67]

First, the variables, along with a timestamp, are initialized within the function node (see Fig. 5.24, lines 1 to 15). The table is stored in the central flow’s storage, a storage only accessible within this flow, and accessed through the reference in line 23. If the table does not already exist, it is created by the if-condition above (see Fig. 5.24, lines 18 to 19). Next, the existing row number is determined, and if the smallest row number is greater than one, it is adjusted accordingly. This ensures that the row numbers of the table always start from one and do not accumulate indefinitely over time (see Fig. 5.24, lines 28-34).

```

37 // Iterate over each message in the array
38 msg.payload.forEach(function (msgItem) {
39 // Initialize an empty table row for each entry
40 var row = { Row: rowNum, Parameter: "", Value: "", WarningLevel: "", Timestamp: "" };
41
42 // Add timestamp
43 row.Timestamp = timestamp;
44
45 // Kuka Temperature
46 if (msgItem.browseName.includes(Kuka_Temp)) {
47 row.Parameter = msgItem.browseName;
48 row.Value = msgItem.payload + " °C";
49 // Assignment of the "hard" threshold ("light" threshold is set via Excursion-Node)
50 row.WarningLevel = (msgItem.payload > 40 || msgItem.payload < 10) ? "hard warning" : "light warning";
51 }
52
53 // Add the row only if at least one value is assigned
54 if (row.Parameter !== "") {
55 tableRows.push(row);
56 rowNum++; // Increment rowNum only if a row is added
57 }
58 });
59
60 // Update the table rows in the flow storage
61 flow.set("tableRows", tableRows);
62
63 // Forward the updated row data for the table
64 if (tableRows.length > 0) {
65 return { payload: tableRows, topic: "Maintenance-Alert" };
66 } else {
67 return null; // No message if no conditions are met
68 }
69 }

```

Figure 5.25: Adding rows to the alert-table [67]

Analogous to the previously explained function node for maintenance alert messages, this function node is configured to iterate over each item in the array that includes measurement or parameter names and check if a specific variable contains a part of that name (see Fig. 5.25, line 46). At the beginning of the iteration loop, an empty table row, consisting of columns for the row number, parameter name, parameter value, warning level, and timestamp, is initialized. If the if-condition is met, the row is filled with the corresponding content (see Fig. 5.25, lines 46-51).

Subsequently, the *rowNumber* is incremented to reflect the addition of a new row, and the updated table is saved within the central flow's storage (see Fig. 5.25, line 103). Finally, the generated table is forwarded through the function node's output (see Fig. 5.23 and 5.25, line 107).

The function node's output is connected to a squared function node that configures the width of each table row. The subsequent table node then displays the table and its contents within the user interface. Additionally, the output from "Alert-Table (reported)" is linked to a button labeled "Refresh Alert-Table". When triggered, it activates a subsequent function node

labeled “flow.get,” which retrieves the current table from the central flow’s storage and forwards it to the table node for display. This action refreshes the contents of the table (see Fig. 5.23).

The second button, “Delete Alert-Table,” is connected to a squared function node that configures a subsequent notification node. This notification node displays a pop-up on the user interface, asking if the content of the alert table should be deleted. If confirmed, it triggers the connected node labeled “flow.set empty table,” which clears the table's content. The table is then refreshed by wiring it to the “flow.get” node (see Fig. 5.23).

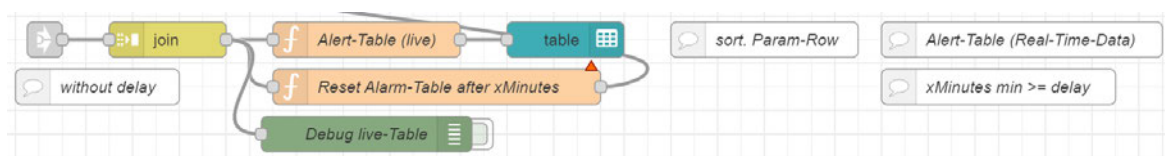


Figure 5.26: Alert table for real-time data

The second table, generated by the function node “Alert-Table (live)”, works analogously to “Alert-Table (reported)”, but uses a different variable for the table within the central flow’s storage. A key difference is that, unlike the previous table and all previous nodes, it is wired to the second join node within the central flow, receiving data from the condition monitoring flow without delay (see Fig. 5.26 and 5.15). This setup allows the table to display alert data in real-time, enabling maintenance personnel to see the current state and evolution of a threshold-exceeding parameter in addition to or in comparison with the reported parameter value and time at the moment of the excursion, as provided by the previously described “Alert-Table (reported)”.

As the join node transmits an array of real-time alert data every second, according to the central timestamp, and the “Alert-Table (reported)” needs to be automatically updated as well, an additional function node, located below “Alert-Table (live),” is wired to the join node (see Fig. 5.26). This function node is triggered every second and compares the timestamp of each table row within the “Alert-Table (reported)” to a predefined variable *xMinutes*, set to a delay of five minutes. If the difference between the current time and the timestamp of a row exceeds this delay, the row is either updated with the latest alert data from the real-time array or deleted if the issue was mitigated, through a wire to the button

“Refresh Alert-Table” (see Fig. 5.25). This ensures both tables reflect the most current information and maintain synchronicity between real-time and reported alerts.

5.2.3 Notification interruptions

In addition to the maintenance alerts and notifications within the central flow, as well as their respective nodes, an additional flow including two switches, labeled “Silencer” and “Silencer-Reset”, is implemented to introduce an additional functionality to the overall application within its user interface, and will be explained in the following paragraph:

When maintenance personnel are permanently stationed at the plant's workstation, monitoring the user interface and inspecting the alert tables to gain insights into the plant's condition, constant visual warnings in the form of pop-ups or email notifications, as well as acoustic signals, can be disruptive. These notifications might do more harm than good since the maintenance personnel are already at the central workstation and don't need further and constant notification, while addressing an issue. To address this, a silencer function is implemented to allow the notifications to be disabled via the user interface, without needing to access the Node-RED editor workspace.

However, if the notifications need to be temporarily disabled to focus on or mitigate a specific problem, it's essential to ensure that the notifications are not permanently turned off when the maintenance personnel are engaged in tasks and may not be able to manually reactivate them, as this poses a safety risk. To address this, alongside the silencer function, a silencer reset function is implemented using a second switch. This reset function ensures that notifications are automatically reactivated after a preset time, even if they were manually silenced. The flow and nodes responsible for both functionalities are depicted in Fig. 5.27.

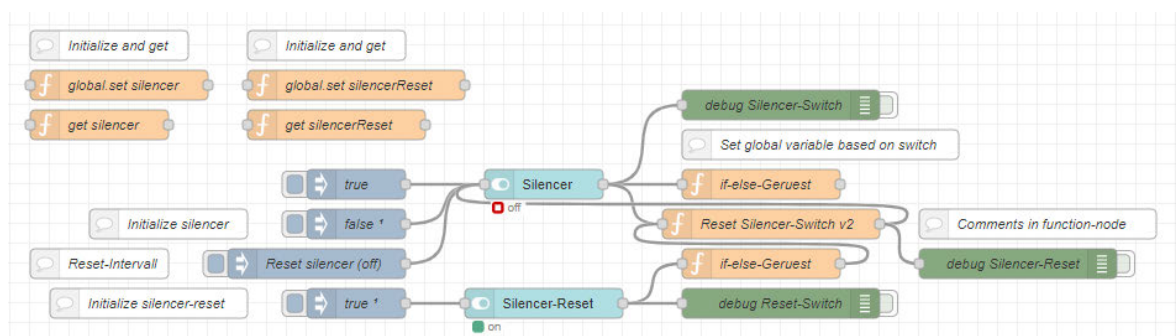


Figure 5.27: Nodes providing silencer functionality [67]

Figure 5.27 shows the initial state of both switches at start-up, represented by the light blue nodes at the center of the flow. The “Silencer” switch is off, while the “Silencer-Reset” switch is on (see Fig. 5.27). This configuration is achieved through the inject nodes, which are set to inject a corresponding boolean value at start-up. These nodes are wired to their respective inputs and labeled accordingly through comments. The other inject nodes in Fig. 5.27 are implemented solely for testing purposes.

Both functionalities are managed through global variables, which are accessible by all flows and initialized by the function nodes in the upper left corner of Fig. 5.27. When the silencer switch is turned on through the user interface, it triggers a subsequent function node that sets its global variable to true, reflecting its active state. Additionally, it triggers another function node below, labeled “Reset Silencer-Switch v2”, which includes a timer and starts if the reset switch and its global variable are also set to true. Once the timer is completed, the silencer switch is automatically reset to false, i.e. off, via the corresponding wiring (see Fig. 5.27).

If needed, the “Silencer-Reset” switch can also be turned off. In this case, it triggers a subsequent function node as well, that sets its global variable to off, reflecting its state. As a result, the “Silencer” switch will not be automatically reactivated. The “Reset Silencer-Switch v2”, which is responsible for reactivating the “Silencer” switch, contains additional functionalities, implemented to cover specific scenarios.

If the “Silencer-Reset” switch is toggled on and off multiple times, the timer will be reactivated each time the switch is turned on, only considering the most recent activation. Similarly, if the “Silencer-Reset” switch is turned on while the “Silencer” switch is turned off or reactivated before the timer expires, the timer will be reinitialized as well.

In order to disable the notifications, the global variable of the “Silencer” switch is read and checked by squared function nodes within the central flow, acting as a gate while receiving threshold-exceeding data. If the variable is true, the data is not forwarded to the subsequent nodes responsible for notifications, therefore disabling them (see Fig. 16).

All the nodes and flows previously shown can be individually customized and expanded according to the preferences of the maintenance personnel. The modular design allows thresholds to be tailored specifically to each monitored component. Additionally, as an open system, it supports the addition of new components, accommodating future changes and developments in the production plant or evolving maintenance needs.

The creation of the user interface and dashboards, whose functionalities are enabled by the previously described nodes, as well as their layout and arrangement, will be explained in the following section:

5.3 User interface and dashboards

Within this section, the creation of the user interface and dashboards within Node-RED is described, along with the integration of links and displays of and to the ERP-Lab, OpenCart web shop, Grafana, InfluxDB and Node-REDs editor workspace. Finally, the design and utilization of dashboards within Grafana is explained to provide more detailed and in-depth views of the production plant's data.

5.3.1 Node-RED dashboards

For the user interface and dashboards within Node-RED, Node-RED dashboard UI nodes are employed. In total, five dashboards are created, including the following:

- Alert-Dashboard
- Plant-Overview
- IMS (Energy-View)
- IMS (Counter-View)
- KUKA Overview

First, the necessary data flow to visualize the real-time data within the dashboards is established, as previously indicated by the dashboard UI nodes in Fig. 5.5. These dashboards include a real-time view of each IMS Station and the KUKA robot, represented by the last four dashboards listed. The final three dashboards provide specific visualizations for each individual parameter. The “Plant-Overview” dashboard offers a comprehensive view of all

production plant time series data, displayed in distinct graphs for related parameters. Finally, the “Alert-Dashboard“ serves as the primary user interface for displaying maintenance alerts and their associated information, including details of threshold-exceeding data and machine learning outputs. It also integrates additional functionalities and buttons to interact with the tables and notifications, as mentioned in Section 5.2.

IMS (Counter-View)

The visualization of the data within the IMS dashboards is realized with corresponding charts and text fields, previously displayed in Fig. 5.5, showing the flow of IMS Station 1. For each time series parameter, a chart node is used and configured as a line chart to display the parameter value in relation to time. For each counter, a text node is used to display the corresponding value. The chart and text nodes of each IMS Station are wired to the switches, enabling the data flow and assigning the parameters as shown in Fig. 5.5, and then arranged and displayed within the dashboard layout editor to form the dashboard “IMS (Counter-View)”, as shown in Fig. 5.28:

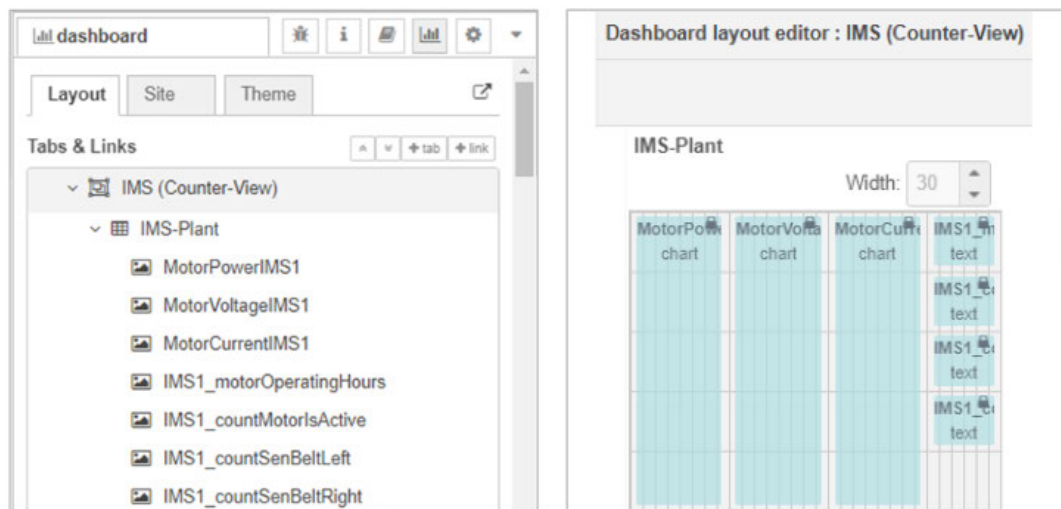


Figure 5.28: Dashboard layout IMS (Counter-View) [67]

This process is repeated for every IMS Station and their respective data, wiring their nodes and categorizing and arranging their charts and displaying counter values through text nodes accordingly. Figure 5.28 serves as an example for the setup of IMS Station 1.

IMS (Energy-View)

For the dashboard of the “IMS (Energy-View)” a new and separate data flow is created, as shown in Fig. 5.29:

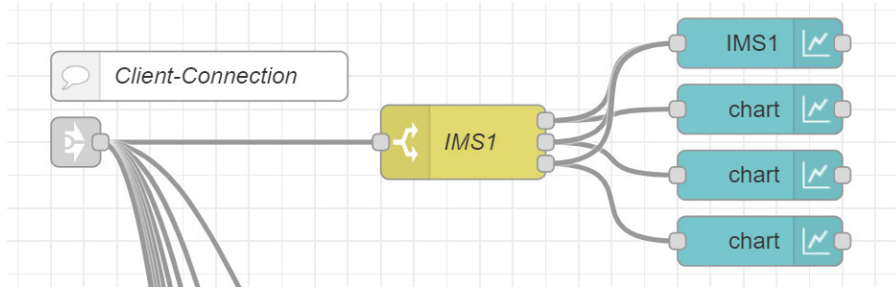


Figure 5.29: Charts of IMS Station 1 within the Energy-View [67]

A separate switch node for each IMS Station is created and wired to a link node, which is connected to the central OPC UA client. The switch node, as shown in the example of the switch for IMS Station 1 (see Fig. 5.29), routes each parameter, such as current, voltage, and power, to a distinct chart node for the “IMS (Energy-View)” dashboard. The chart node above those three charts, labeled “IMS1” and wired to all three of the switch’s outputs, visualizes all three parameters within a single graph and is used for the “Plant-Overview” dashboard. All chart nodes are hereby configured as line charts and arranged as shown in Fig. 5.30, with three charts per IMS Station arranged vertically and the stations positioned side by side:

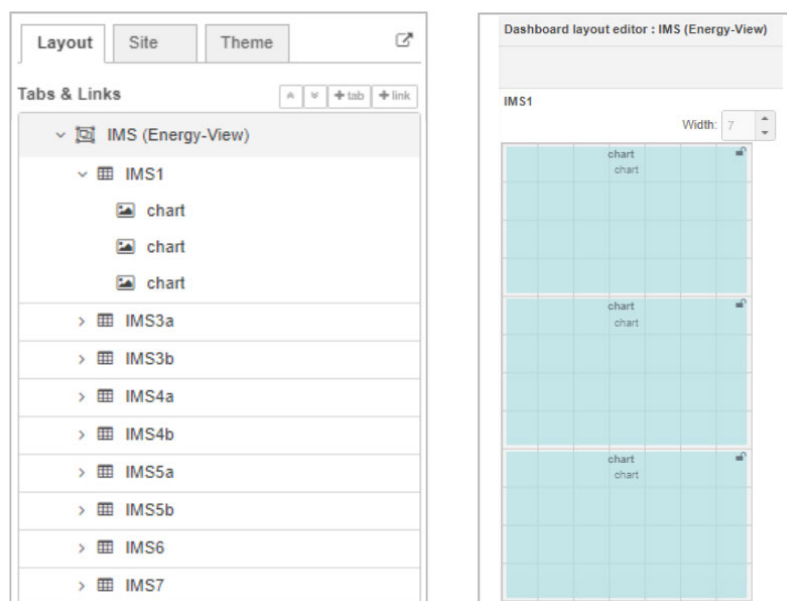


Figure 5.30: Dashboard layout IMS (Energy-View) [67]

KUKA Overview

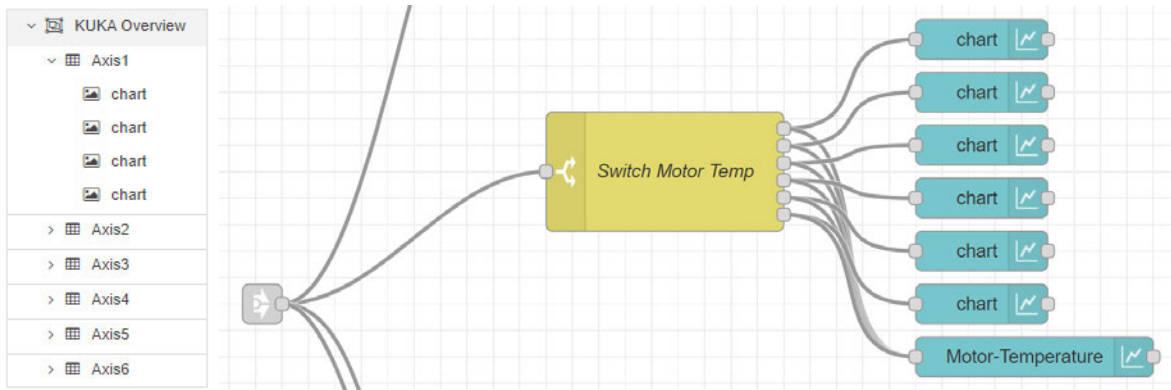


Figure 5.31: Charts of the axis motor temperature within the KUKA Overview [67]

This process is repeated for each axis and axis parameter of the KUKA robot, as shown in Fig. 5.31. The charts are configured and arranged in a manner similar to the IMS Stations, with four charts per axis arranged vertically and the six axes arranged horizontally for the “KUKA Overview” dashboard.

Plant-Overview

The "Plant-Overview" dashboard is created in a similar manner to the previous two dashboards, using the labeled charts wired to every output of the switches simultaneously, as shown in Fig. 5.29 and 5.31. The complete data flows for all dashboards, as well as the arrangement of charts for the "Plant-Overview," can be found in Appendix A.3.

Alert-Dashboard

Using the template node, the table nodes and buttons (see Fig. 5.15), as well as the switches (see Fig. 5.27), previously configured and described in Sections 5.2.2 and 5.2.3, the “Alert-Dashboard” is created. As shown in Fig. 5.32, it provides a view for the latest alert message through the template node, contains the reported and real-time alert tables, along with the “Refresh Alert-Table” and “Delete Alert-Table” buttons as well as the “Silencer” and “Silencer-Reset” switches.

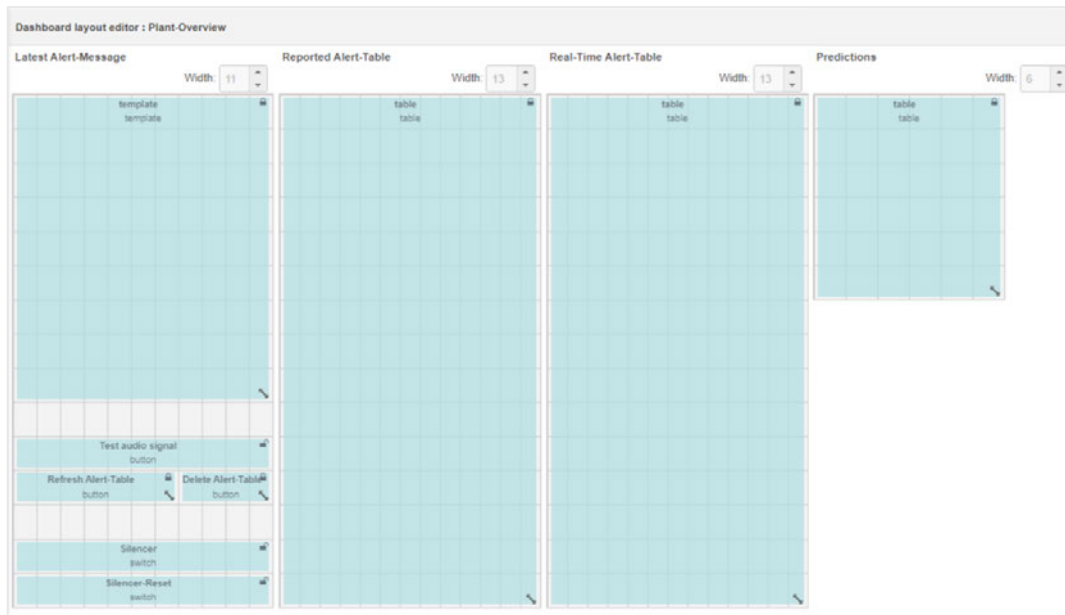


Figure 5.32: Layout of the Alert-Dashboard [67]

In addition, it consists of a placeholder table for the machine learning outputs, labeled “Predictions” and implemented in Section 5.4, as well as one additional button, which will be explained briefly in the following paragraph:

The button, labeled “Test audio signal,” located below the template node and above the other buttons and switches (see Fig. 5.32), is wired to an audio out node, as shown in Fig. 5.33.

