

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Dominik Löffler

Development of Predictive Maintenance Concepts
for a Networked Production Plant

Dominik Löffler

Development of Predictive Maintenance Concepts
for a Networked 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 10. August 2019

Dominik Löffler

Thema der Masterthesis

Entwicklung von prädiktiven Instandhaltungskonzepten für eine vernetzte Produktionsanlage

Stichworte

Industrie 4.0, Prädiktive Wartung, Internet der Dinge, Cyber-Physische Systeme, Intelligente Daten, Cloud

Kurzzusammenfassung

Diese Arbeit beschreibt die Erweiterung einer Industry 4.0 Produktionsanlage um prädiktive Instandhaltungskonzepte. Im Mittelpunkt steht die Vernetzung der physischen Daten mit der dienstbasierten Cloud-Plattform Machine Advisor, die eine kontinuierliche Überwachung der Systemleistung ermöglicht. Der Betreiber der Anlage wird informiert, sobald die Daten auf mögliche Fehlerzustände hinweisen. Darüber hinaus wird MATLAB für eine detailliertere Analyse der gesammelten Informationen verwendet.

Dominik Löffler

Title of the paper

Development of Predictive Maintenance Concepts for a Networked Production Plant

Keywords

Industry 4.0, Predictive Maintenance, Internet of Things, Cyber-Physical Systems, Smart Data, Cloud

Abstract

This paper describes the extension of an Industry 4.0 production plant by predictive maintenance concepts. The focus is on networking the physical data with the service-based cloud platform Machine Advisor, enabling continuous monitoring of system performance. The operator of the plant is informed as soon as the data indicates potential error conditions. Furthermore, MATLAB is used for a more detailed analysis of the collected information.

Acknowledgments

Above all, I want to express my sincere gratitude to Prof. Dr.-Ing. Florian Wenck and Associate Prof. Dr.-Ing. Shen JianQiang for their friendly and competent advice and the opportunity to realize this project in Shanghai.

Furthermore, thanks to the Hamburg University of Applied Sciences and the Shanghai-Hamburg College, making the whole stay possible.

I also would like to thank the company Lucas-Nuelle for providing the production plant at SHC and their great support.

For the great collaboration, I would like to thank all employees at SHC. A special thanks to Wu Di for translating and explaining Chinese documentation.

An exceptional thanks go to my friends and family for their great support not only during this thesis but during my studies.

On a final note, thanks to all those from Shanghai, especially Kong Qiangqiang, who supported me the last months and made me feel homelike.

Contents

List of Figures	8
List of Tables	10
Listings	11
1 Introduction	14
1.1 Motivation	14
1.2 Research Goals	15
1.3 Structure of the Thesis	15
2 Theoretical Foundations	16
2.1 Industry 4.0	16
2.1.1 Cyber-Physical Systems	16
2.1.2 Internet of Things	18
2.2 Predictive Maintenance	19
2.3 Programmable Logic Controller	20
2.3.1 Definition of a Control System	20
2.3.2 Functioning of a PLC	21
2.3.3 The PLC in Terms of Industry 4.0	22
2.4 Open Platform Communications Unified Architecture	23
2.4.1 Generic OPC UA Information Models	24
2.4.2 Address Space Model	24
2.5 Profinet	25
2.5.1 Profinet IO Real-Time Behavior	25
2.5.2 Profinet IO-System	25
2.6 Modbus/TCP-Communication	27
2.7 Machine Learning	27
2.7.1 Classification	28
2.7.2 Neural Network	29
2.8 Development Tools	30
2.8.1 Totally Integrated Automation Portal	30
2.8.2 Siemens OPC UA Modeling Editor	30