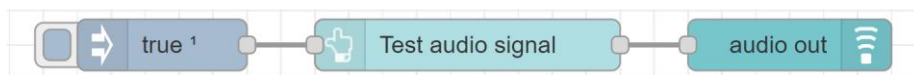


Figure 5.33: Test audio signal button [67]

This button provides the string "audio online" as an input to test whether the device accessing the user interface is capable of producing an audio signal, and therefore able to provide an acoustic alert for maintenance personnel in case of a changing condition within the production plant. The nodes are located within the central flow and wired to an inject node, which triggers the button automatically once at system startup, indicating that the audio and the system are online. Afterwards, the button can be pressed manually within the user interface to verify the audio functionality at any time.

Accessibility of dashboards and sites

All dashboards are configured to be accessible by selecting their name within a sidebar menu. In addition, links to all tools and tool interfaces, as well as the ERP-Lab and OpenCart webshop, are provided within the sidebar menu to enable central access from Node-RED's dashboards as the primary user interface, as shown in Fig. 5.34.

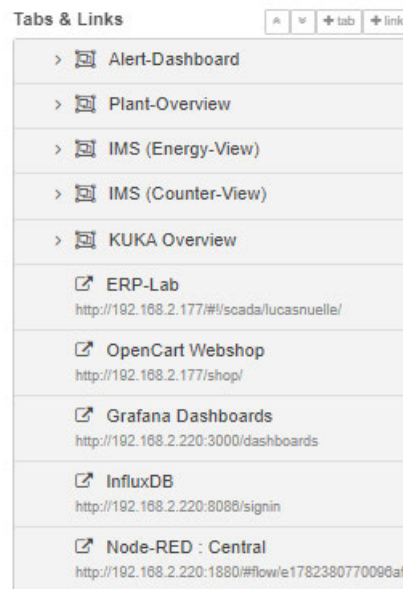


Figure 5.34: Tabs and links of the user interface [67]

To ensure that all notifications are recognized, they are configured to display within all tabs of the user interface. When accessing the links, a new tab opens with the selected view. Additionally, if desired, the display of the pages can be configured to occur within the frame of the Node-RED dashboards.

5.3.2 Grafana dashboards

As the dashboards of the IMS Stations and KUKA robot are only able to display the real-time data and provide no additional functionalities to inspect the data in more detail or interact with it through the user interface, a series of dashboards is created within Grafana (see Fig. 5.35).

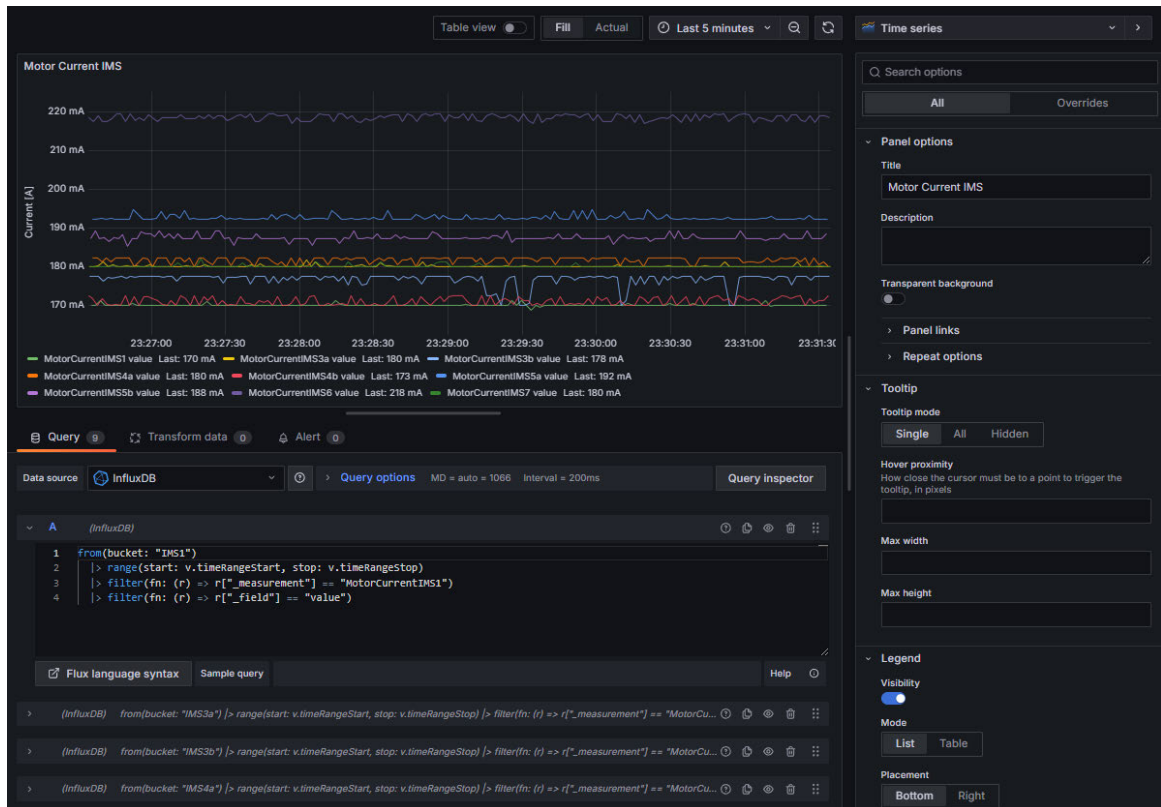


Figure 5.35: Creation of a time series graph with flux queries in Grafana [67]

The creation of a time series graph in Grafana is achieved by using Flux queries from InfluxDB (see Fig. 5.8), as illustrated in Fig. 5.35. In this instance, seven queries are used to visualize the motor current for all IMS Stations within a single time series graph. In this way, multiple time series graphs and gauges are created and then organized into separate dashboards to provide distinct views and representations of the data generated by each individual IMS Station or the KUKA robot, as well as combined overviews of energy or power consumption.

A section or part of these overviews and dashboards is presented in Fig. 5.36 and 5.37 below:



Figure 5.36: The current of each KUKA axis as an excerpt of its dashboard [67]

Figure 5.36 shows a section of the KUKA robot's overview within Grafana. It visualizes the current of each of its axes within a time series graph as well as a bar gauge, enabling to see both the progression and the current value simultaneously. Besides the current of each axis, the dashboard also includes corresponding views for the temperature, velocity, and torque of each axis in the same manner.

The dashboard for IMS Station 4a is shown in Fig. 5.37 as an example for all stations. It features displays for energy consumption and all relevant counters, as well as time series graphs for the current, voltage, and power of its conveyor belt motor.

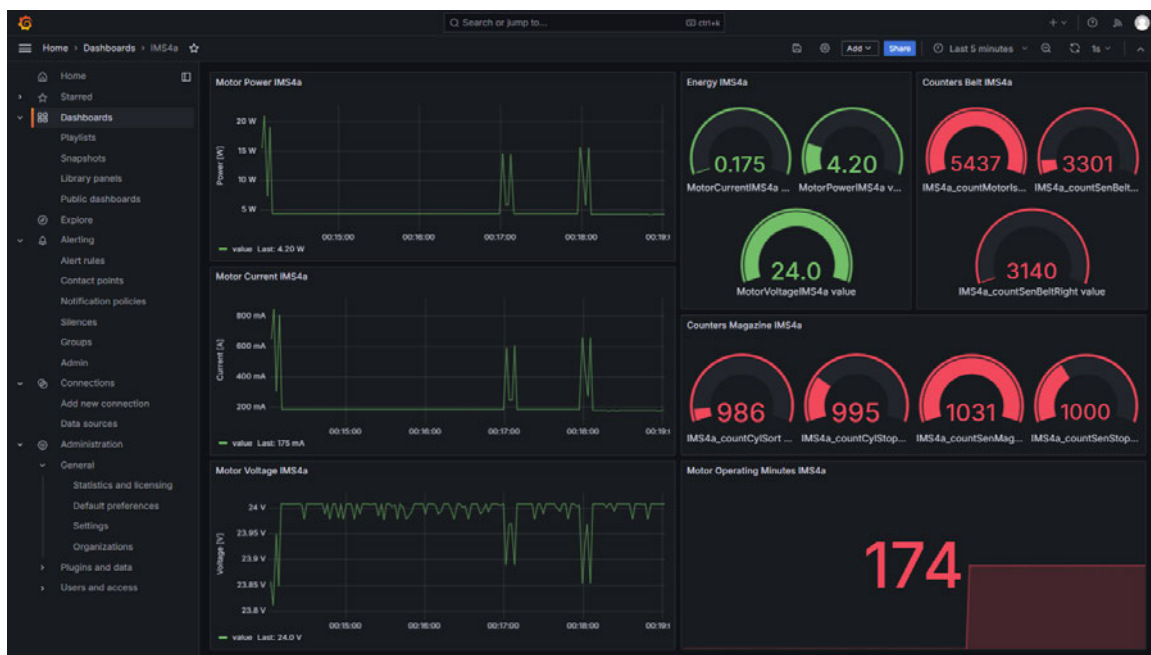


Figure 5.37: Dashboard of IMS Station 4a [67]

The refresh rate is set to one second to display the data in real-time, with the displayed time range set to the last five minutes. If a wider range or specific period of historical data needs to be inspected, this can be adjusted using a dropdown menu above the gauge for the counters of its belt (see Fig. 5.37).

In addition to a specific view for each station, an overview of all IMS Stations is created, similar to Figures 5.36 and 5.37, to provide a comprehensive view of the conveyor belt consumption across all stations simultaneously. A list of all dashboards is provided in Fig. 5.38 below:

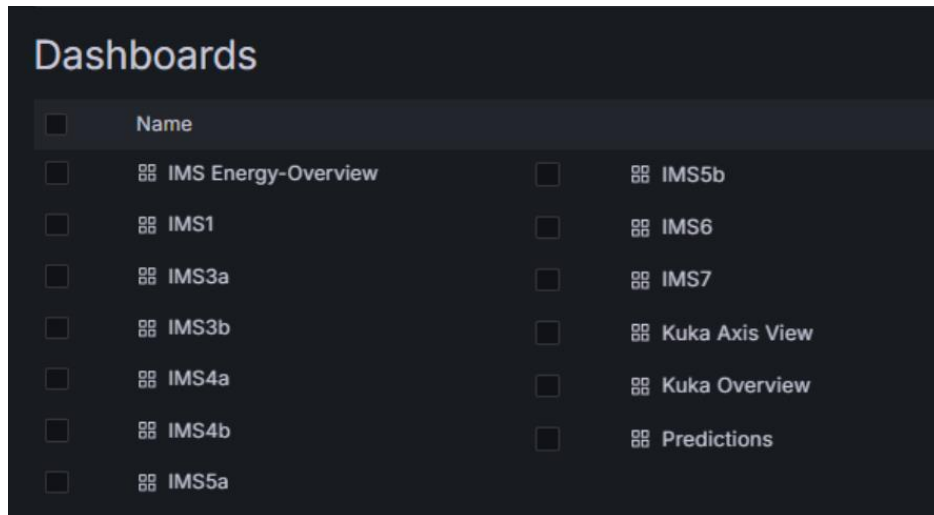


Figure 5.38: List of all Grafana dashboards [67]

This list also includes a placeholder for the machine learning outputs, similar to the "Alert-Dashboard" (see Fig. 5.32 and 5.38). The generation of these outputs, including the selection and training of machine learning algorithms within MATLAB, as well as the subsequent integration of a final model, will be described in the following sections.

5.4 Data analysis through Machine Learning

This section describes the process of acquiring and labeling representative data from the production plant to form training datasets for the machine learning algorithms, followed by the feature selection and extraction, as well as the training and evaluation of these algorithms. Finally, a data flow between the final machine learning models and the database will be implemented to enable automatic analysis during the production plant's runtime and to provide outputs for the user interface.

The focus of this section is to provide a general overview and workflow of the steps involved, from data acquisition to the selection and training of machine learning algorithms using MATLAB's advanced capabilities and toolboxes, taking the fundamentals described in Section 2.6 into account. This overview avoids detailing each process with data science and code specifics, as the toolboxes within MATLAB are designed to facilitate and streamline these tasks. The goal is to identify and implement a suitable classifier for the previously presented application, using a practical approach.

5.4.1 Acquiring representative data

The data collected from the production plant has to provide relevant information about its current state and condition. Therefore, the time series data provided by the conveyor belt motors and the individual axes of the KUKA robot is acquired during regular production sequences and at regular intervals. Since power consumption is the product of current and voltage, this parameter is exclusively used to analyze the motors' performance and identify potential issues. In addition to the data representing the plant's normal operation, possible distinct failure modes and classes are defined to be predicted or classified by the algorithms during future production.

As data representing failures, faults, and breakdowns are not available, likely failure modes on the conveyor belts are defined, manually simulated, and collected. Since it is not feasible to manually generate data representative of faults for the KUKA robot and anticipate how these would manifest at this specific plant and production environment, the regular data of the KUKA robot is used to train an anomaly detection algorithm.

The defined classes for the conveyor belts failure states or classes are listed below:

- Slowed
- Blocked
- Belt loose: 1
- Belt loose: 2
- Signal on/off

For the first class, the production plant operates normally, and folded paper sheets are positioned between the belts and their rotating components to slow them down. This represents worn-out belts that are no longer elastic or simulates objects that may get caught in the conveyor belt during production. This is performed on all circularly arranged conveyor belts simultaneously.

The second class, "blocked," is recorded by manually blocking a belt completely by holding and locking its rotating components in one position. This is performed on a few random occasions during production and on a single conveyor belt, as this is a time-consuming manual task. Meanwhile, the other conveyor belts continue to operate normally.

The classes "belt loose 1" and "belt loose 2" are recorded by forcing their respective signals for moving to the right in the TIA Portal (signals, see Table 3.1). The belts are operated constantly while loosening one belt, and then both belts completely. This is intended to represent belts getting completely loose or tearing during production. This is performed outside the normal production process on all circularly arranged conveyor belts. For the last listed class, a custom function block within the TIA Portal is used to repeatedly switch the individual motors on and off at intervals ranging from 0.2 to 1 second, differing by 0.5 seconds each time, over an extended period. This is also performed outside the normal production process on all circularly arranged conveyor belts to simulate signal failures and interruptions between the PLCs and the conveyor belts during production.

The data representing all classes is collected and stored using the previously described data flow (see Section 5.1) and saved in InfluxDB, alongside data representing the KUKA robot's normal operational behavior, which is simultaneously acquired during regular production. In this context, regular production refers to operations involving one, two, or three carriers within the production system, aiming to cover all variations and effects from intermittently occupied stations and waiting carriers. The data is then exported as CSV files from InfluxDB and stored in a folder accessible by MATLAB. For the upcoming tasks and sections, MATLAB R2023b Update 6 is used, including the "Predictive Maintenance Toolbox", the "Statistics and Machine Learning Toolbox" and the "Deep Learning Toolbox".

The previously described data collection and processing methods result in the application of both supervised and unsupervised learning approaches. Specifically, supervised learning methods are used to analyze the conveyor belts, while unsupervised learning is applied to the KUKA robot. Both approaches will be explained in the following sections, while the focus lies on the steps and tasks related to the conveyor belts and supervised learning. Afterwards, a brief perspective on anomaly detection for the KUKA robot will be provided to offer a general understanding of the practical approach.

5.4.2 Data analysis of conveyor belts

Organizing, labeling and preprocessing the data

First, the power consumption during the normal operational behavior of the individual conveyor belt motors is inspected, as shown in Fig. 5.39:

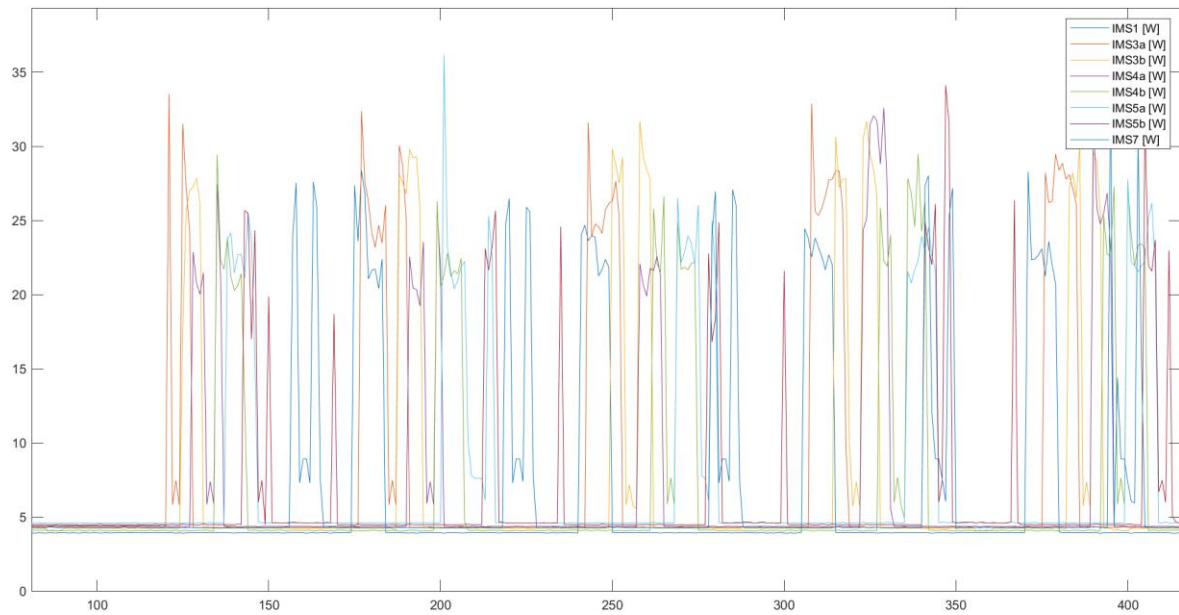


Figure 5.39: Normal power consumption of the conveyor belt motors [67]

The diagram shows the power consumption in watts over the number of data points, representative of seconds, during the normal operational behavior of the circularly arranged conveyor belts. It shows in a simplified manner, that the power consumption of each motor has a baseline or ground consumption and briefly peaks when turned on, before settling at a certain level during transport and returning to the baseline. Upon closer examination, however, it becomes evident that each motor has a different ground level, as shown in Fig. 5.40:

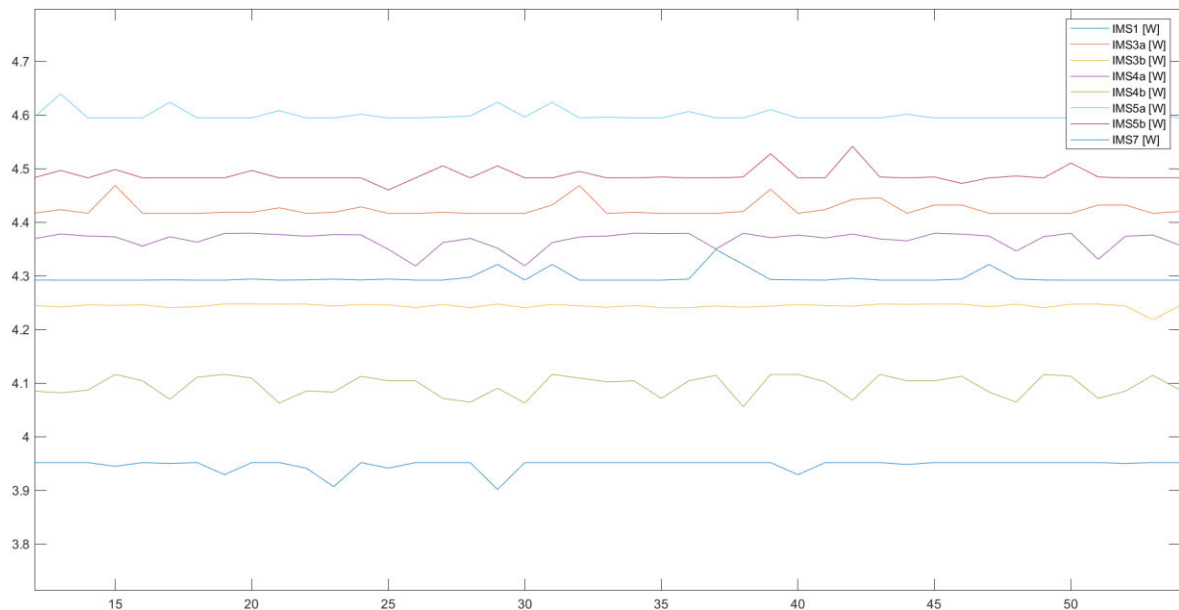


Figure 5.40: Ground level of circular arranged conveyor belt motors [67]

To avoid the need to train multiple customized machine learning algorithms for each conveyor belt, the data is preprocessed to standardize it across all motors. Since power consumption during production is primarily at the ground level and the motors are sequentially turned on and off, the data is centered around zero using the median. Additionally, missing values are filled using linear interpolation of adjacent values (see Fig. 5.41).

```
% Apply normalization
dateiname_center = normalize(daten, "center", "median", "DataVariables", "Value");

% Fill Missing Values --> no missing values
dateiname_center = fillmissing(dateiname_center, 'linear');
```

Figure 5.41: Preprocessing [67]

To ensure that no information about individual peaks, including error classes (see Fig. 5.43), is unintentionally lost, and because the data is all on the same scale, no further normalization beyond the already applied method is used. The data after preprocessing, is displayed in Fig. 5.42, centered around zero.