2.8.3	KUKA.WorkVisual	31
2.8.4	Machine Advisor	31
2.8.5	ComXBox	32
2.8.6	MATLAB	33
3	Industry 4.0 Production Line	34
3.1	Technical Description	35
3.1.1	Process Specification	35
3.1.2	Components	37
3.2	Application of Industry 4.0	46
4	Requirements Analysis	47
4.1	Functional Requirements	47
4.2	Comparison between Machine Advisor and MATLAB	48
5	Hardware Configuration	50
5.1	Plant Networking	50
5.2	Communication between Controllers	51
5.3	Data Exchange with KUKA Robot	53
5.3.1	Selection of Robot Transfer Data	54
5.3.2	Configuration in KUKA.WorkVisual	55
5.3.3	Configuration in TIA Portal	58
5.4	Connection to Machine Advisor	60
5.4.1	Setup of Modbus TCP/IP Server	60
5.4.2	Configuration of ComX Box and Machine Advisor	63
5.5	Establishment of OPC UA Communication	67
5.5.1	Activation of OPC UA Server	67
5.5.2	Creation of an OPC UA Client	68
6	Program Design and Implementation	70
6.1	Data Collection	70
6.1.1	Selection of IMS Stations Data	70
6.1.2	Introduction of User-Defined Data Types	72
6.1.3	Program Extension of IMS Stations	73
6.1.4	Program Extension of KUKA Robot	76
6.1.5	Program Extension of PLC_1	77
6.2	Data Analysis in Machine Advisor	81
6.2.1	Interpretation of Data	81
6.2.2	Alarm Management	82
6.3	Data Analysis in MATLAB	83
6.3.1	Creation of a server interface with SiOME	83

6.3.2	Querying Data	84
6.3.3	Pattern Recognition Network	86
7	Functional Test	89
7.1	Data Monitoring in Machine Advisor	89
7.2	Evaluation in MATLAB	94
8	Conclusion	99
8.1	Summary	99
8.2	Outlook	100
	Bibliography	101
A	Appendix	105
A.1	Parameter Template: USST_industrylab4_0	106
A.2	Data Type: DATA_TO_MASTER	110
A.3	Data Type: KUKADATA_TO_MASTER	111
A.4	Data Block: ModbusServerData	112

List of Figures

2.1	Decomposition of the automation hierarchy	17
2.2	Smart Metering – Measurement of physical data	18
2.3	Schematic structure of a control circuit	21
2.4	OPC UA layer model	23
2.5	Profinet IO system	26
2.6	Modbus frame encapsulation in TCP segments	27
2.7	Neural network for classification	29
2.8	SiOME operating principle	31
2.9	Machine Advisor	32
2.10	ComXBox	33
3.1	Industry 4.0 production line	34
3.2	Workpieces	35
3.3	Plant schematics	36
3.4	IMS1: Conveyor belt with PLC	37
3.5	IMS3x: Assembly station of the bottom part	38
3.6	IMS4x: Assembly station of the top part	39
3.7	IMS5x: Assembly station of the bolt	40
3.8	IMS6: Testing station	41
3.9	IMS7: Handling station	42
3.10	KUKA robot	43
3.11	Structure of the ERP-Lab	44
3.12	SCADA tab of the ERP-Lab	46
5.1	Plant network	50
5.2	Operating mode of I-Device	51
5.3	Configuration of the transfer area	52
5.4	I-Device communication	53
5.5	Axes of the robot	54
5.6	KUKA Profinet configuration	56
5.7	IO mapping editor	56
5.8	Mapping fieldbus IO signals	57
5.9	KUKA IO-Device in TIA Portal	59

5.10 TIA Portal network view	59
5.11 Communication between Modbus TCP/IP server and client	60
5.12 <i>MB_Server</i> function block	61
5.13 Data block attributes for Modbus server data	61
5.14 Connection parameters for Modbus communication	62
5.15 Parameter template	64
5.16 ComX Box Modbus TCP/IP configuration	64
5.17 Operating mode of the ComX Box	65
5.18 Definition of the gateway	65
5.19 Upload settings	66
5.20 Configured ComX Box gateway	66
5.21 Activation of the OPC UA Server	67
5.22 OPC UA subscriptions	68
5.23 Selection of the runtime license	68
6.1 User-defined data type in the PLC tags	72
6.2 Data collection function block	74
6.3 Function block <i>FB_CONTROLLERDATA</i>	78
6.4 Data block structure in TIA Portal	80
6.5 Alarm settings	82
6.6 Data and alarm management view	83
6.7 Information model for the plant data	84
7.1 Data monitoring view	89
7.2 Historical data view	90
7.3 Dashboard for IMS1	91
7.4 Low-level Error occurrence	91
7.5 Creation of an alarm ticket	92
7.6 Notification of an open alarm ticket	92
7.7 Acknowledgment of an alarm	93
7.8 Notification of a high-level alarm	93
7.9 Regular samples of axis one	94
7.10 Error samples of axis one	95
7.11 Network structure	96
7.12 Performance	96
7.13 Confusion matrix	97
7.14 Influence of the error of axis one	98