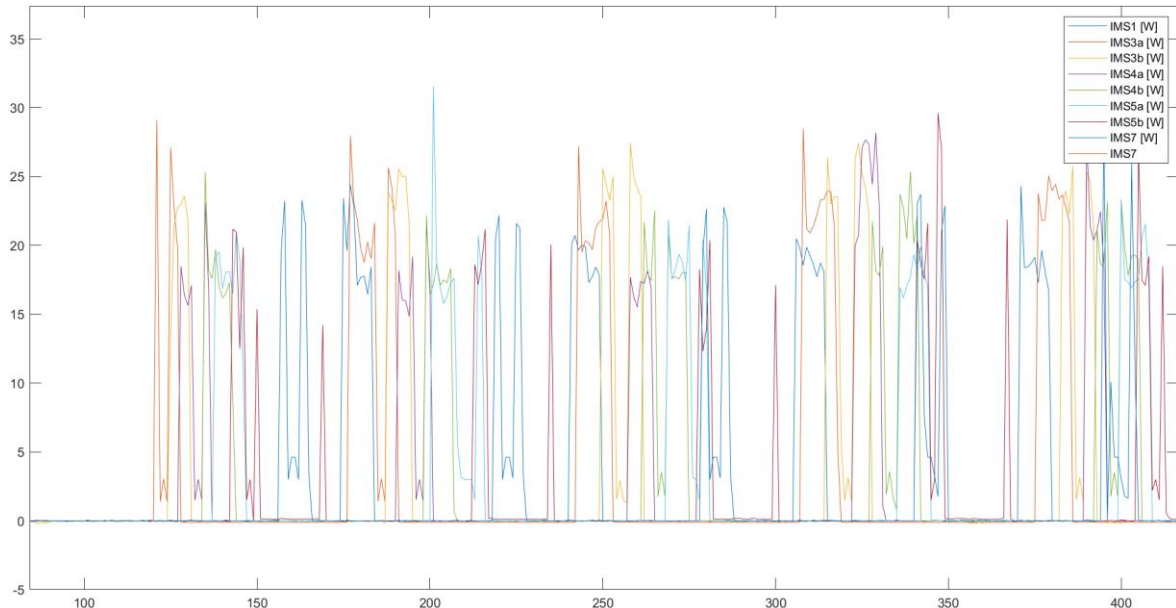


Figure 5.42: Data preprocessed and centered around zero [67]

During the download of the CSV files, the files were named according to the classes, which are used to automatically label them using a custom script. However, since each class contains a ground level, differentiation must be made accordingly to have distinct

representations of each class. Therefore, the previously listed classes are expanded to include the class "ground." A data point is labeled as "ground" if it, along with its preceding nine data points, has a value below 0.2 W. The different classes are exemplarily shown in Fig. 5.43, with "ground" representing the baseline around zero and "normal" in dark blue:

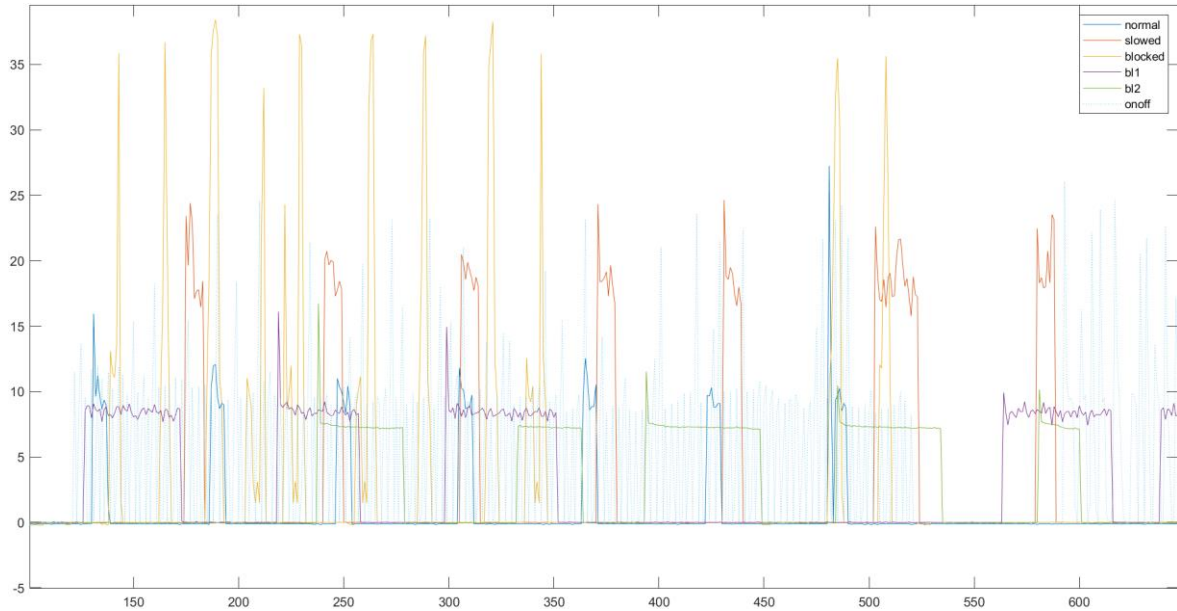


Figure 5.43: Visualization of the different classes [67]

Automated feature extraction

As shown in Fig. 5.43, the individual classes present distinct characteristics that allow to differentiate between them, based on features, such as the peak or maximum time series value, mean, rms (root mean square), and other possible differentiating features. To identify and extract all relevant features, the “Diagnostic Feature Designer” app in MATLAB is used, which is a part of the “Predictive Maintenance Toolbox”. This app automizes the process of feature extraction based on statistical and signal-based characteristics and provides a ranking of features based on their ability to differentiate between classes.

For this purpose, the data is organized into sequences of individual timetables based on their labels. Each sequence ranges from the activation of a motor to its stop and is therefore separated from sequences of the same, or other classes, by a sequence of "ground" data. Each timetable, representing a distinct sequence for its class, is then compiled into a large .mat file, which consists of two columns containing all timetables and their labels.

Through this approach, time series features can be extracted from the data, while the resulting feature set is not bound by the time series data or temporal sequences themselves, allowing the training of a wide range of machine learning models, without being limited to algorithms capable of handling time series data.

Afterwards, the .mat file is uploaded into the “Diagnostic Feature Designer,” where automated feature extraction is performed. Based on the ranking from a Kruskal-Wallis test, which assesses the features according to their differentiability, as well as scatter plots (see Fig. 5.44), the following twelve features are selected:

- Signal statistics:
 - RMS
 - Peak value
 - Mean
- Time series features:
 - Q3 (third quartile)
 - Maximum
 - Median
- Time series model:
 - AIC (Akaike Information Criterion)
 - RMS
 - Variance
 - Mean
 - Median
 - MAE (Mean Absolute Error)
 - MSE (Mean Squared Error)

As MATLAB does not specify how each feature is computed, scatter plots are used as an additional method to verify their differentiability by comparing them within a feature space. In the example shown in Fig. 5.44, the feature space is formed by the signal statistics peak value (y-axis) and mean (x-axis), with the data points representing different classes accordingly.

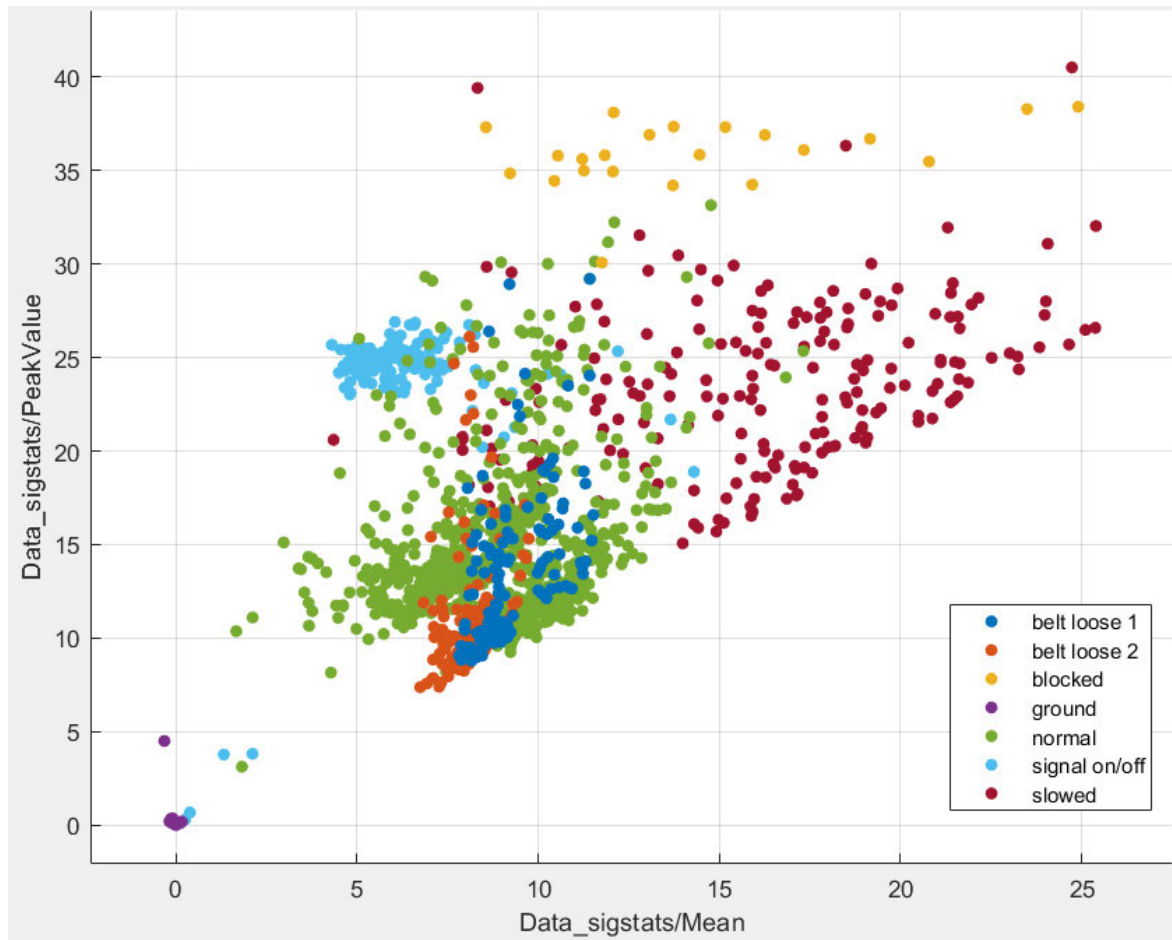


Figure 5.44: Scatter plot of the signal statistics peak value ... mean

The scatter plot provides a visualization of the class distribution, indicating that the selected features can effectively separate the different classes, though with a few outliers. However, it also reveals that the samples per class are unevenly distributed. This uneven distribution results from the manual aggregation of data representative of faults, as well as the consolidation of the “ground” levels from all other classes into a its own category. The features of the “ground” class hereby show the highest distinguishability, with points clustered closely around zero. The “blocked” class has the fewest samples, as illustrated in Fig. 5.44, due to the described method of data acquisition.

After the selection and extraction of features, the “Diagnostic Feature Designer” allows for exporting these features to the MATLAB workspace, as well as generating MATLAB code, that can be used for automation or integration into custom workflows. Additionally, it enables direct export of the features into the “Classification Learner” app, part of the “Statistics and Machine Learning Toolbox,” which facilitates the training and evaluation of machine

learning models. In the following paragraph, the entire feature set will be used to train classifiers to estimate their performance and identify the most suitable machine learning algorithm. The feature set will then be refined and augmented to ensure an even distribution across all classes and achieve a representative accuracy.

Training classifiers with the entire sample size

After the automated feature extraction, the resulting feature table is checked for and cleared of any possible missing values before being loaded into the “Classification Learner” app. Figure 5.45 shows the window for creating a new session, where the selected features, a 10-fold cross-validation, and a 10% holdout for the test dataset are specified:

Data set

Data Set Variable: FeatureTable_clean (3084x13 table)

Response: ☒ From data set variable, ☐ From workspace. Label (cell, 7 unique)

	Name	Type	Range
<input type="checkbox"/>	Label	cell	7 unique
<input checked="" type="checkbox"/>	Data_sigstats/Mean	double	-0.320723 .. 40.5278
<input checked="" type="checkbox"/>	Data_sigstats/PeakValue	double	0.000102758 .. 41.4969
<input checked="" type="checkbox"/>	Data_sigstats/RMS	double	2.85001e-05 .. 40.5394
<input checked="" type="checkbox"/>	Data_tsfeat/Maximum	double	-0.17489 .. 41.4969
<input checked="" type="checkbox"/>	Data_tsfeat/Median	double	-0.229541 .. 40.5278

Buttons: Add All, Remove All, Refresh

[How to prepare data](#)

Validation

Validation Scheme: Cross-Validation

Protects against overfitting. For data not set aside for testing, the app partitions the data into folds and estimates the accuracy on each fold.

Cross-validation folds: 10

[Read about validation](#)

Test

☒ Set aside a test data set

Percent set aside: 10

Use a test set to evaluate model performance after tuning and training models. To import a separate test set instead of partitioning the current data set, use the Test Data button after starting an app session.

[Read about test data](#)

Buttons: Start Session, Cancel

Figure 5.45: Creating a session in the "Classification Learner" app [67]

The "Classification Learner" is then used to train a series of machine learning algorithms from the app’s comprehensive set, which includes SVM, decision trees, discriminant analysis, logistic regression, naive Bayes, nearest neighbor classifiers, and simple neural networks. These algorithms are ranked based on their accuracy, with the following three models providing the highest validation accuracies:









Models		
Sort by Accuracy (Va...  		
  		
 2.11 SVM	Accuracy (Validation): 98.4%	
Last change: Quadratic SVM 12/12 features		
 2.12 SVM	Accuracy (Validation): 98.3%	
Last change: Cubic SVM 12/12 features		
 2.29 Neural Network	Accuracy (Validation): 97.7%	
Last change: Wide Neural Network 12/12 features		

Figure 5.46: Excerpt of the ranking based on validation accuracy [67]

A quadratic SVM with a polynomial kernel achieves the highest validation accuracy of 98.4%, closely followed by a cubic SVM with 98.3% and a neural network with a comparable validation accuracy of 97.7%. However, an examination of the SVMs confusion matrix and the sample size of each class reveals significant class imbalance (see Fig. 5.47).

		Model 2.11 (Quadratic SVM)								
True Class	belt_loose_1	belt_loose_2	blocked	ground	normal	signal_onoff	slowed		TPR	FNR
	124	1			4				96.1%	3.9%
	1	121			1				98.4%	1.6%
			18		3		1		81.8%	18.2%
				1417					100.0%	
					714	4	5		98.8%	1.2%
						7	153	1	95.0%	5.0%
	1		3		12		185		92.0%	8.0%
		belt_loose_1	belt_loose_2	blocked	ground	normal	signal_onoff	slowed		
		Predicted Class								

Figure 5.47: Validation confusion matrix of quadratic SVM with all features [67]

The confusion matrix, which visualizes the true classes against the predicted classes, shows the true classified instances, also referred to as true positives, along its main diagonal. The off-diagonal elements represent misclassifications, including false positives (instances

incorrectly classified as a particular class) and false negatives (instances of a class incorrectly classified as another).

As visualized in Fig. 5.47, the "blocked" class, with its few samples, is underrepresented and less accurately identified due to a higher percentage of misclassifications. In contrast, the "ground" class, which has the largest sample size, is classified with 100% accuracy. This means that the overall accuracy might be misleading, as the high accuracy of the "ground", class, along with its high sample size, inflates the model's overall performance.

While the overall accuracy for the other classes, depicted through the True Positive Rate (TPR) on the right side of the confusion matrix, is above 92%, indicating strong performance and effective prior data preprocessing and feature selection, efforts are being made to improve the classification accuracy for the "blocked" class. This involves increasing its sample size through augmentation and reducing the sample sizes of other classes to achieve a more balanced dataset.






Additionally, misclassified instances are predominantly misclassified as "normal". In a production environment, this can be problematic as failures might go undetected. By reducing the sample size and balancing the dataset, this issue could potentially be mitigated, helping to ensure that failures are more accurately identified.


Training classifiers with a reduced sample size

The sample size of all classes is reduced to 100 randomly chosen samples per class. Additionally, the sample size of the "blocked" class is increased through feature augmentation, where samples are duplicated and multiplied, starting with a factor of 1.01 and increasing by increments of 0.1 until the desired size is achieved. This approach ensures that all classes are represented by 90 sequences during training with cross-validation, while minimizing excessive augmentation of the "blocked" class. The remaining 10% of samples are reserved for testing, consisting only of original features to ensure that they are correctly identified and that the training results are accurate.

Due to the overall smaller sample size, a 5-fold cross-validation is applied. The results of training and ranking all classifiers are displayed, with the top three ranked models shown in Fig. 5.48.

Models


Sort by Accuracy (Va...     



2.11 SVM

Accuracy (Validation): 97.9%


Last change: Quadratic SVM12/12 features



2.28 Neural Network

Accuracy (Validation): 97.5%

Last change: Medium Neural Network12/12 features



2.12 SVM

Accuracy (Validation): 97.1%

Last change: Cubic SVM12/12 features

Figure 5.48: Ranked classifiers trained with smaller sample set [67]

Again, the quadratic SVM shows the highest accuracy among all trained classifiers, achieving an overall validation accuracy of 97.9%, which is only 0.5% less accurate than the model trained on all feature samples. The second-ranked model is a medium neural network, referring to its size, with a validation accuracy of 97.5%. The cubic SVM, previously ranked second, holds the third position with a validation accuracy of 97.1%. The confusion matrix of the trained quadratic SVM is presented below:

Model 2.11 (Quadratic SVM)							
True Class	belt_loose_1	87			3		
	belt_loose_2	3	87				
	blocked			90			
	ground				90		
	normal	2				88	
	signal_onoff	1			1	87	1
	slowed			2			88
		belt_loose_1	belt_loose_2	blocked	ground	normal	signal_onoff
Predicted Class							

96.7%	3.3%
96.7%	3.3%
100.0%	
100.0%	
97.8%	2.2%
96.7%	3.3%
97.8%	2.2%
TPR	FNR

Figure 5.49: Validation confusion matrix of trained quadratic SVM [67]

The confusion matrix shows that the equally distributed samples across all classes are classified with a minimum classification accuracy of 96.7%. The accuracy of the "normal" and

"belt loose 2" classes slightly decreased, while the accuracy of all other classes increased to an even level. The identification accuracy of the "blocked" class improved significantly from approximately 81% to 100% due to the increased number of learning examples. Additionally, the misclassification of classes as "normal" was reduced (see Fig. 5.48).

Afterwards, an attempt was made to optimize the SVM based on its hyperparameters. However, the initial model trained by the "Classification Learner" proved to have the best accuracy. The model is then tested with the previously partitioned test set to verify if its performance is reproducible and applicable to the original samples. The hyperparameters of the model and the test results are presented in Fig. 5.50 below:

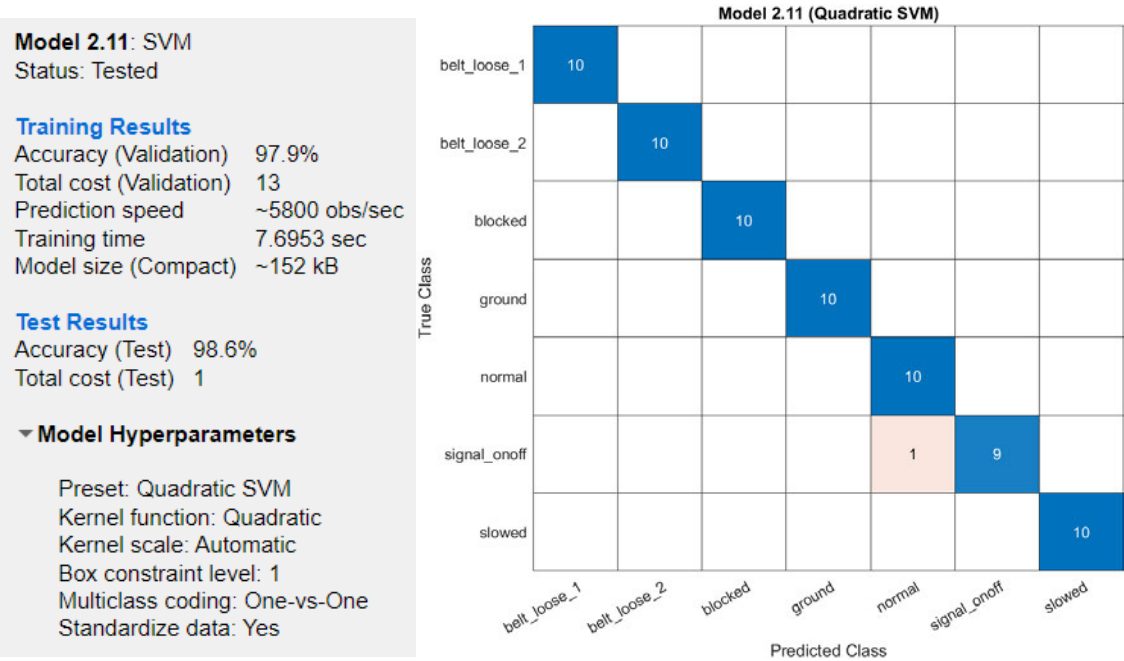


Figure 5.50: Quadratic SVM training and test results [67]

During testing, the classifier, performing one-vs-one classification with a box constraint level of one (representing the parameter for the soft margin), achieves an accuracy of 98.6% by misclassifying only one instance (see Fig. 5.50). The test sample size is relatively small, which results in a slightly higher accuracy. However, due to the prior application of 5-fold cross-validation, this presents a robust final evaluation of the classifier's performance. The model is capable of processing approximately 5800 observations per second, as shown in Fig. 5.50. This high processing speed, combined with its accuracy, makes it suitable for real-time application within the condition-based and predictive maintenance solution.

5.4.3 Anomaly detection for the KUKA robot

This section will briefly cover the training of three machine learning algorithms to identify anomalies during the operation of the KUKA robot, enabling the detection of unknown issues during production. The anomaly detection is performed on the KUKA robot as a whole and is trained using the current, torque, and velocity data of all axes. The motor temperature is not used for training the anomaly detector, as it consists of a single, mostly steady temperature value and is already monitored through the condition monitoring.

Analogous to the previous section, the collected data of the KUKA robot axis is uploaded into MATLAB and labeled accordingly, with the difference that, in this case, all data is labeled as "normal." In addition to the "normal" data, data representative of abnormal behavior is collected by performing the regular sequences of the KUKA robot during production at double speed. This data is labeled as "abnormal" accordingly. Since the KUKA robot mostly remains in its idle position and only moves during the transport of a workpiece, both classes also consist of a ground level for each robot axis that needs to be considered "normal" by the future model during production. Therefore, the data is separated into sequences consisting of "ground," "normal," and "abnormal" data and stored as timetables, similar to the data of the conveyor belts. The final table of data consists of three columns: one that contains the timetable, one named "segments" that differentiates between the three types of sequences, and a column named "label" that only consists of two different labels, with "ground" labeled as "normal." The data mainly differs in its intensity, as shown by the current and torque of axis 1 in Fig. 5.51:



Figure 5.51: normal data (left) and anormal data (right)

The data is then uploaded into the “Diagnostic Feature Designer,” and the top fifteen ranked features are selected. The features are subsequently exported to the MATLAB Workspace and divided into a training feature set, consisting of 201 normal samples, and a testing set, comprising 22 normal and 17 abnormal samples. This data is used to train a binary SVM, an Isolation Forest, and an Autoencoder for anomaly detection, as described in the following paragraphs. As the data only consists of normal data, no cross-validation is performed.

Binary SVM

```
% Training of binary SVM (OutlierFraction 0 --> all data are normal)
mdlSVM = fitcsvm(featureNormal, 'Type', 'Standardize', true, 'OutlierFraction', 0);

% test SVM
[~,scoreSVM] = predict(mdlSVM,featureTestNoLabels);
isanomalySVM = scoreSVM<0;
predSVM = categorical(isanomalySVM, [1, 0], ["Anomaly", "Normal"]);
```

Figure 5.52: Creation of the binary SVM [67]

The binary SVM is trained according to the settings displayed in Fig. 5.52. The features are standardized, and the *OutlierFraction*, which refers to the percentage of possible anomalies within the data, is set to zero. Afterwards, the SVM is tested, showing the following results:

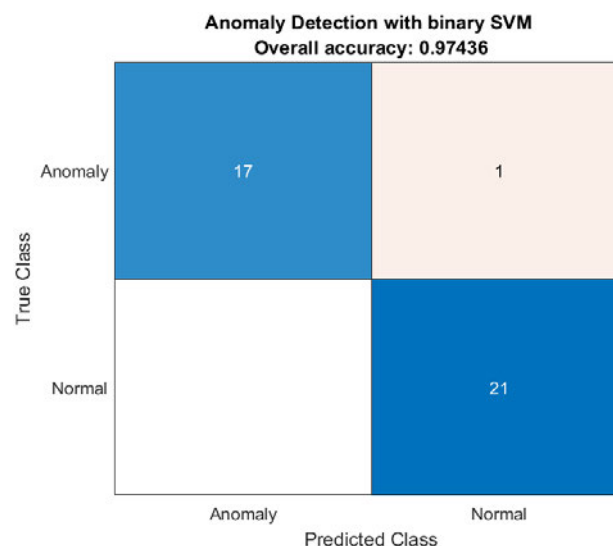


Figure 5.53: Confusion matrix SVM [67]

The SVM is able to sufficiently differentiate between normal and abnormal data, as shown in Fig. 5.53. Only one sample is misclassified as "normal," resulting in an overall accuracy of around 97.4%.

Isolation Forest

```
% train IsolationForest
[mdlIF,~,scoreTrainIF] = iforest(featureNormal{:,2:end},'ContaminationFraction',0.1);

[isanomalyIF,scoreTestIF] = isanomaly(mdlIF,featureTestNoLabels.Variables);
predIF = categorical(isanomalyIF, [1, 0], ["Anomaly", "Normal"]);
```

Figure 5.54: Creation of the Isolation Forest [67]

The iForest is trained according to the settings displayed in Fig. 5.54. Hereby, it is observed, that setting the *ContaminationFraction*, which also refers to the percentage of possible anomalies within the data, to zero, leads to the model being not able to distinguish between the classes at all. Therefore, the *ContaminationFraction* is set to 0.1. As a result, iForest is able to distinguish between the classes completely, as shown in Fig. 5.54 below:

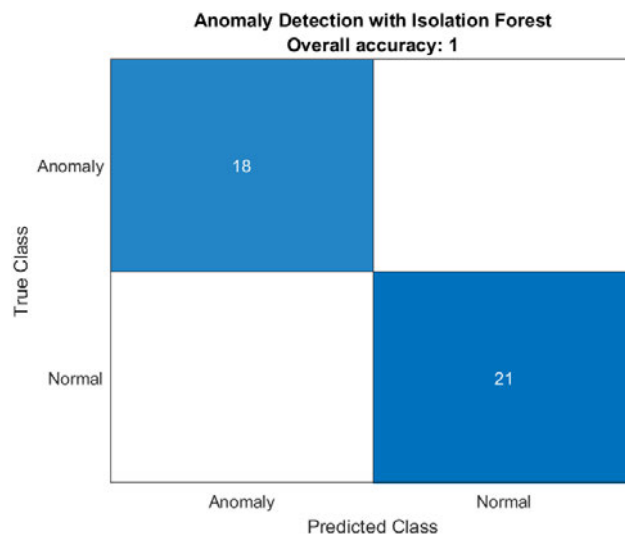


Figure 5.55: Confusion matrix iForest [67]

Autoencoder

Finally, the Autoencoder (AE) is trained. The general structure of the AE is established according to the principles outlined in Section 2.6, with the encoder and decoder configured as opposite-facing pyramids of fully connected layers. Each layer utilizes a ReLU activation function, and RMSprop is set as the optimizer with MATLAB's default learning rate of 0.001. The output layer is configured as a regression layer to allow the autoencoder to reconstruct the input data. After setting these core configurations, the remaining hyperparameters are tuned empirically, resulting in the following settings (see Fig. 5.56):

```
% Feature Dimension
featureDimension = 15; % 15 Features

% Define feedforward neural network layers
layers = [
    featureInputLayer(featureDimension, 'Normalization', 'none')
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(32)
    reluLayer
    fullyConnectedLayer(64)
    reluLayer
    fullyConnectedLayer(featureDimension)
    regressionLayer];

% Set Training Options
options = trainingOptions('rmsprop', ...
    'Plots', 'training-progress', ...
    'MiniBatchSize', 15, ...
    'MaxEpochs', 200, ...
    'Shuffle', 'every-epoch', ...
    'Verbose', false);

% train net
autoenc = trainNetwork(trainData, trainData, layers, options);
```

Figure 5.56: Creation of the AE [67]

The autoencoder is trained on the training data and then used to reconstruct the test data. The reconstruction error is computed using the mean squared error (MSE) between the original and reconstructed test data, as well as for the training data. Based on the mean MSE of the training data, an empirical threshold is defined to detect the anomalies, where test samples with reconstruction errors exceeding this threshold are identified as anomalies, as shown in Fig. 5.57:

```
% Reconstruction of the testdata
reconstructedDataTest = predict(autoenc, testData);

% Reconstruction error for test data
mseTest = mean((testData - reconstructedDataTest).^2, 2);
disp(['Mean Reconstruction Error (Test Data): ', num2str(mean(mseTest))]);

Mean Reconstruction Error (Test Data): 69.5922

% Threshold for anomaly detection based on training data
trainMSE = mean((trainData - predict(autoenc, trainData)).^2, 2);
disp(['Mean Reconstruction Error (Training Data): ', num2str(mean(trainMSE))]);

Mean Reconstruction Error (Training Data): 14.9535

threshold = 3 * mean(trainMSE);

% Anomaly detection in the test data
isAnomalyTest = mseTest > threshold;
```

Figure 5.57: Defining the threshold [67]

The threshold is defined as three times the mean reconstruction error of the data. The results are depicted in Fig. 5.58.

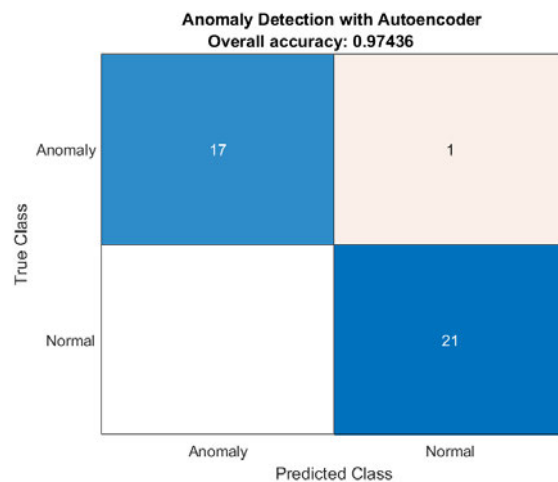


Figure 5.58: Confusion matrix AE [67]

The Autoencoder performs as accurately as the SVM. Therefore, the model is not further validated, and due to the good performance and simple principle of the Isolation Forest, the Isolation Forest is preferred as a solution.

5.5 Integration

After the training and selection of the machine learning model, it is integrated into the solution. Due to time constraints the scope of this thesis, this is only demonstrated by the integration of the classifiers for the individual IMS Stations and presented as follows:

5.5.1 Performing predictions within MATLAB

First, the trained model is exported to the MATLAB workspace, along with code for the automatic extraction of features. Then, a function is implemented that, given the name of an IMS Station as an input, receives the real-time data from InfluxDB, extracts the features, and performs a prediction. The prediction is then forwarded back to InfluxDB and stored in a designated bucket, available to be accessed within Node-RED.

In the following paragraph, the contents of the function will first be described sequentially, followed by the calling of the function separately within threads for each IMS station.

The function *runMachineLearningPrediction* hereby receives the name of a station as its input, as shown in Fig. 5.59. First, the trained model is loaded from within the workspace, followed by the establishment of communication between MATLAB and InfluxDB. This is achieved through HTTP requests, as shown below:

```
function runMachineLearningPrediction(stationName)

% Loading trained model
load('myModel_KW12.mat', 'trainedModel_KW12');

% Setting query
query = sprintf('from(bucket: "%s") |> range(start: -3s) |> filter(fn: (r) => r._measurement == "MotorPower%s")', stationName, stationName);

% Set HTTP-Request-Options
url = 'http://192.168.2.220:8086/api/v2/query?org=SGC-HAW';
headers = [
    matlab.net.http.HeaderField('Content-Type', 'application/vnd.flux'), ...
    matlab.net.http.HeaderField('Accept', 'application/csv'), ...
    matlab.net.http.HeaderField('Authorization', 'Token Qpii58fg697Thaf95-InPn8W8VOyfJmuHu3g-T73KfKxY6X9xoi912JPdf1006HyG3CoRB8vehF10hFHZJvXZw==')
];
body = matlab.net.http.MessageBody(query);
request = matlab.net.http.RequestMessage('POST', headers, body);

% Send request and receive response
response = send(request, matlab.net.URI(url));

% Extract the data from the response
data = response.Body.Data;
```

Figure 5.59: Communication between MATLAB and InfluxDB [67]

Hereby, the query is defined by using the name of a station for specifying the bucket as well as the measurement it includes. Then the specifics of the request are defined, including the network address of InfluxDB, the organization and necessary access token. The query is set to receive the data of the last three seconds, which represents the time of a carrier being transported from the beginning to the end of the conveyor belt, without interruptions. This range is empirically determined and has proven to be suitable for the task. Afterwards, the request and the data extracted from the response (see Fig. 5.59).

The data is then arranged and normalized accordingly to extract the features. Since the data is previously normalized by setting the baseline to zero with its median, which can differ depending on the accessed short sequences during production, it is centered by subtracting a defined, station-specific baseline value, as shown in Fig. 5.60 with the example of IMS Station 1:

```
% Rename the columns of the extracted timetable
tt.Properties.VariableNames = {'Sample', 'Value'};

% Instead of normalization, subtract the baseline value (baseline of the stations)
if(stationName == "IMS1")
    tt(:, 2) = tt(:, 2) - 3.9;
```

Figure 5.60: Subtract baseline values [67]

Afterwards, the data is transferred into a readable format for the feature extraction algorithm, and the features are extracted. After removing unwanted features that were generated automatically but were not used for training, the data is fed into the machine learning model, as shown in Fig. 5.61 and a prediction is performed.

```
% Create a cell containing the timetable
dataCell = {tt};

% Create a table with a 'Data' column that contains the cell
table = table(dataCell, 'VariableNames', {'Data'});

% Extract features
[featureTable, ~, ~] = diagnosticFeatures_KW(table);
featureTable = removevars(featureTable, "Data_res_tsfeat/Q3");
featureTable = removevars(featureTable, "Data_res_tsfeat/Maximum");
featureTable = removevars(featureTable, "Data_res_sigstats/Mean");

% Feed the ML model
[yfit12, ~] = trainedModel_KW12.predictFcn(featureTable);

% pass prediction
prediction = yfit12{1};
```

Figure 5.61: Performing the prediction [67]

The prediction is then automatically forwarded to InfluxDB using the same HTTP request structure shown in Fig. 5.59 and is stored in a designated bucket for predictions.

The function *runMachineLearningPrediction* itself is called by iterating through a list of station names and launching a corresponding number of threads using MATLAB's parallel pool functionalities. These threads are utilized to enable real-time predictions for each IMS Station in parallel, while ensuring that there are no delays. Each thread is hereby assigned an individual station, as shown in Fig. 5.62.

```
%List of stations
stations = {'IMS1', 'IMS3a', 'IMS3b', 'IMS4a', 'IMS4b', 'IMS5a', 'IMS5b', 'IMS6', 'IMS7'};

% Start parallel pool
pool = gcp('nocreate');
if isempty(pool)
    pool = parpool(length(stations));
end

% Start predictions of the stations in individual threads
try
    parfor i = 1:length(stations)
        stationName = stations{i};
        while true
            try
                % prediction for the Station
                runMachineLearningPrediction(stationName);
            catch ME
                disp(['Error processing station ' stationName ': ' ME.message]);
            end
            pause(1); % Wait one second, before the next prediction
        end
    end
catch ME
    disp(['Error in parallel execution: ' ME.message]);
end
```

Figure 5.62: Calling the machine learning prediction in threads [67]

The threads, once initiated, call the function *runMachineLearningPrediction* every second, until being manually terminated. This setup can be visualized as a central PC or workstation providing real-time machine learning predictions for the IMS stations to operators or maintenance personnel. To ensure that these predictions are accessible through the user interface and to trigger corresponding notifications, they need to be integrated and processed within Node-RED. This process will be explained in the following paragraphs.

5.5.2 Accessing the predictions within Node-RED

Within Node-RED, an additional flow is created to access data from InfluxDB and process it to trigger notifications via email, pop-ups, and audio alerts, as well as to display the data on the user interface. The corresponding flow is depicted in Fig. 5.63:

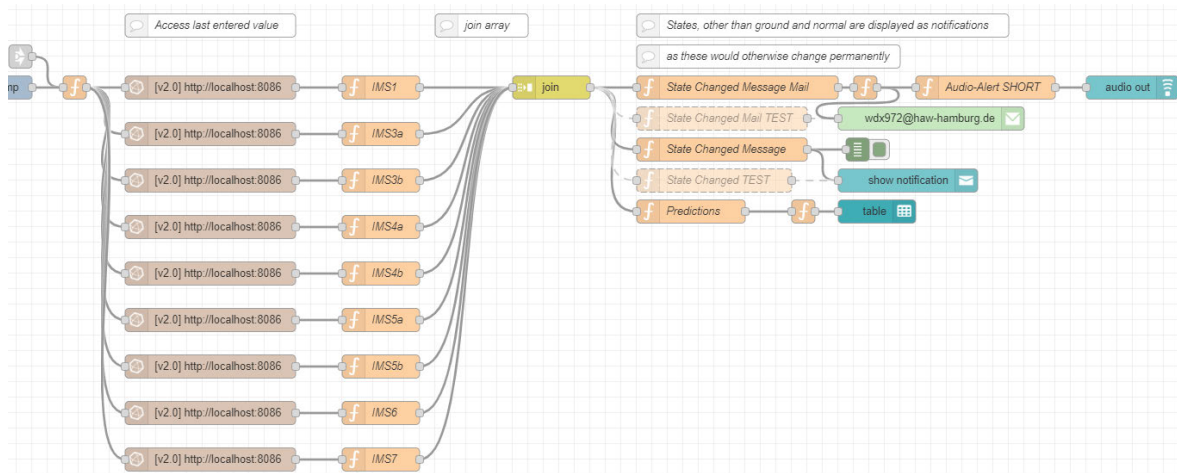


Figure 5.63: Prediction data flow [67]

The predicted outputs of each classifier are accessed through InfluxDB-in nodes, which are triggered by the central timestamp and configured to access the last recorded value. The output of the InfluxDB-in nodes is then forwarded to subsequent function nodes. These nodes, labeled according to their associated station, isolate the relevant information. All predictions are then aggregated into an array through a join node and processed similarly to the central notification flow described in Section 5.2.2 (see Fig. 5.15 and 5.63). The function nodes are configured to generate messages and trigger notifications only if the condition or state of a station differs from "ground" or "normal," in order to enable relevant alerts and prevent noise caused by the otherwise constant switching between these two classes.

The resulting messages are then distributed to an audio node and an e-mail node for notifications, both controlled by the silencer functionalities of the user interface through a preceding squared function node (see Fig. 5.63). Additionally, through the function node "State Changed Message" and a subsequent notification node, the messages are displayed as pop-ups. Finally, the last function node, "Predictions," is configured to display all outputs in a table on the user interface, within the placeholder previously shown in Fig. 5.32.

Through this configuration, all the relevant information is automatically stored, processed and accessible through the overall condition-based and predictive maintenance solution. In the following chapter all final functionalities and dashboards are presented and tested according to their intended design and the requirements, defined in Section 4.1.

6 Functional Testing

Within this chapter, the final and complete condition-based and predictive maintenance solution is tested through the user interface. First, the general functionality and accessibility of the user interface and its provided links sites and dashboards is tested. Then, the individual dashboard views of the production plant are examined, followed by the condition-based and predictive maintenance capabilities through their display of data and the resulting notifications. The entire configuration is tested on the production plant's workstation as well as on a Windows PC with an Intel Core i7-10510U CPU and 8 GB RAM.

6.1 General functionality

The accessibility of the dashboard ensures the accessibility of the individual dashboards, sites and tools, as well as the display of all the information relevant for operators and the maintenance personnel. The user interface is tested on multiple devices and provides a general maneuverability through its user interface and tabs, regardless of the device it is accessed with. Through the sidebar menu, all linked tools are accessible from within the user interface. It is depicted in Fig. 6.1 below:

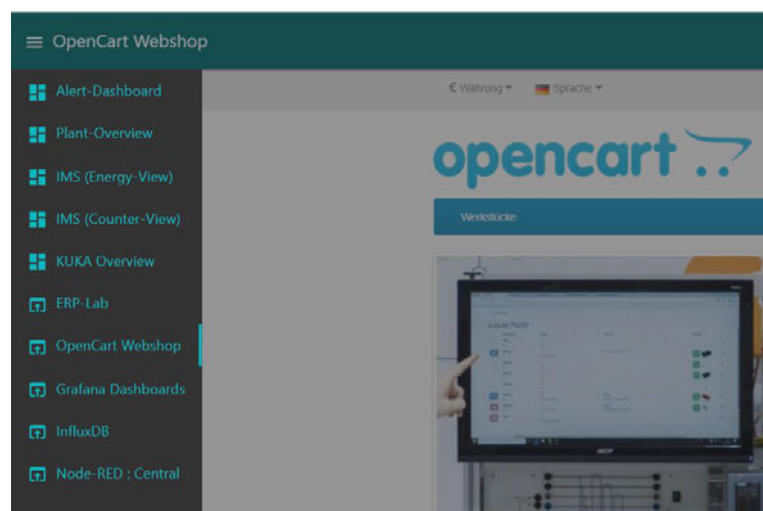


Figure 6.1: Sidebar menu [67]

All provided tabs and links can be accessed equally and freely through the sidebar menu by clicking on the corresponding upper left icon within the user interface. However, displaying

a site within the frame of the user interface is only possible for the OpenCart web shop (see Fig. 61), as the other external sites loose part of their functionality. Therefore, by clicking on their icons, a new tab with the respective site opens.

6.2 Accessibility of data

The access and display of real-time data are central aspects and requirements of the user interface and overall solution. The individual dashboards provide an overview of the various stations and parameters. An excerpt of the IMS (Counter-View), IMS (Energy-View), and KUKA Overview is shown in Fig. 6.2:

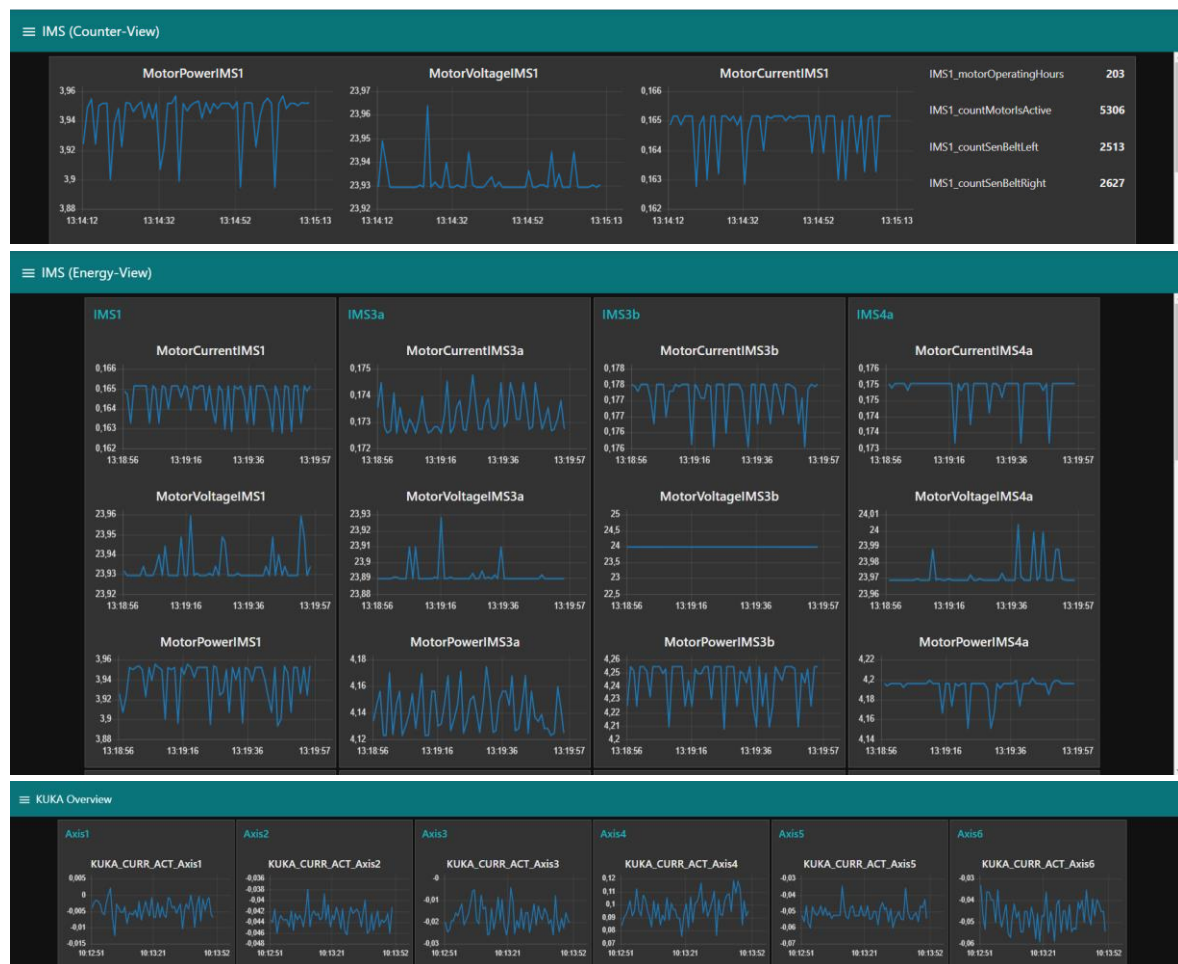


Figure 6.2: Excerpt of dashboards [67]

As shown, each individual parameter can be inspected and is refreshed every second. The data of different KUKA robot axis or IMS Stations can therefore be inspected and compared

to one another continuously and in real-time. These dashboards provide a detailed and expanded view of all the production plants components simultaneously. If a more compact view is needed, the dashboard “Plant-Overview” can be used, shown in Fig. 6.3



Figure 6.3: Plant-Overview [67]

This dashboard combines all of the production plants data within distinct groups and enables a complete overview within a single tab of the user interface. In addition, by hovering over a parameter, the legend and the exact value can be viewed.