List of Tables

3.1	IMS1: Conveyor Belt with PLC	37
3.2	IMS3x: Assembly Station of the Bottom Part	39
3.3	IMS5x: Assembly Station of the Bolt	40
3.4	IMS6: Testing Station	41
3.5	IMS7: Handling Station	43
4.1	Functional requirements	47
4.2	Comparison between Machine Advisor and MATLAB	48
5.1	Addresses of I-Device communication	53
5.2	Configuration of robot variables	57
5.2	Configuration of robot variables	58
6.1	IMS stations data	70
6.1	IMS stations data	71
6.2	"BELT".IMS identifier	73
6.3	Monitoring types	81
7.1	Parameter comparison	95

Listings

5.1	Creation of an OPC UA client	69
6.1	Data collection function block	75
6.2	Signal declarations in the <i>config.dat</i>	76
6.3	Assignment of variables in <i>sps.sub</i>	76
6.4	Recording of the motor runtime	78
6.5	Conversion of robot data	79
6.6	Function <i>TransferControllerData</i>	80
6.7	Reading the OPC UA node array	84
6.8	Reading the OPC UA node array	85
6.9	Output of the <i>analyzeData</i> function	86
6.10	Parameter for the neural network	87
6.11	Creation of the pattern recognition network	87

List of Abbreviations

CPS	Cyber-Physical System
PLC	Programmable Logic Controller
IMS	Industrial Mechatronic System
RFID	Radio-Frequency Identification
HMI	Human Machine Interface
MES	Manufacturing Execution System
ERP	Enterprise Resource Planning
SCADA	Supervisory Control and Data Acquisition
IoT	Internet of Things
HTML	Hypertext Markup Language
CPU	Central Processing Unit
TCP/IP	Transmission Control Protocol/Internet Protocol
OPC	Open Platform Communications
UA	Unified Architecture
TSN	Time-Sensitive Networking
SOA	Service-Oriented Architecture
DA	Data Access
AC	Alarms and Conditions
HA	Historical Access
IO	Input and Output
CBA	Component Based Automation
RT	Real-Time

IRT	Isochronous Real-Time
MBAP	Modbus Application Header
PDU	Protocol Data Unit
TIA	Totally Integrated Automation
GSD	General Station Description
SiOME	Siemens OPC UA Modeling Editor
FTP	File Transfer Protocol
PC	Personal Computer
I	Input
Q	Output
INT	Integer
DWORD	Double Word
SIM	Subscriber Identity Module
SCL	Structured Control Language
LSB	Least Significant Byte
MSB	Most Significant Byte
XML	Extensible Markup Language
SMS	Short Message Service

1 Introduction

1.1 Motivation

For several years, the industry has been undergoing a significant evolution known as Industry 4.0. While the meaning of this term as a high-tech strategy was initially unclear, it is now crystallizing more and more. In order for such a great change in industrial processes to take place and for new technologies to be used in an economically beneficial and high-quality way, the industry has to agree on the content of the industrial revolution [1].

An essential characteristic is the digitally networked realization of industrial processes from product design through production design to the operation of the entire production plant. In a smart factory, the combination of software components with mechanical and electronic parts is called cyber-physical system (CPS). To achieve Industry 4.0, these CPS have to fulfill special requirements concerning architectural models, communication and data continuity, data processing for humans and intelligent products and production units [2].

Another product that emerges from the fourth industrial revolution is data. Already referred to as the fourth production factor, the variety of data produced by a smart factory is becoming increasingly important. Established data management systems such as database systems are often no longer sufficient enough and have to be extended by appropriate software transforming a large amount of heterogeneous data into smart data [3]. An application scenario for data analysis is predictive maintenance. Any production plant aims to ensure the availability, performance, and reliability of each component. Avoiding expensive downtimes and keeping maintenance costs as low as possible, thus maintenance is optimally scheduled during planned downtimes. Only elements that could impair the operation of the plant are replaced. While the reason for the sudden failure of a plant during production operation is quite obvious once the fault has occurred, predicting when it will happen is much more difficult. The analysis of the smart data at runtime can provide information on upcoming failures and their type as well as on the remaining runtime.