All graphs provide a comprehensive view of the current value of each individual parameter of the production plant. If a more detailed view of the production plant's data is needed or a specific time window needs to be inspected, Grafana can be used and accessed through the sidebar menu.

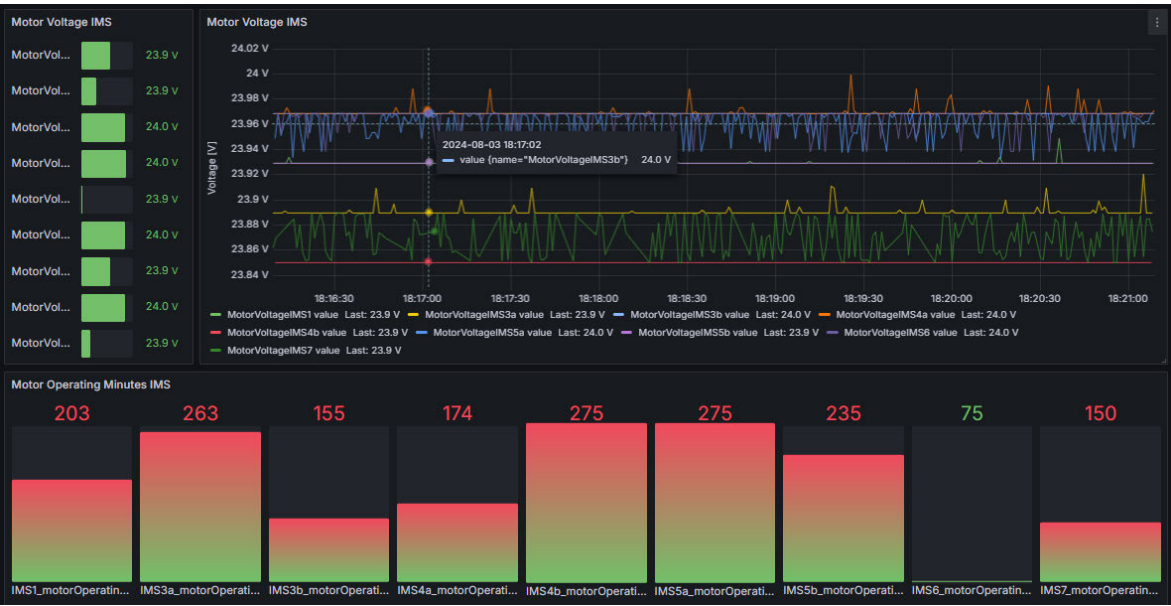


Figure 6.4: Grafana dashboard [67]

The dashboards within Grafana allow for the creation of diverse and custom dashboards, as shown in Fig. 6.4. This enables the combination and visual highlighting of specific parameters. Figure 6.4 shows the voltage of all IMS Stations in two different forms. Additionally, the motor operating duration in minutes is displayed by the bar gauge on the bottom of the dashboard, indicating higher values through the color transition. By hovering over the graphs, specific parameter values can also be displayed (see Fig. 6.4).

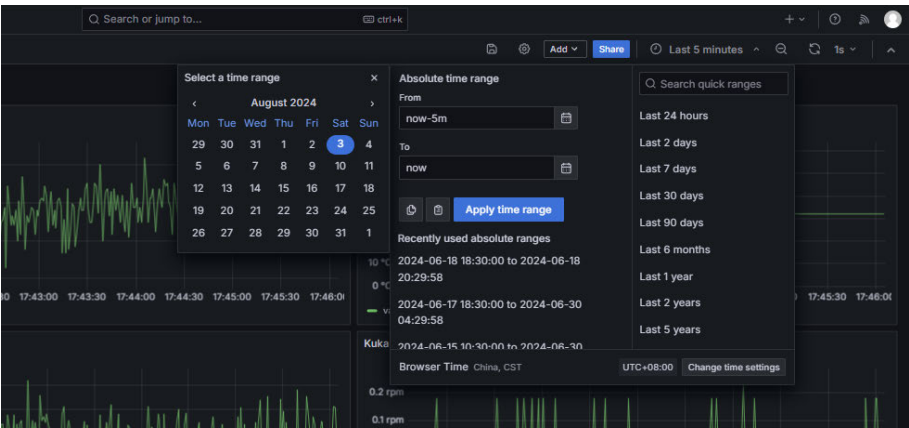


Figure 6.5: Grafana time range selector [67]

Grafana hereby allows for the viewing of historical data through its time range selector, as shown in Fig. 6.5, providing a useful tool for analyzing trends and patterns over specific time periods. This is particularly useful for in-depth analysis and manual monitoring of

production plant parameters from a maintenance perspective. In addition, the historical data can be accessed in a tabular view, both from within Grafana, as well as InfluxDB, providing the storage of data. As the data can be accessed and visualized by all tools, the requirement of historical and real-time data accessibility is met, as well as the appropriate storage of data within a database. If wanted, the existing dashboards and buckets can be extended and customized for specific maintenance needs.

6.3 Condition-based and preventive maintenance capabilities

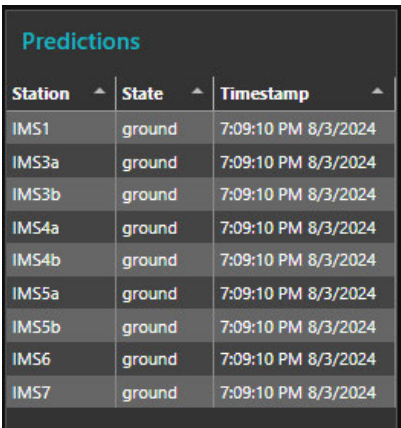
Through the previously described flows (see Section 5.2.1 and Fig. 5.11), the data of the production plant is automatically monitored by customizable and defined thresholds. If these thresholds are exceeded, the relevant data is displayed within the “Alert-Dashboard,” specifically within the “Reported Alert-Table” and the “Real-Time Alert-Table,” as shown in Fig. 6.6:

Reported Alert-Table					Real-Time Alert-Table				
R...	Parameter	Value	Warning Level	Timestamp	Parameter	Value	WarningLevel	Timestamp	
1	KUKA_CURR_ACT_Axis1	-0.003200000151991844 A	light warning	7:07:30 PM 8/3/2024	KUKA_CURR_ACT_Axis1	-0.0060000000052154064...	light warning	7:09:10 PM 8/3/2024	
4	KUKA_MOT_TEMP_Axis1	32 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis1	32 °C	light warning	7:09:10 PM 8/3/2024	
5	KUKA_MOT_TEMP_Axis2	30 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis2	30 °C	light warning	7:09:10 PM 8/3/2024	
6	KUKA_MOT_TEMP_Axis3	31 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis3	31 °C	light warning	7:09:10 PM 8/3/2024	
7	KUKA_MOT_TEMP_Axis4	32 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis4	32 °C	light warning	7:09:10 PM 8/3/2024	
8	KUKA_MOT_TEMP_Axis5	33 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis5	33 °C	light warning	7:09:10 PM 8/3/2024	
9	KUKA_MOT_TEMP_Axis6	32 °C	light warning	7:07:40 PM 8/3/2024	KUKA_MOT_TEMP_Axis6	32 °C	light warning	7:09:10 PM 8/3/2024	
2	KUKA_TORQUE_ACT_Axis1	-0.004999999888241291 ...	light warning	7:07:40 PM 8/3/2024	KUKA_TORQUE_ACT_Axis1	0.07699999958276749 N...	hard warning	7:09:10 PM 8/3/2024	
3	KUKA_VEL_ACT_Axis1	0 rpm	light warning	7:07:40 PM 8/3/2024	KUKA_VEL_ACT_Axis1	0 rpm	light warning	7:09:10 PM 8/3/2024	

Figure 6.6: Alert tables [67]

The “Reported Alert-Table” displays the data at the time when the threshold is triggered, while the “Real-Time Alert-Table” displays its exact current state. Both tables display the parameter name, value, an according timestamp as well as the warning level to an alert, allowing to differentiate between hard and light warnings (see Fig. 6.6). This setup allows users to view all maintenance alerts and their current states, allowing quick assessments of the situation. This is tested by manually adjusting the thresholds within Node-RED. Additionally, the “Reported Alert-Table” can be sorted according to each column header.

In addition, the predictive maintenance outputs provided by the trained SVM in MATLAB, are also displayed within the dashboard, as shown in Fig. 6.7:



Station	State	Timestamp
IMS1	ground	7:09:10 PM 8/3/2024
IMS3a	ground	7:09:10 PM 8/3/2024
IMS3b	ground	7:09:10 PM 8/3/2024
IMS4a	ground	7:09:10 PM 8/3/2024
IMS4b	ground	7:09:10 PM 8/3/2024
IMS5a	ground	7:09:10 PM 8/3/2024
IMS5b	ground	7:09:10 PM 8/3/2024
IMS6	ground	7:09:10 PM 8/3/2024
IMS7	ground	7:09:10 PM 8/3/2024

Figure 6.7: Predictive maintenance output [67]

It shows each individual IMS station, its predicted state, and the current timestamp. The state is directly transferred from MATLAB to InfluxDB and accessed by Node-RED. The state of each station is therefore logged with a current timestamp, allowing for traceability and future review. The same applies for the threshold exceeding values, as shown in Fig. 6.8, meeting the requirements of Section 4.1

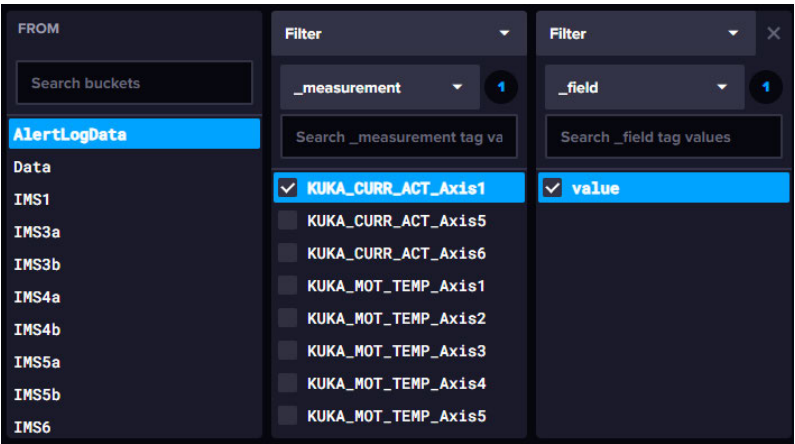


Figure 6.8: Logging of threshold exceeding data in InfluxDB [67]

The machine learning predictions are tested during regular production and manually induced faults. The model is able to distinguish between classes sufficiently. However, it is noticed that due to the small input sequences, the stopping of carriers during production is sometimes classified as "slowed" since the conveyor belt temporarily slows down.

In addition to the visualization within Node-RED, the individual states can be monitored in real-time in Grafana, as shown in Fig. 6.9.

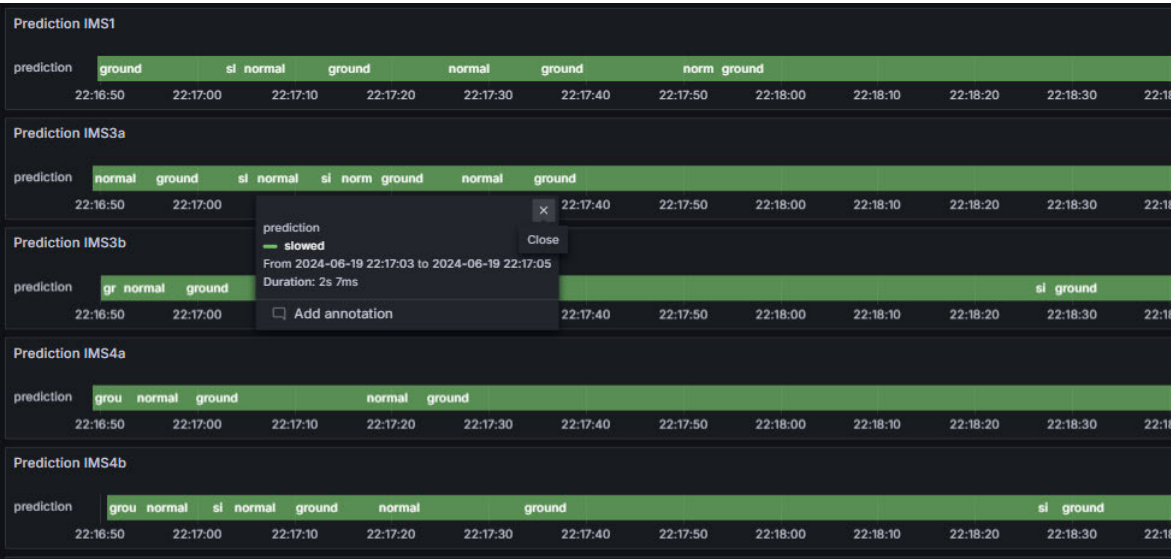


Figure 6.9: Visualization of the states within Grafana [67]

Through a correspondingly configured dashboard in Grafana, the predictions of the machine learning model can be visualized as sequences over time. This allows for the observation of transitions between the different states of a station as well as their durations. The specific time and duration of each state can be viewed by clicking on a specific sequence, as shown in Fig. 6.9. Interacting with the production plants conveyor belts is visualized accordingly within this dashboard, allowing one to see the change of its state almost directly over time. Through Grafana, the predicted states of the IMS Stations can therefore be observed in real-time, as well as historically.

However, running seven machine learning models simultaneously within threads presented a challenge for the Windows PC used, occasionally causing the application to crash. It is therefore recommended to use a more computationally advanced PC for this task or to limit the number of simultaneously running machine learning algorithms.

6.4 Notifications

Notifications and maintenance alerts are an important component of the overall solution, ensuring that any deviations or issues within the production process are promptly addressed. If a threshold is exceeded or the state of the IMS Stations differs from “normal” or “ground”, operators or maintenance personnel are notified through different maintenance alerts, allowing them to take appropriate measures in a timely manner.

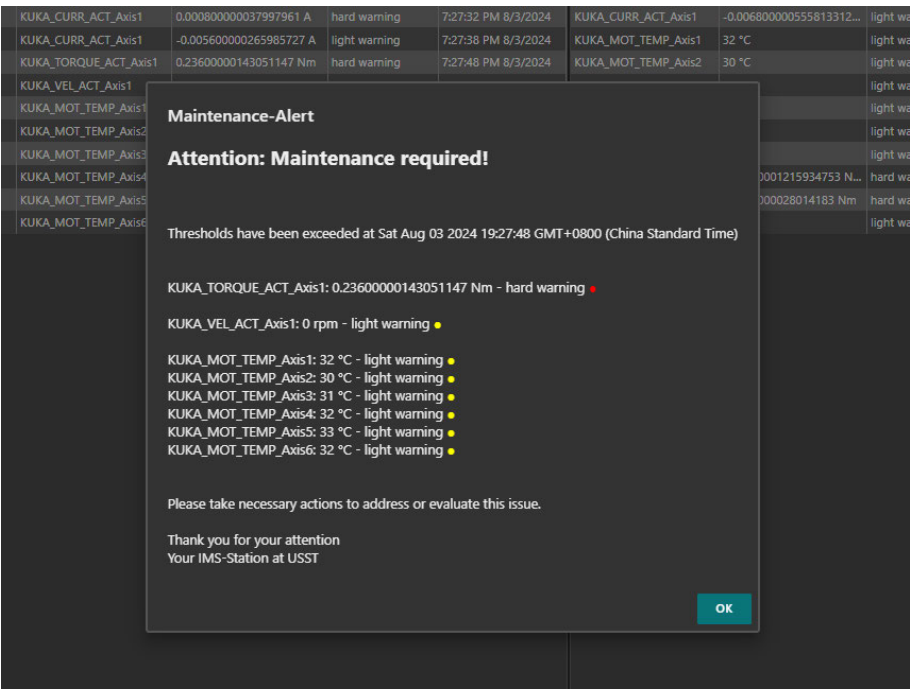


Figure 6.10: Maintenance-Alert pop-up [67]

Figure 6.10 shows a maintenance alert pop-up that appears whenever a threshold is triggered. The pop-up remains on the screen until it is acknowledged by clicking "OK" or replaced by a new pop-up. Hard warnings are highlighted with a red circle, while light warnings are indicated with a yellow circle, as intended. The pop-up provides information about which thresholds were triggered and at what time. Within the display, the alerts are categorized according to their parameters, offering a clear overview of the current situation.

If the predicted state of a station changes, a small pop-up appears in the upper right corner of the screen, as shown in Fig. 6.11. This alert notifies maintenance personnel of the state change, allowing them to differentiate between different notifications while preventing the screen from being blocked by too many full-sized pop-ups.

ML-Alert		
Station IMS1: blocked		
Predictions		
Station	State	Timestamp
IMS1	blocked	4:18:03 PM 8/3/2024
IMS3a	ground	4:18:03 PM 8/3/2024
IMS3b	ground	4:18:03 PM 8/3/2024
IMS4a	ground	4:18:03 PM 8/3/2024
IMS4b	ground	4:18:03 PM 8/3/2024
IMS5a	ground	4:18:03 PM 8/3/2024
IMS5b	ground	4:18:03 PM 8/3/2024
IMS6	ground	4:18:03 PM 8/3/2024
IMS7	ground	4:18:03 PM 8/3/2024

Figure 6.11: Machine Learning alert [67]

Alongside the pop-ups, an audio alert is triggered regarding both alert types, either by stating that a state has changed or that a threshold has been exceeded. Lastly, e-mail notifications are sent to notify the maintenance personnel through the production plant.

IMS-Plant@outlook.com

Maintenance-Alert

13:44

Attention: Maintenance required!...

IMS-Plant@outlook.com

ML-Alert

13:44

The condition/state of following a...

IMS-Plant@outlook.com

Maintenance-Alert

13:42

Attention: Maintenance required!...

Attention: Maintenance required!

Thresholds have been exceeded at Sat Aug 03 2024 19:44:19 GMT+0800 (China Standard Time)

KUKA_CURR_ACT_Axis1: -0.003999999724328518 A - light warning

Please take necessary actions to address or evaluate this issue.

Thank you for your attention

Your IMS-Station at USST

Figure 6.12: E-Mail notifications [67]

The e-mail notifications also differ in content, as shown on the left in Fig. 6.12, sending distinct messages for exceeding thresholds and changing conditions or states. All notification functionalities work as intended, ensuring that maintenance personnel, both within and outside the proximity of the production plant, are promptly informed, thereby meeting the requirements outlined in Section 4.1.

The entire Alert-Dashboard is shown in Fig. 6.13. It displays the latest alert message within a template on the left and includes all previously shown tables. Additionally, this dashboard features buttons to delete or refresh the “Reported Alert-Table,” allowing operators or

maintenance personnel to interact with the table as needed, after addressing or resolving the underlying issues of a reported alert. The Silencer switches located at the bottom left disable notifications as intended and are reset after a certain period of time. The order of interactions with these buttons hereby does not affect their functionality, with the notifications only permanently disabled with the “Silencer-Reset” switch turned off. The delay in alert messages effectively limits the throughput, ensuring that the system avoids overwhelming users with excessive notifications.

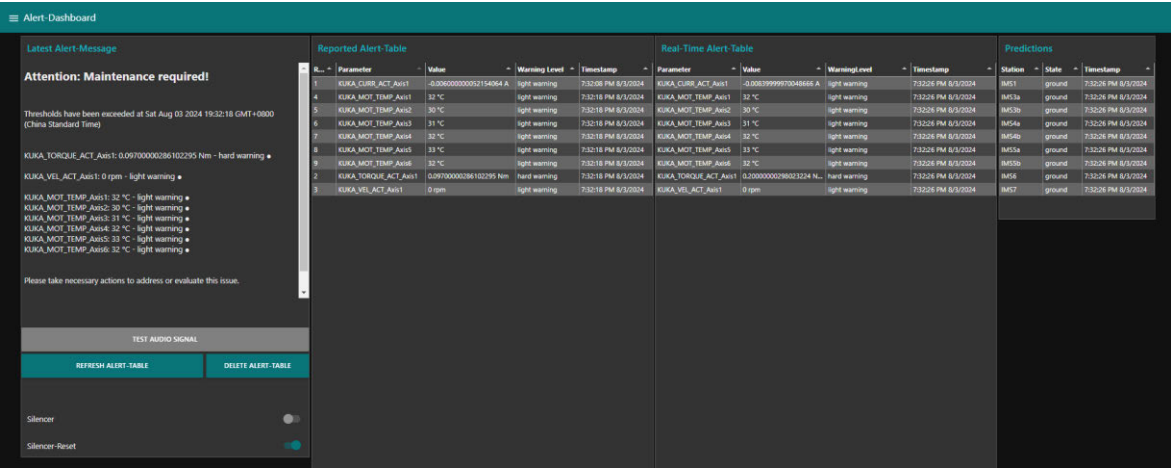


Figure 6.13: Alert-Dashboard [67]

Overall, the implemented condition-based and predictive maintenance solution fulfills the requirements for user interface accessibility, as well as the management of historical and real-time data. It effectively displays all relevant information and automates processes related to condition monitoring, classification, notifications, and data logging.

7 Conclusion

Within this thesis, a condition-based and predictive maintenance solution for the Industry 4.0 production plant, located at the Shanghai-Hamburg College of the University of Shanghai for Science and Technology, has been implemented, ranging from establishing a data flow and storing data, to performing automated condition monitoring and machine learning classification within a collection of interacting tools.

A significant aspect of the implementation displays the creation of the data flow and development of a user interface within Node-RED, integrating all selected tools by establishing the data flow and providing real-time data and condition monitoring functionalities.

7.1 Summary

First, the communication and data flow between the central PLC of the production plant and Node-RED was established. This forms the foundation for data distribution and is based on the pre-existing aggregation of relevant data within the central PLC's main data block. The data is accessed through OPC UA, utilizing the built-in OPC UA server of the central PLC. Through the established access, a series of data flows was configured to store and organize the data within a InfluxDB database. Afterwards, using the functionalities and node palette of Node-RED, the existing access and flow of data were utilized to establish an integrated and automated condition monitoring solution based on customizable thresholds.

After establishing the foundation for communication and data flow, the next steps focused on enhancing the solution regarding maintenance alerts, notifications and the capability to perform real-time monitoring and data visualization. Functionalities to display threshold exceeding data, as well as to send automated notifications via e-mail, perform acoustic alerts and generate pop-ups were implemented to notify maintenance personnel according to the current condition of the production plant.

Subsequently, dashboards were created in Node-RED to provide an overview of the current condition of the production plant, integrating the notification functionalities and displaying threshold exceeding parameters as well as their real-time values.

Within distinct dashboards for the individual stations and robot of the production plant, each parameter was visualized and displayed in real-time, enhancing the data monitoring capabilities and, in combination with the notifications, allowing operators to quickly assess the system's status and identify any deviations from normal operating conditions. To access and enable the inspection of historical data, Grafana was integrated to query and visualize the stored data directly from InfluxDB. In Grafana, customized dashboards were created to provide comprehensive views of individual and grouped parameters through graphs and gauges and enable users to examine current and historical data trends.

At last, and in addition to real-time monitoring, predictive maintenance functionalities were implemented. Machine learning models were trained in MATLAB, and a support vector machine was integrated into the solution to classify and predict potential conditions of the conveyor belt motors of each station within the production plant. The model was integrated into the solution by accessing the real-time data from InfluxDB within MATLAB, performing predictions within threads for each individual station simultaneously, and storing the output back into InfluxDB. These predictions were then accessed by Node-RED and integrated to the user interface, by extending notification functionalities and the display of data, the latter both within Node-RED and Grafana. The final solution was then presented and tested regarding the defined requirements.

Overall, the solution increases maintenance capabilities by enabling communication, automated monitoring, and data analytics. It essentially represents an Internet of Things and Services, establishing connections between different tools and providing the storage of data, condition monitoring, and data analysis as a service.

7.2 Outlook

The condition-based and predictive maintenance solution is configured as an open and modular system, enabling the integration of further tools and functionalities within and outside of Node-RED. The system could be improved by integrating a direct communication between Node-RED and MATLAB, to enable the remote activation and deactivation of the predictive maintenance model. Furthermore, based on the proposed approach, an anomaly detector for the KUKA robot could be integrated, further enhancing the predictive maintenance capabilities.

Moreover, the user interface could be graphically enhanced to visualize the changing conditions within a graph or based on a color-code, rather than a tabular view. A digital representation of the production plant could be implemented or integrated into Node-RED and its user interface as well, to visualize the production process and enable operators and maintenance personnel to gain a better insight of changing conditions.

Additionally, during the course of the thesis, optical incremental sensors were ordered for the production plant, but could not be implemented in time. These sensors could be integrated to measure the speed of the conveyor belts, thereby generating additional data. This integration would provide more precise monitoring over the conveyor belt operations and could be used to enhance both the condition-based and predictive maintenance functionalities.

Bibliography

- [1] Deutsches Institut für Normung e. V: DIN 31051:2019-06
- [2] Deutsches Institut für Normung e. V: DIN EN 13306:2018-02
- [3] Bengtsson, M.: Mälardén University Press Dissertations No. 48: On Condition Based Maintenance and its implementation in industrial settings, Arkitektkopia Västerås 2007
- [4] Hansford Sensors: The pros and cons of different maintenance strategies <https://hansfordsensors.com/blog/the-pros-and-cons-of-different-maintenance-strategies/>
Access: 04.02.2024
- [5] Schenk, M.; Endig, M.; Freund, C. et al.: Instandhaltung technischer Systeme: Methoden und Werkzeuge zur Gewährleistung eines sicheren und wirtschaftlichen Anlagenbetriebs, Springer-Verlag Berlin Heidelberg 2010
- [6] MainCert: Main-Cert Handbuch: Kompetenzbeschreibung zur Vorbereitung auf das Zertifizierungsverfahren für Führungskräfte der Instandhaltung im Industrieservice, European Maintenance Management Certification, 2014
- [7] Pawellek, G.: Integrierte Instandhaltung und Ersatzteillogistik: Vorgehensweise, Methoden, Tools, Springer-Verlag Berlin Heidelberg 2013, 2016
- [8] Leidinger, B.: Wertorientierte Instandhaltung: Kosten senken, Verfügbarkeit erhalten, Springer Fachmedien Wiesbaden GmbH 2014, 2017
- [9] Lambertz, B.: Condition Based Maintenance:
<https://maint-care.de/knowhow/condition-based-maintenance/#ziel-der-zustandsorientierten-instandhaltung>
Access: 27.04.2024
- [10] Google Classroom: What is Bernoulli's equation?
<https://www.khanacademy.org/science/physics/fluids/fluid-dynamics/a/what-is-bernoullis-equation>
Access: 28.04.2024
- [11] IBM: What is CBM?
<https://www.ibm.com/topics/condition-based-maintenance>
Access: 28.04.2024

- [12] Trout, J.: Condition-based Maintenance: A Complete Guide
<https://www.reliableplant.com/condition-based-maintenance-31823>
Access: 28.04.2024
- [13] Ahmad, R.; Kamaruddin, S.: An overview of time-based and condition-based maintenance in industrial application, *Computers & Industrial Engineering*, vol. 63, Issue 1, pp. 135-149, August 2012
- [14] Wang, H. -K.; Huang, H. -Z.; Li, Y. -F. et al.: Condition-based maintenance with scheduling threshold and maintenance threshold, *IEEE Transactions on Reliability*, vol. 65, no. 2, pp. 513-524, June 2016
- [15] Sobral, J.; Guedes Soares, C.: Preventive Maintenance of Critical Assets based on Degradation Mechanisms and Failure Forecast, *IFAC-PapersOnLine*, vol. 49, Issue 28, pp. 97-102, 2016
- [16] Tran Anh, D.; Dabrowski, K.; Skrzypek, K.: The Predictive Maintenance Concept in the Maintenance Department of the “Industry 4.0” Production Enterprise, *Foundations of Management*, vol. 10, pp. 283-292, 2018
- [17] Serradilla, O.; Zugasti, E.; Rodriguez, J.; Zurutuza, U.: Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects, *Applied Intelligence* 52, pp. 10934-10964, 2021
- [18] Reinheimer, S.: *Industrie 4.0: Herausforderungen, Konzepte und Praxisbeispiele*, Springer Vieweg, April 2017
- [19] Ten Hompel, M.: *IT und autonome Systeme in der Logistik*, Springer Vieweg, 2023
- [20] Bauernhansl, T.; Vogel-Heuser, B.; ten Hompel, M.: *Handbuch Industrie 4.0: Band 1: Produktion*, 3. Auflage, Springer Vieweg, April 2023
- [21] Pistorius, J.: *Industrie 4.0 – Schlüsseltechnologien für die Produktion: Grundlagen • Potenziale • Anwendungen*, Springer-Verlag GmbH, 2020
- [22] Czichos, H.: *Technologie: Systemdenken und interdisziplinäres Ingenieurwesen*, Springer Vieweg, Juni 2022
- [23] IBM: Was ist das IoT?
<https://www.ibm.com/de-de/topics/internet-of-things>
Access: 10.06.2024
- [24] Babel, W.: *Internet of Things und Industrie 4.0, essentials plus online course*, Springer Vieweg, 2023

- [25] Tableau: Time Series Analysis: Definition, Types, Techniques, and When It's Used
<https://www.tableau.com/learn/articles/time-series-analysis#:~:text=Time%20series%20data%20is%20data%20that%20is%20recorded%20over%20consistent,data%20and%20cross%2Dsectional%20data.>
Access: 12.06.2024
- [26] Kayan, H.; Nunes, M.; Rana, O.; Burnap, P.; Perera, C.: Cybersecurity of Industrial Cyber-Physical Systems: A Review, ACM Computing Surveys, vol. 54, no. 11s, article 229, September 2022.
- [27] MathWorks: Predictive Maintenance with MATLAB
<https://de.mathworks.com/content/dam/mathworks/ebook/gated/predictive-maintenance-ebook-all-chapters.pdf>
Access: 08.07.2024
- [28] Plattform Industrie 4.0: Was ist Industrie 4.0?
<https://www.plattform-i40.de/IP/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html>
Access: 08.07.2024
- [29] Lehrbuch Psychologie: Gesamtglossar aller Bücher: L
<https://lehrbuch-psychologie.springer.com/lexikon/l>
Access: 09.07.2024
- [30] Metzler Lexikon Philosophie: Lernen
<https://www.spektrum.de/lexikon/philosophie/lernen/1209>
Access: 09.07.2024
- [31] Frick, D.; Gadatsch, A.; Kaufmann, J.; Lankes, B.; Quix, C.; Schmidt, A.; Schmitz, U.: Data Science: Konzepte, Erfahrungen, Fallstudien und Praxis, Springer Vieweg, 2021
- [32] Twain, T.: A brief breakdown of declarative vs. imperative programming, TechTarget, February 2022
<https://www.techtarget.com/searchapparchitecture/tip/A-brief-breakdown-of-declarative-vs-imperative-programming>
Access: 09.07.2024
- [33] Frochte, J.: Maschinelles Lernen: Grundlagen und Algorithmen in Python, 3. überarbeitete und erweiterte Auflage, Carl Hanser Verlag München, 2021
- [34] Géron, A.: Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme, 2. Auflage, O'Reilly Verlag, 2020

- [35] Mohri, M; Rostamizadeh, A.; Talwalkar, A.: Foundations of Machine Learning, The MIT Press Cambridge, Massachusetts London, England, 2012
- [36] Bishop, C. M.: Pattern Recognition and Machine Learning, Springer Science+Business Media, February 2006
- [37] MathWorks: Support Vector Machine: What Is a Support Vector Machine?
<https://de.mathworks.com/discovery/support-vector-machine.html>
Access: 12.07.2024
- [38] Kersting, K.; Lampert, C.; Rothkopf, C.: Wie Maschinen lernen: Künstliche Intelligenz verständlich erklärt, Springer Fachmedien Wiesbaden GmbH, 2019
- [39] European IT Certification Institute: What is the significance of the margin in SVM and how is it related to support vectors?, August 2023
<https://eitca.org/artificial-intelligence/eitc-ai-mlp-machine-learning-with-python/support-vector-machine/understanding-vectors/examination-review-understanding-vectors/what-is-the-significance-of-the-margin-in-svm-and-how-is-it-related-to-support-vectors/#:~:text=A%20larger%20margin%20implies%20a,and%20noise%20in%20the%20data.>
Access: 13.07.2024
- [40] Kim, E.: Everything You Wanted to Know about the Kernel Trick (But Were Too Afraid to Ask), September 2013
https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html
Access 14.07.2024
- [41] Liu, F. T.; Ting, K. M.; Zhou, Z.: Isolation Forest, Eighth IEEE International Conference on Data Mining, 2008
- [42] cpp-learning: Isolation Forest
<https://cpp-learning.com/wp-content/uploads/2022/01/decision-tree.png>
Access: 16.07.2024
- [43] Fritschi, L.; Lenk, K.: Parameter Inference for an Astrocyte Model using Machine Learning Approaches, May 2023
- [44] Sonnet, D.: Neuronale Netze kompakt: Vom Perceptron zum Deep Learning, IT kompakt, Springer Vieweg, 2022
- [45] Kruse, R.; Borgelt, C.; Braune, C.; Klawonn, F.; Moewes, G.; Steinbrecher, M.: Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze: 2., überarbeitete und erweiterte Auflage, Springer Vieweg, 2015

- [46] Gillhuber, A.; Kauermann, G.; Hauner, W.: Künstliche Intelligenz und Data Science in Theorie und Praxis: Von Algorithmen und Methoden zur praktischen Umsetzung in Unternehmen, Springer Spektrum, 2023
- [47] Ojha, V. K.; Abraham, A.; Snasel, V.: Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research, Article in Engineering Applications of Artificial Intelligence, April 2017
- [48] Bebis, G.; Georgiopoulos, M.: Feed-forward neural networks: Why network size is so important, IEEE Potentials, vol. 13, issue 4, 1994
- [49] Brownlee, J.: Difference Between Algorithm and Model in Machine Learning, Machine Learning Mastery, August 2020
<https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/>
Access: 19.07.2024
- [50] Dahlkemper, J.: Angewandte industrielle Bildverarbeitung: 7 Einführung in Künstliche Neuronale Netze, Vorlesungsfolien: HAW Hamburg, Fakultät TI, Technik und Informatik
- [51] Chollet, F.: Deep Learning with Python, Manning Publications Co., 2018
- [52] Analytics Vidhya: Gradient Descent Algorithm: How Does it Work in Machine Learning?
<https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/>
Access: 20.07.2024
- [53] Nabi, J.: Towards Data Science: Hyper-parameter Tuning Techniques in Deep Learning, March 2019
<https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8>
Access: 20.07.2024
- [54] Dahlkemper, J.: Angewandte industrielle Bildverarbeitung: 8 Training von Künstlichen Neuronalen Netzen, Vorlesungsfolien: HAW Hamburg, Fakultät TI, Technik und Informatik
- [55] MathWorks: Was ist ein Autoencoder?
<https://de.mathworks.com/discovery/autoencoder.html>
Access: 22.07.2024

- [56] MathWorks: Anomaly Detection in Industrial Machinery Using Three-Axis Vibration Data
<https://de.mathworks.com/help/predmaint/ug/anomaly-detection-using-3-axis-vibration-data.html>
Access: 22.07.2024
- [57] Lucas Nuelle: Technical documentation for offer, internal document
- [58] Lucas Nuelle: CSF 4: ERP Lab for Smart Factory 4.0: ILA course, Lucas Nuelle Lab Document, March 2019
- [59] Löffler, D.: Development of Predictive Maintenance Concepts for a Networked Production Plant, Masterthesis, Hamburg University of Applied Sciences, August 2019
- [60] Schneider Electric: EcoStruxure™ Machine Advisor: Die digitale Serviceplattform für Maschinen
<https://www.se.com/de/de/work/services/field-services/industrial-automation/oem/machine-advisor.jsp>
Access: 27.07.2024
- [61] Influxdata: InfluxDB vs MySQL: A detailed comparison
<https://www.influxdata.com/comparison/influxdb-vs-mysql/>
Access: 27.07.2024
- [62] Influxdata: Telegraf
<https://www.influxdata.com/time-series-platform/telegraf/>
Access: 28.07.2024
- [63] Node-RED: Low-code programming for event-driven applications
<https://nodered.org/>
Access: 28.07.2024
- [64] MathWorks: MATLAB: Mathematik. Grafiken. Programmierung.
<https://de.mathworks.com/products/matlab.html>
Access: 28.07.2024
- [65] Grafana
<https://grafana.com/grafana/>
Access: 28.09.2024
- [66] Influxdata Documentation: Getting started with Flux
<https://influxdata.com/products/flux/>
Access: 28.07.2024

[67] Own illustration

Appendix

A.1 Flows of the IMS Stations and KUKA robot

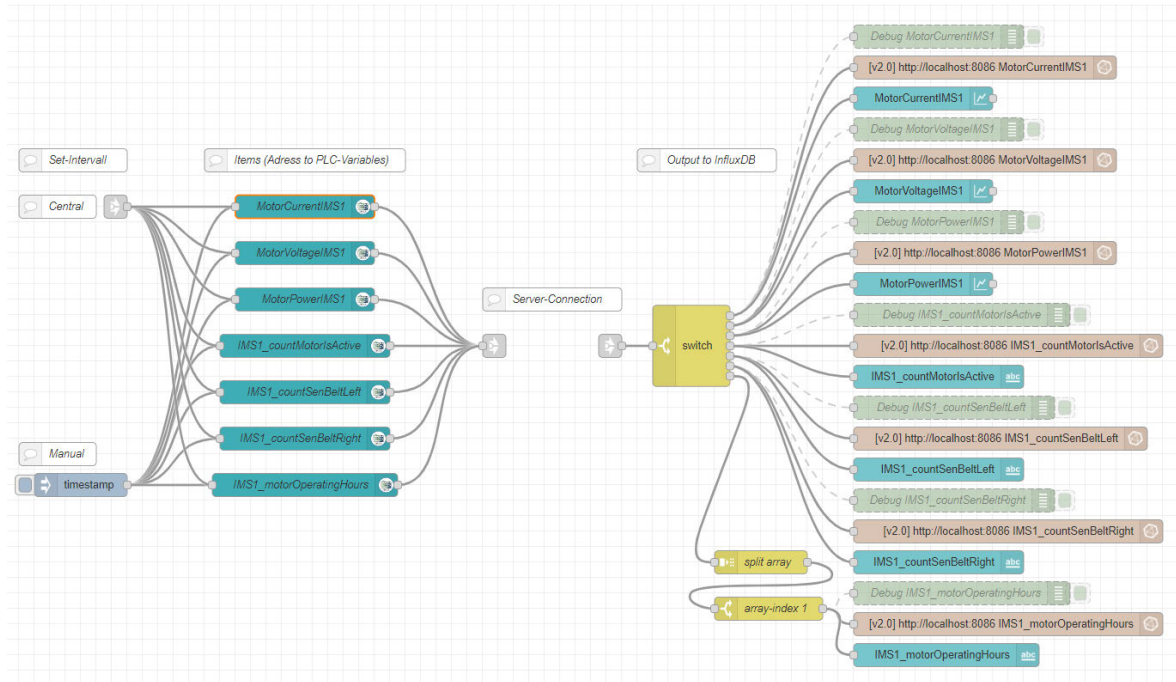


Figure A.1: IMS Station 1

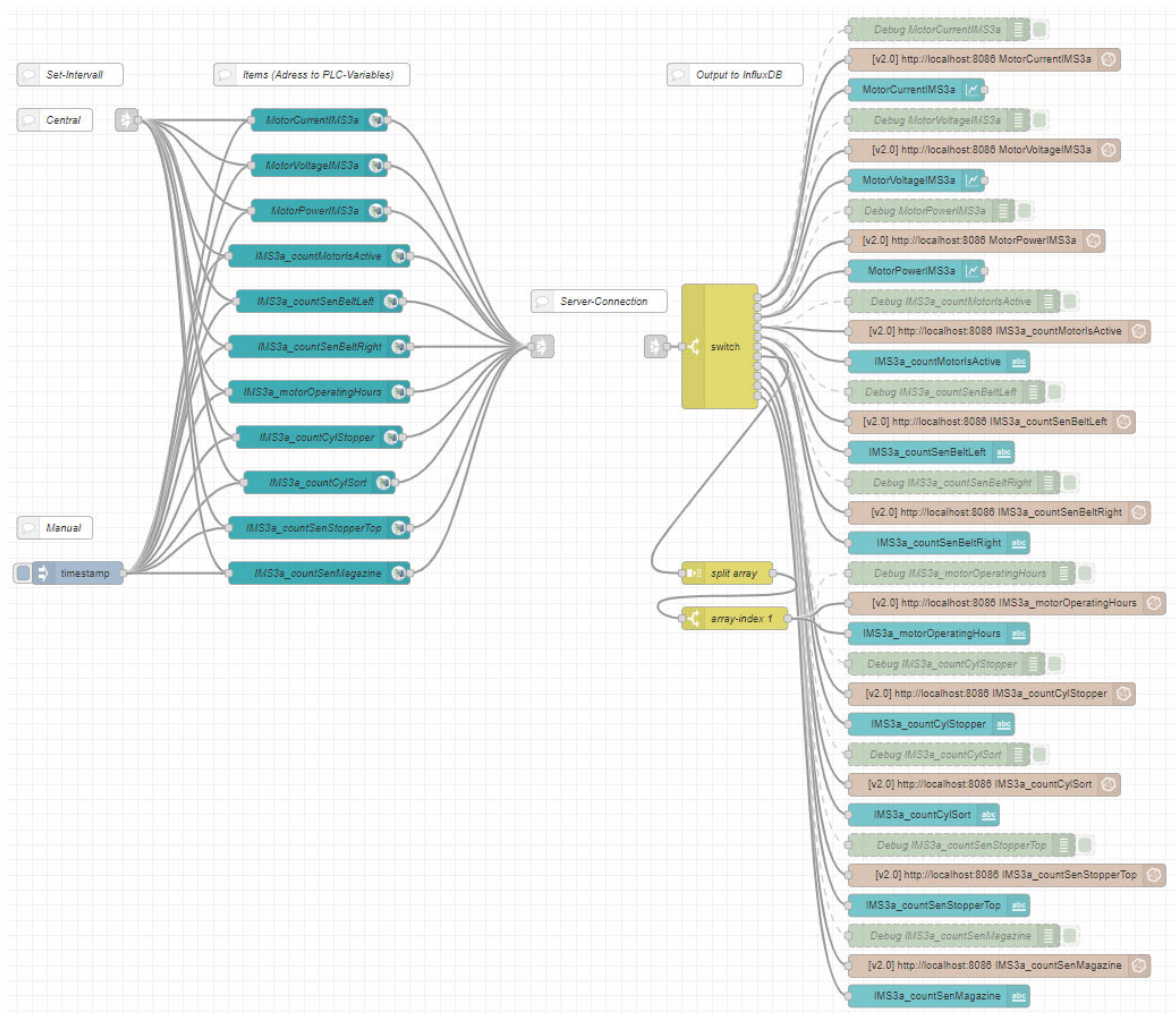


Figure A.2: IMS Station 3a

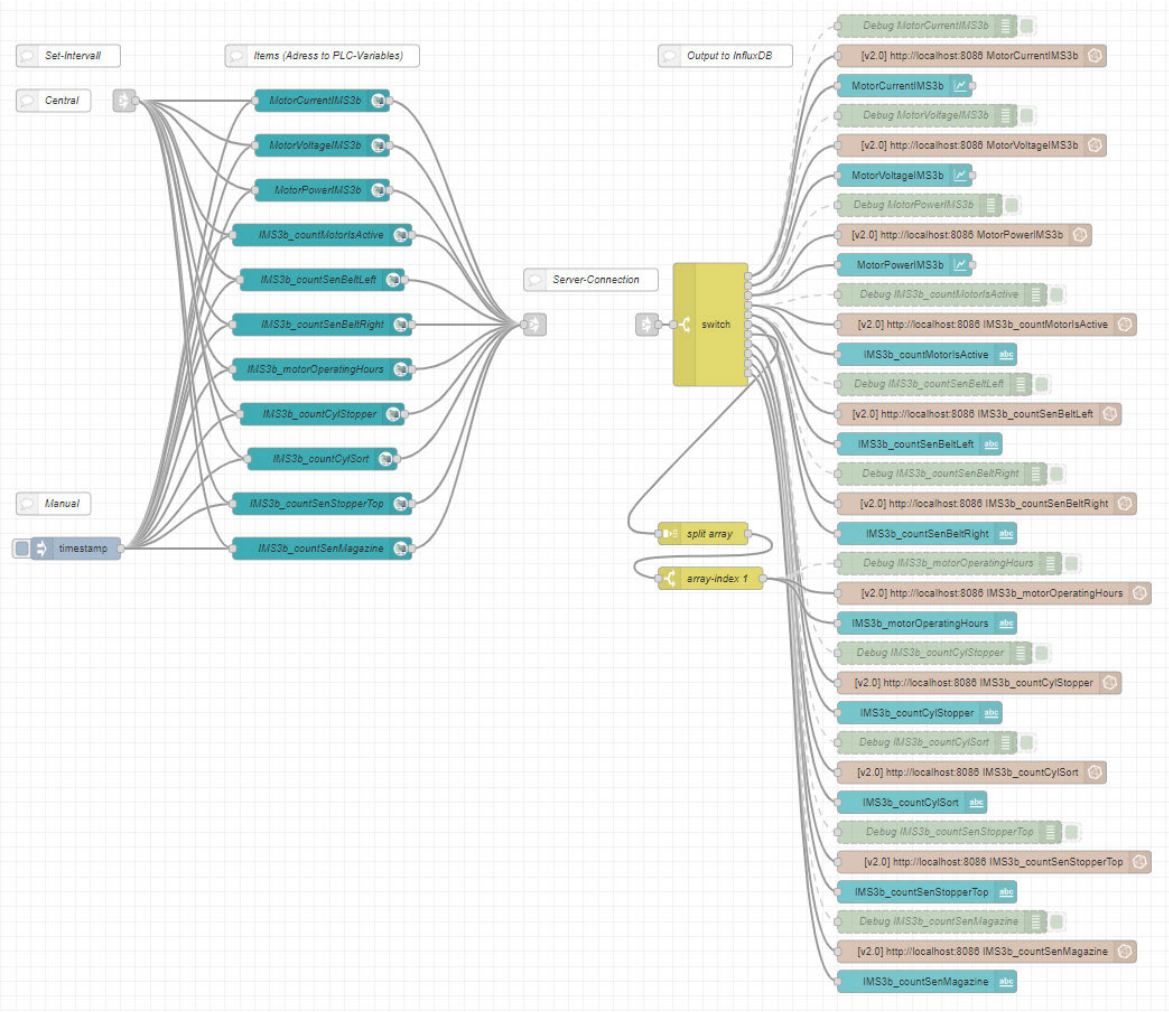


Figure A.3: IMS Station 3b

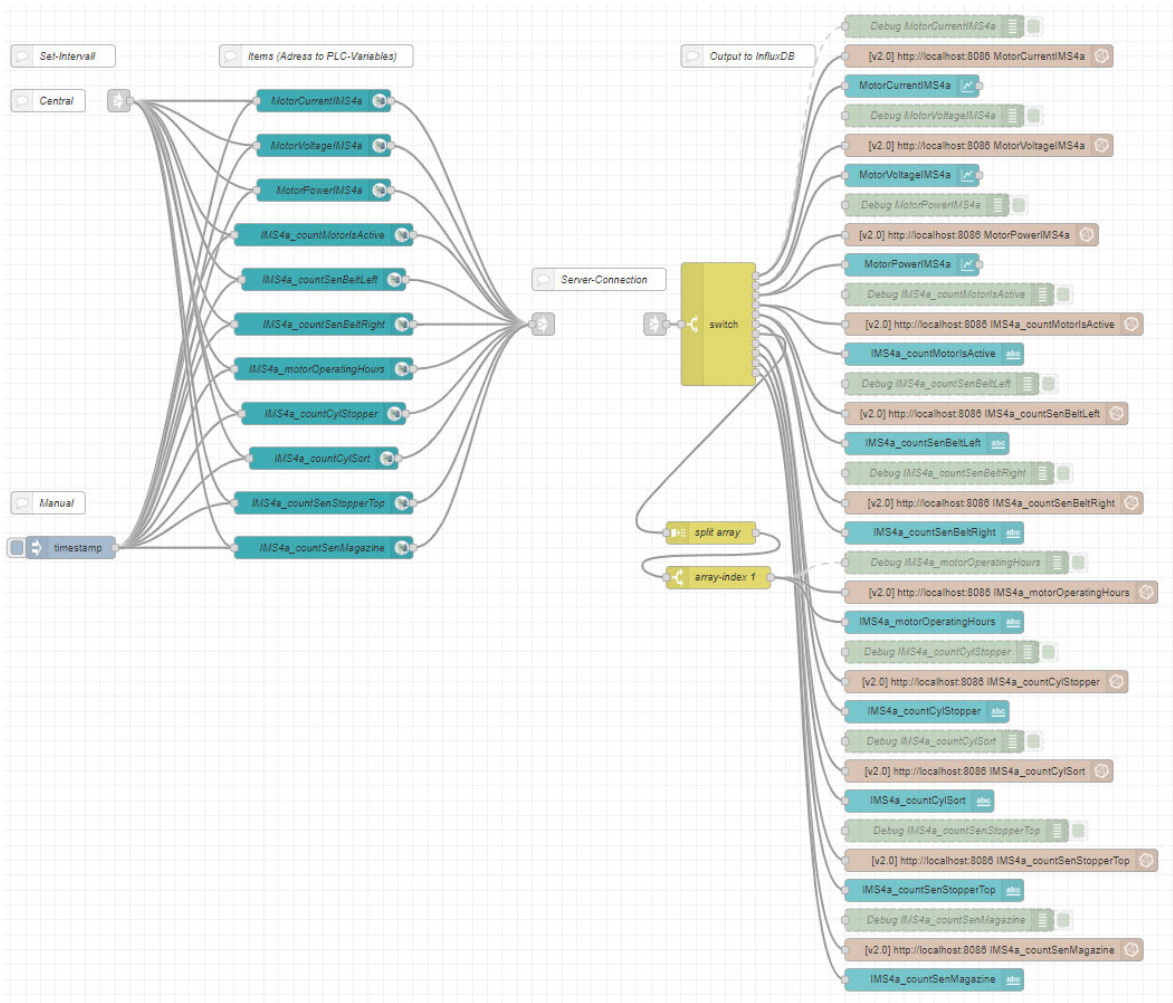


Figure A.4: IMS Station 4a

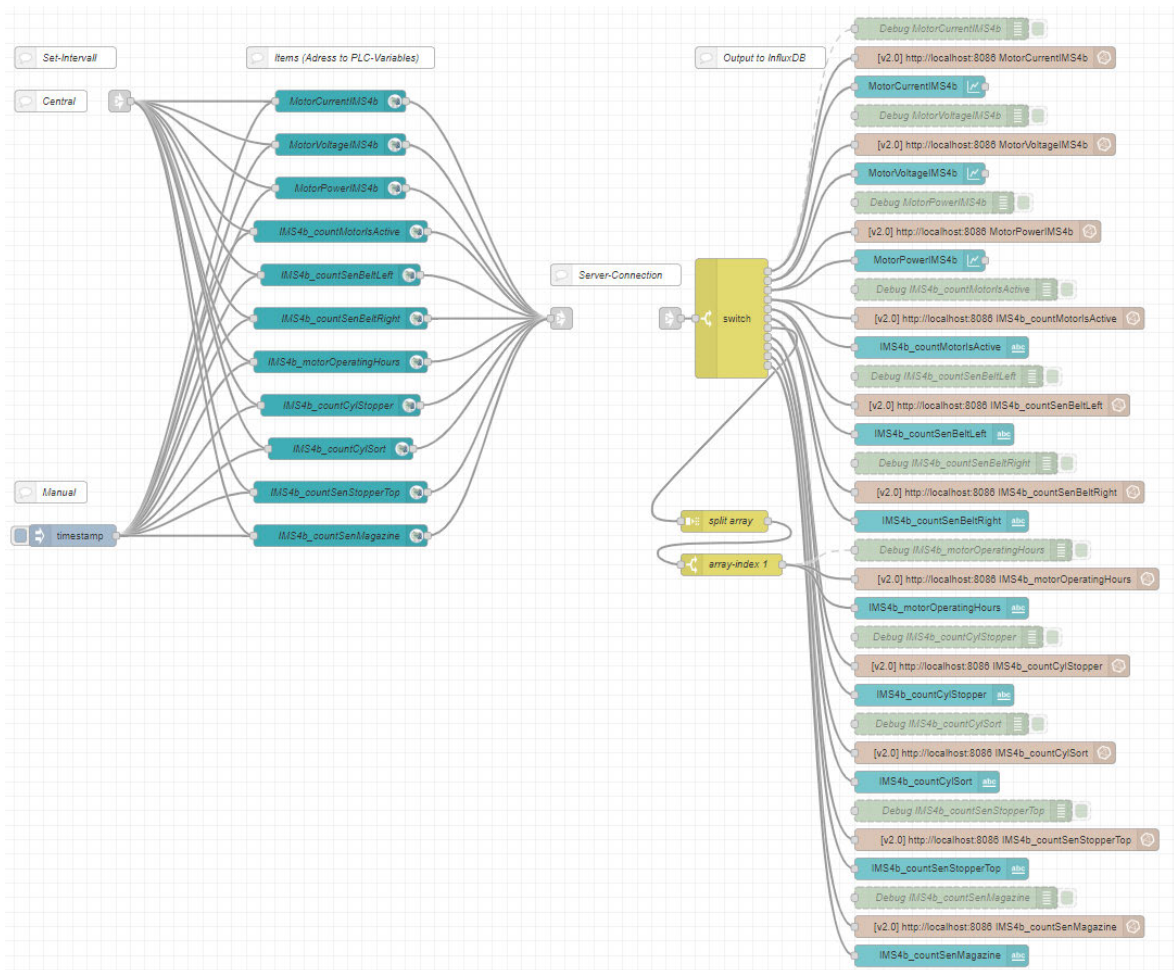


Figure A.5: IMS Station 4b

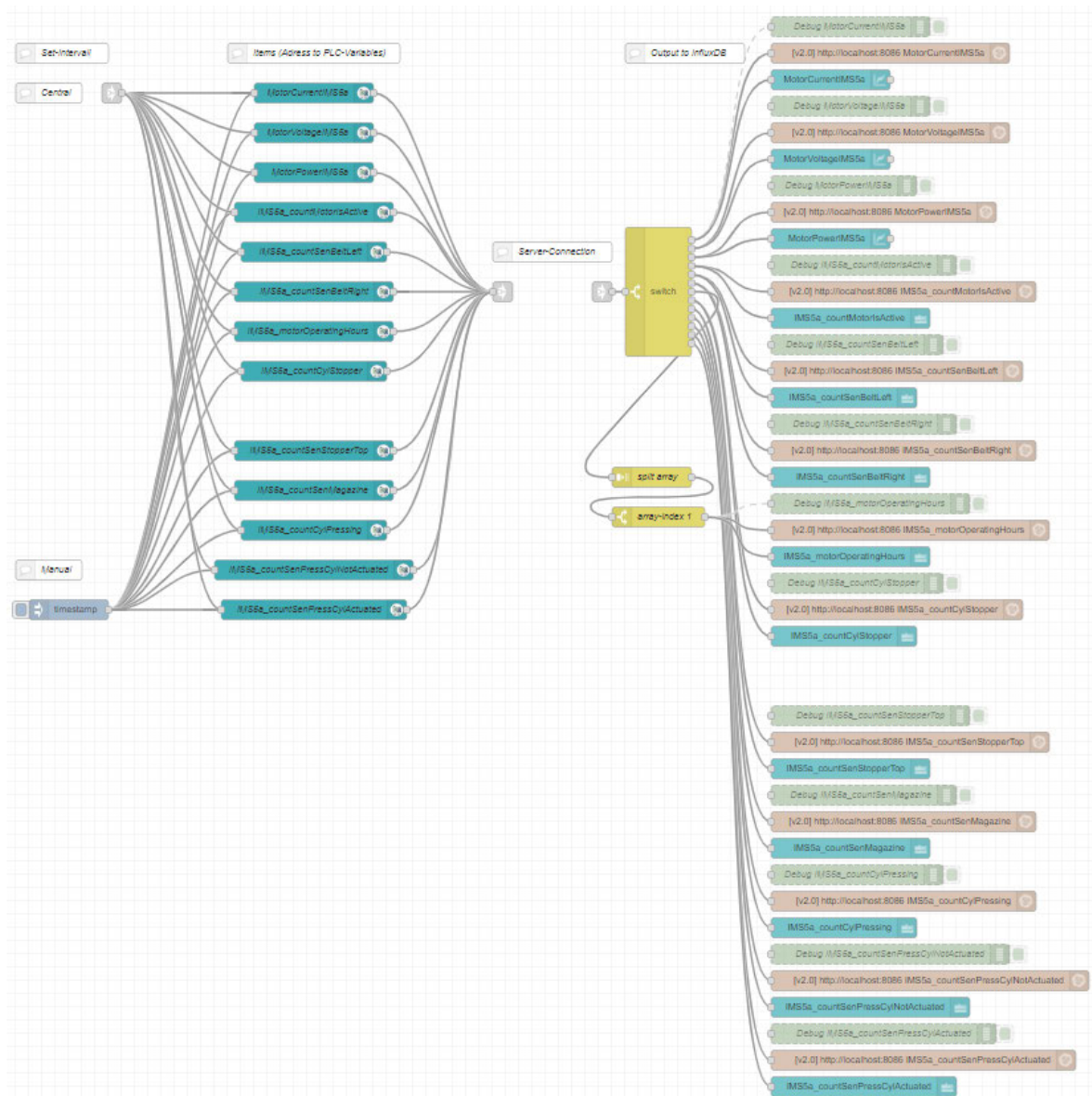


Figure A.6: IMS Station 5a

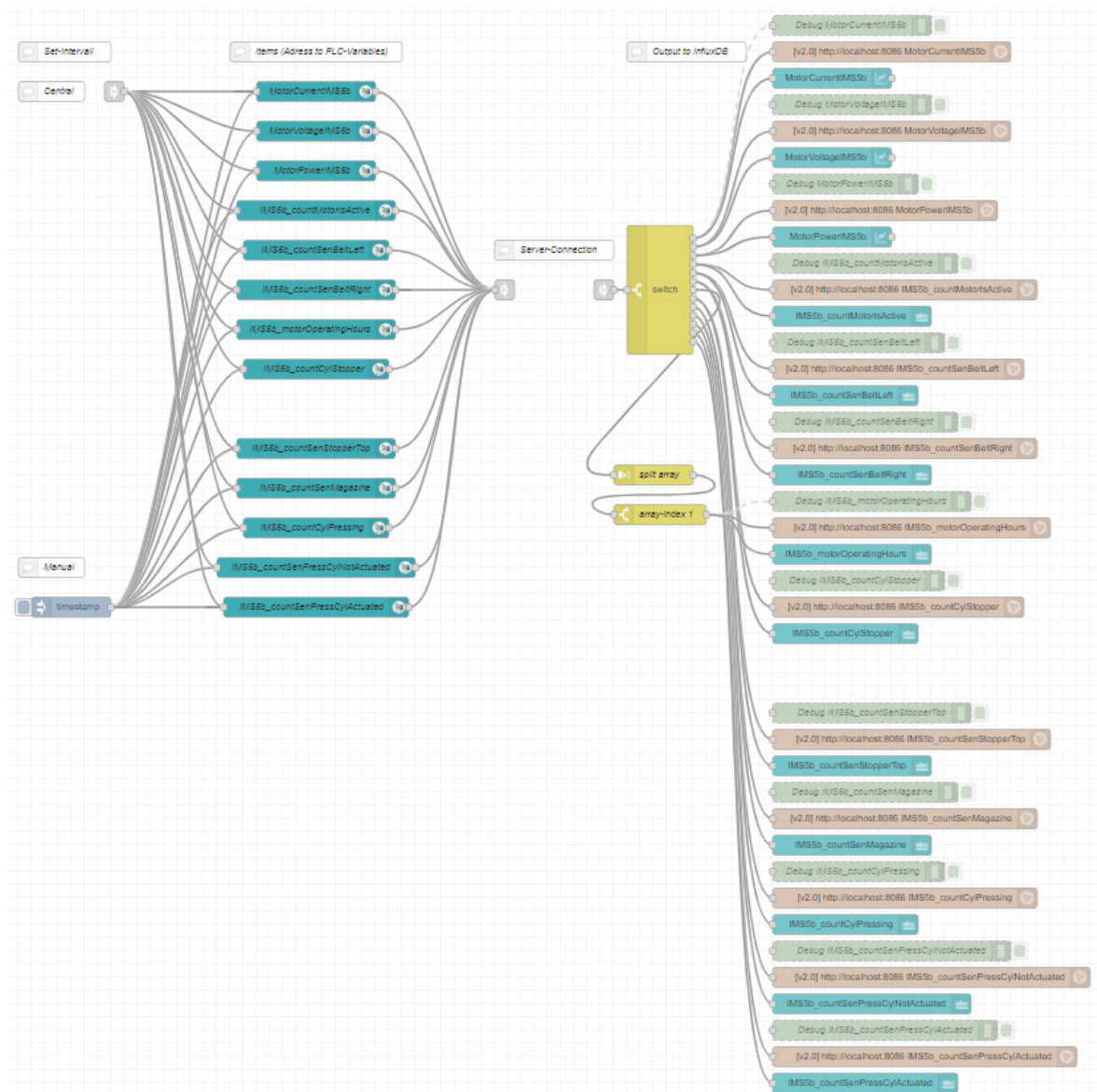


Figure A.7: IMS Station 5b

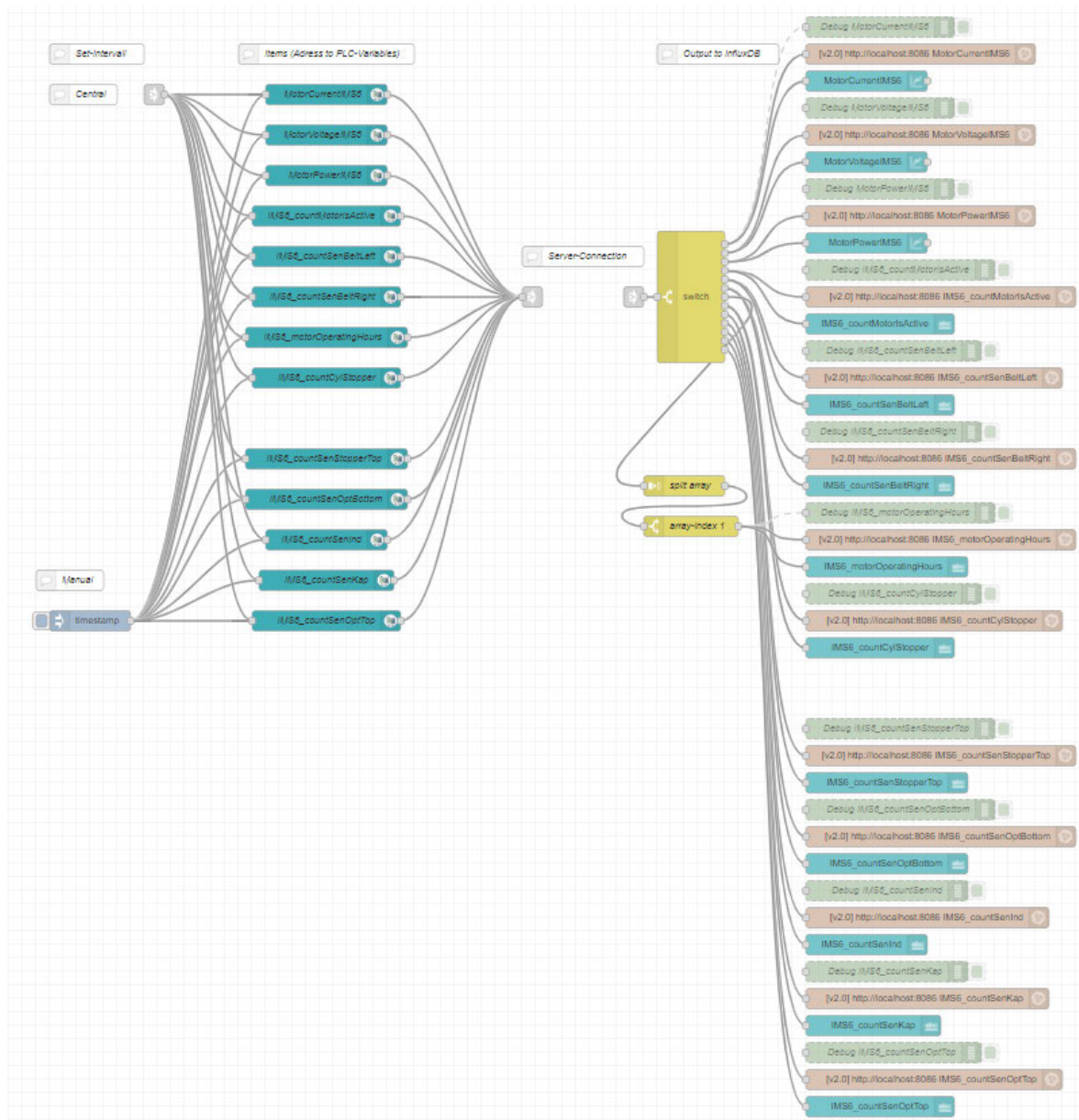


Figure A.8: IMS Station 6

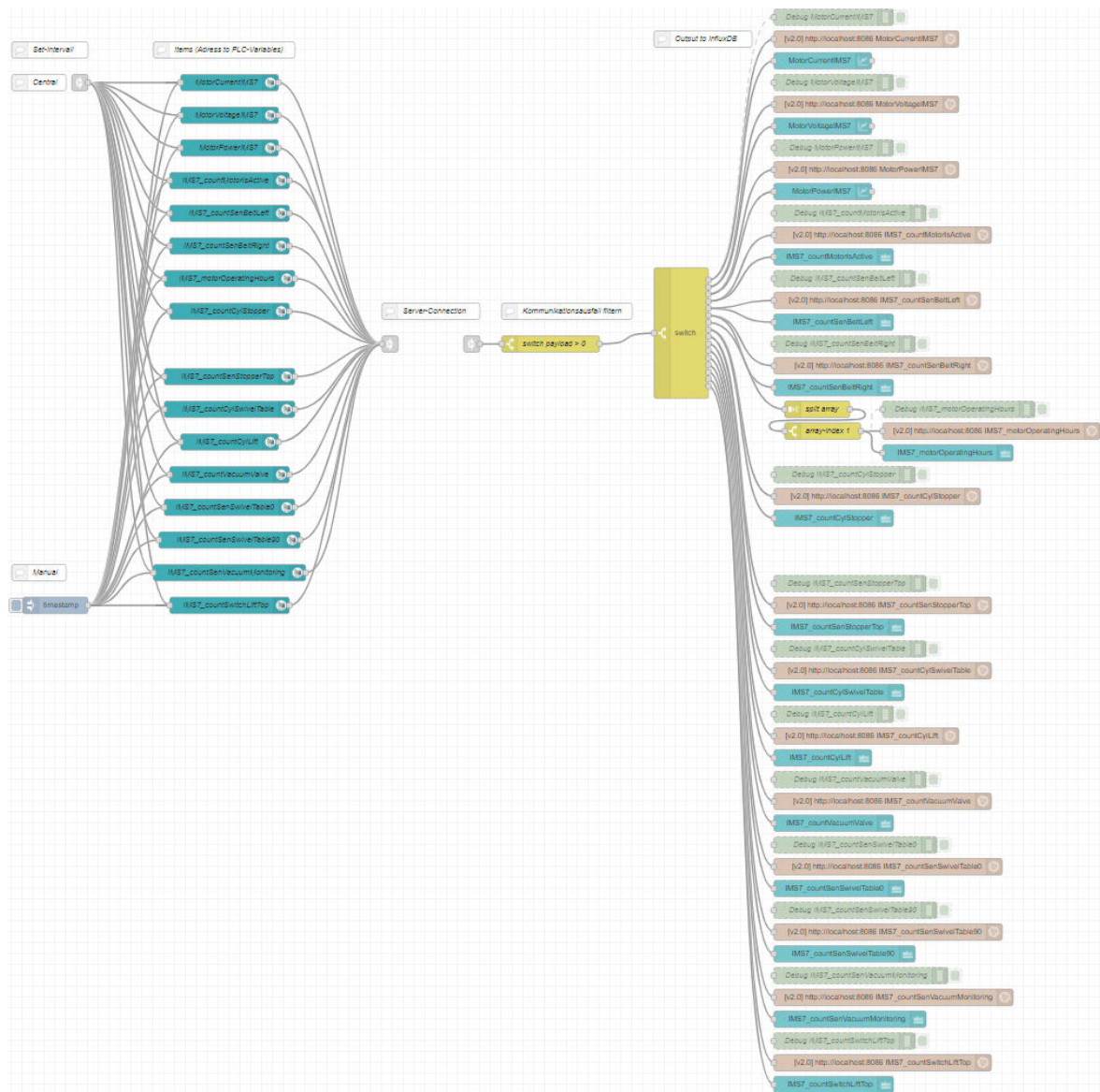


Figure A.9: IMS Station 7

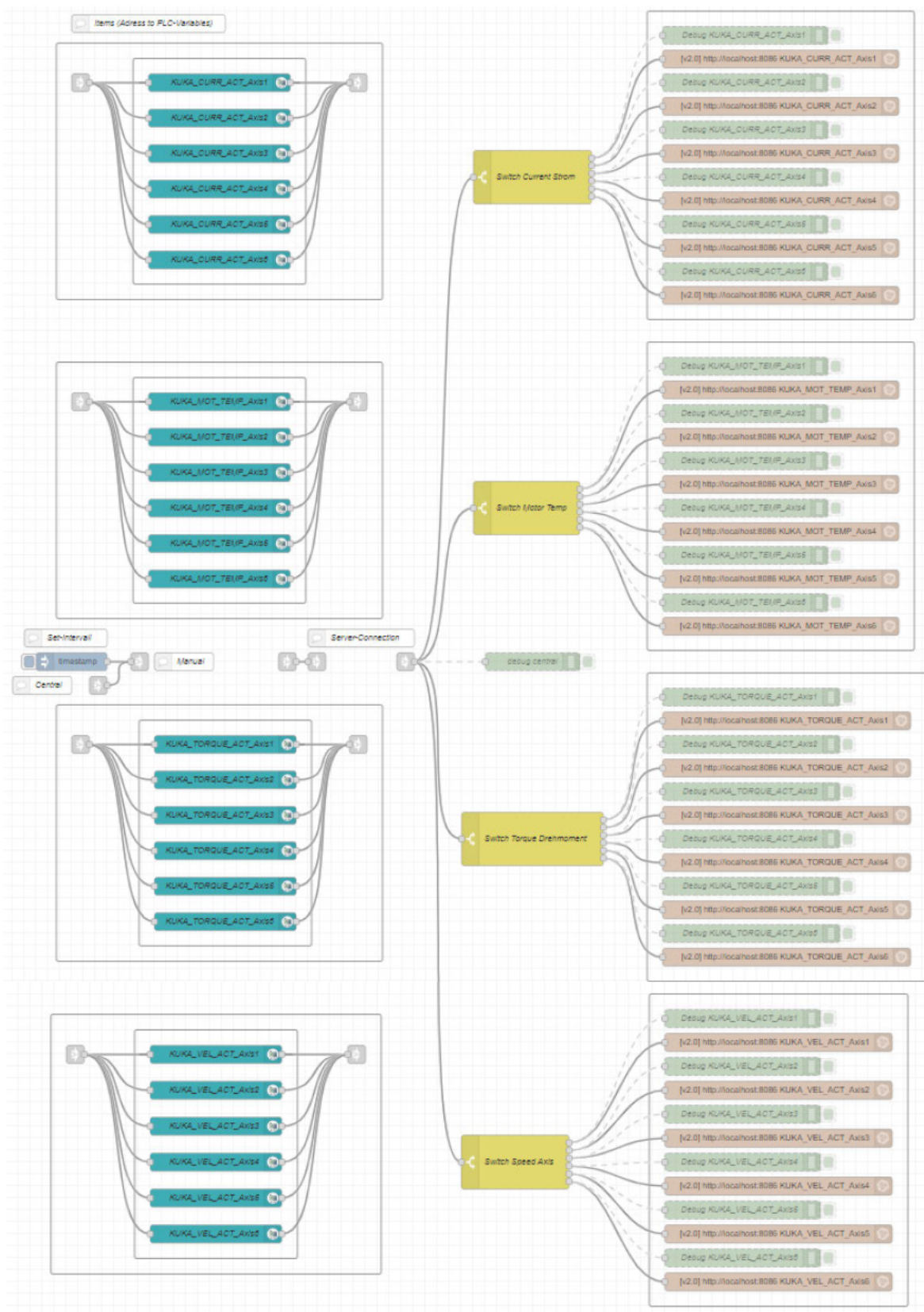


Figure A.10: KUKA robot

A.2 Flow of condition monitoring

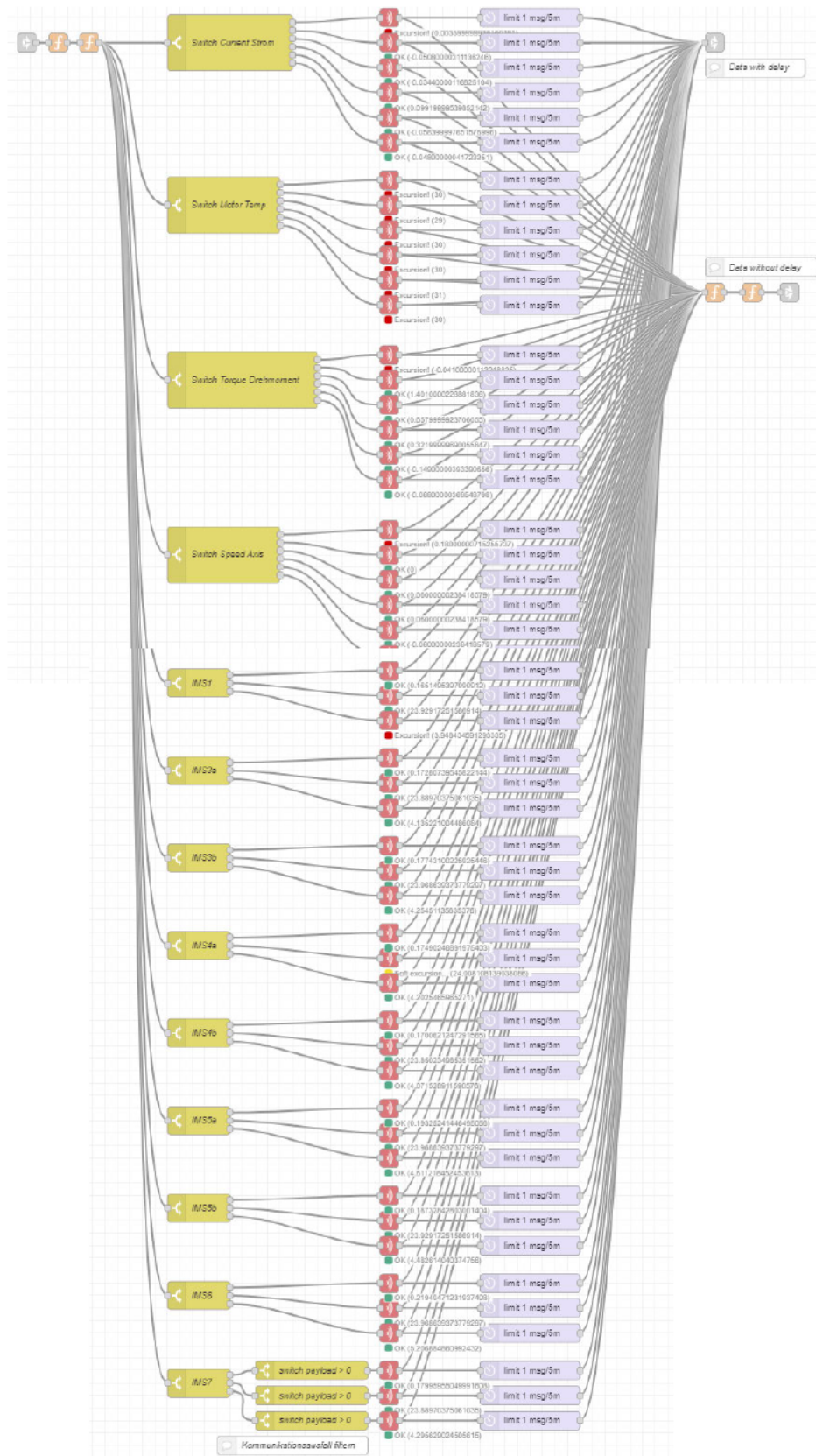


Figure A.11: Condition monitoring flow

A.3 Dashboards

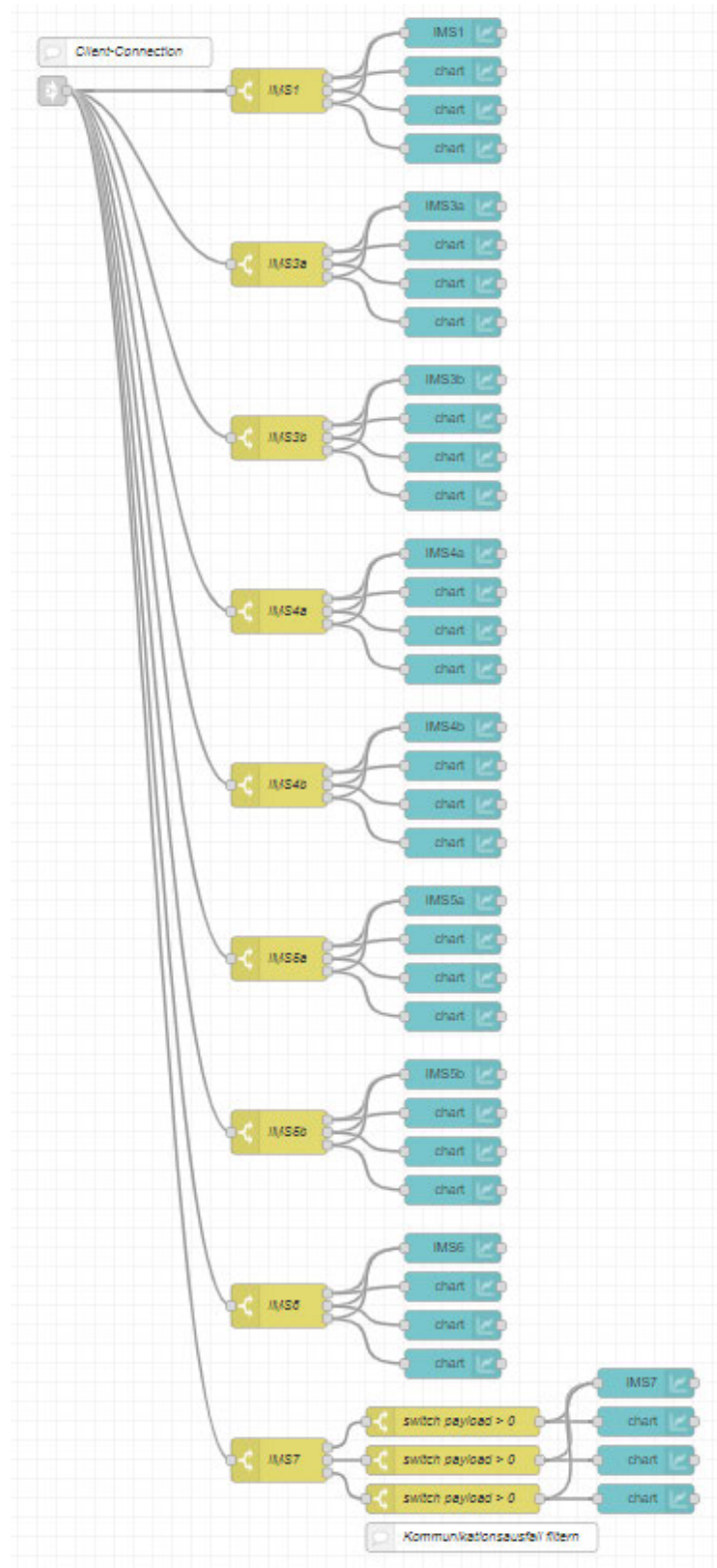


Figure A.12: IMS-Dashboard flow

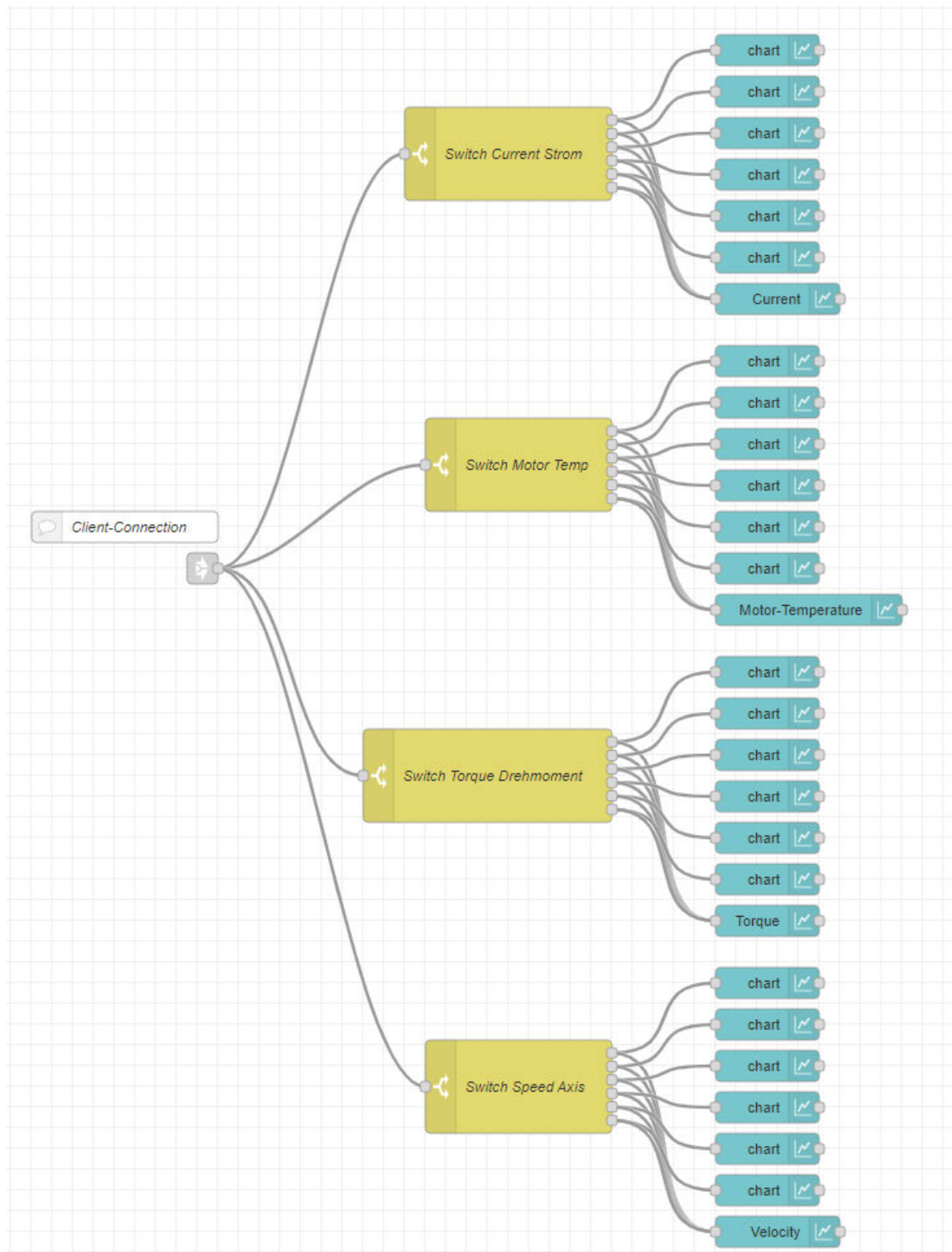


Figure A.13: KUKA-Dashboard flow

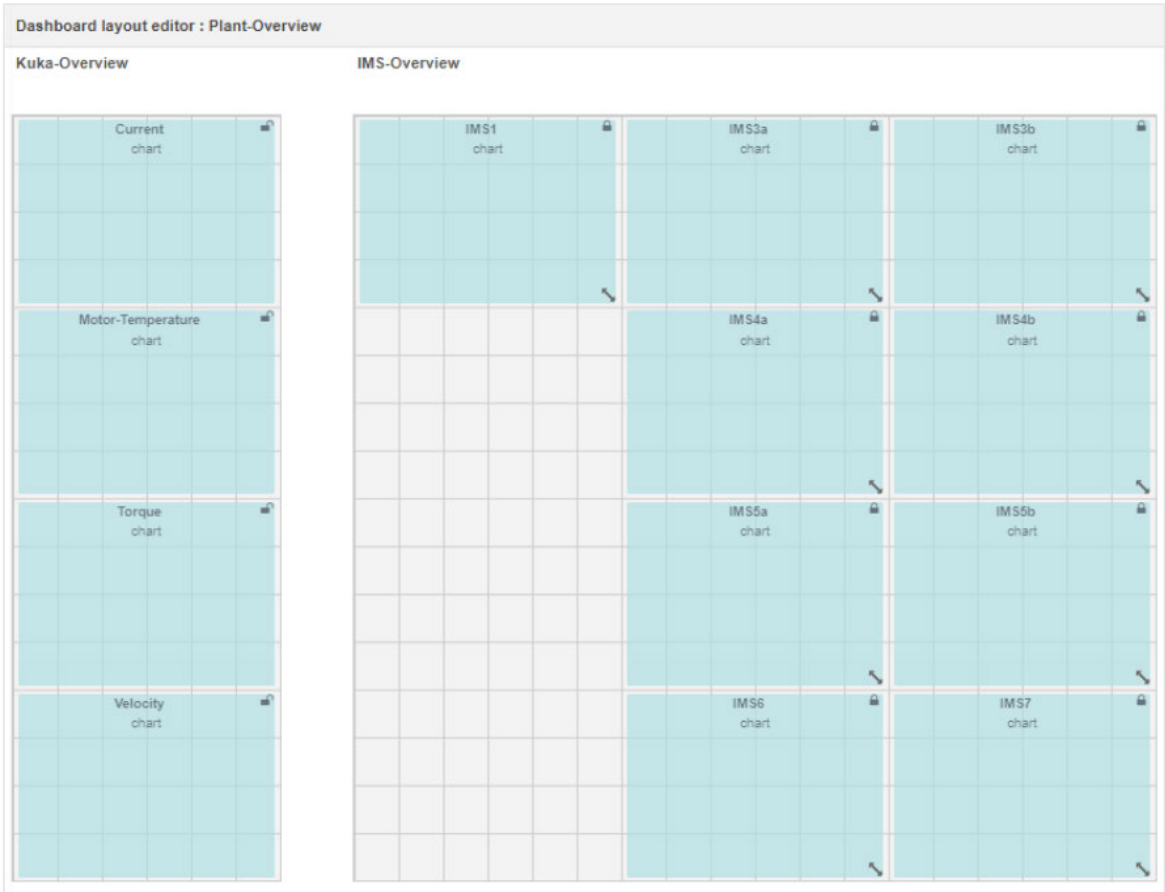




Figure A.14: Plant-Overview layout

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Shanghai, den 04.08.2024

Kenan Deniz