1.2 Research Goals

The most important and usually the most challenging step of the improvement of maintenance processes is the correct identification and classification of the maintenance demand. As an essential component of the fourth industrial revolution, predictive maintenance concepts benefit in particular from the data consistency of complex production systems. Data is now available at all communication levels and can be evaluated profitably using suitable software.

The goal of this thesis is the development of predictive maintenance concepts for a networked production plant. The production plant is an Industry 4.0 plant designed for training purposes by Lucas-Nuelle. It is located at the University of Shanghai for Science and Technology at the Shanghai-Hamburg College. The collected data ought to be loaded into the Machine Advisor of Schneider Electric. The Machine Advisor is a cloud-based service platform allowing the plant operator to view machine data from anywhere around the world. Furthermore, MATLAB is used for a more detailed analysis of the data.

1.3 Structure of the Thesis

Chapter 2 informs the reader about the foundations of Industry 4.0 and predictive maintenance as well as programmable logic controllers. A special attention is attached to communication protocols and shallow neural networks. Furthermore, all development tools used are explained.

After the theoretical foundations, the Industry 4.0 production line is explained in detail in **Chapter 3**.

Functional requirements to Machine Advisor and MATLAB are expounded in **Chapter 4** where they are also being compared.

The network structure of the plant and the required configurations for data exchange as well as the connection to Machine Advisor and MATLAB are described in **Chapter 5**.

Chapter 6 conduces to the program design and implementation. The data collection is clarified, followed by the data analysis in Machine Advisor and MATLAB.

The functional test of the plant is part of **Chapter 7**. Being displayed and supervised in Machine Advisor, the data is also evaluated in MATLAB.

The final **Chapter 8** implicates a summary of the results as well as an outlook to future projects.

2 Theoretical Foundations

For a better understanding of the thesis, this chapter discusses the theoretical foundations. This includes information about Industry 4.0, predictive maintenance, programmable logic controller, neural networks, and the used communication protocols. Finally, the development tools are briefly described.

2.1 Industry 4.0

As an essential aspect of the high-tech strategy Industry 4.0, the further development of production and value creation systems has become evident [4]. This is particularly possible due to the progressive linking of the real and the digital world. As a result of this high-level linkage and the resulting ubiquitous availability of data, new perspectives are emerging for the near future of automation. The aim is to achieve more economical and efficient production through adaptive, self-configuring, and partly self-organizing flexible systems [5].

Vertical integration in production is not entirely new but is regaining importance as a result of horizontal integration in business processes and company networks along with the integration of consistent engineering processes [6]. Considering this almost unlimited networking of different locations, companies will be able to benefit from the experience and knowledge of various production sites.

Another keyword that has emerged from the Industry 4.0 initiative is the Smart Factory. It uses the Internet of Things (IoT) and consists of so-called CPS. Here, humans and machines, as well as other used resources, interact as naturally as in a social network [7].

2.1.1 Cyber-Physical Systems

CPS are the basis of Industry 4.0. Especially in implementation recommendations, they assume a central role [8]. The term CPS refers to intelligent machines or equipment, independently exchanging information, interacting and controlling each other. In [9], CPS are defined as embedded systems with the following characteristics.

- Direct acquisition of physical data using sensors and influencing physical processes by actuators,
- Evaluation and storage of data as well as active or reactive interaction with the physical and digital world based on this data,
- Connected to other CPS through digital networks,
- Use of data and services available worldwide,
- Application of several multimodal human-machine interfaces for more differentiated and dedicated communication and control.

In the conventional automation pyramid, the various levels are usually assigned specific tasks. With the introduction of CPS, data, services, and functions can be stored, accessed, and executed where it is most useful for flexible, effective development and production [5]. Therefore, it is no longer necessary to assign these CPS to a specific automation level. For example, sensor data can not only be accessed directly at the field level but can also be read using the services of a cloud. This trend leads to a step-by-step modularization of the existing automation pyramid (see Fig. 2.1).

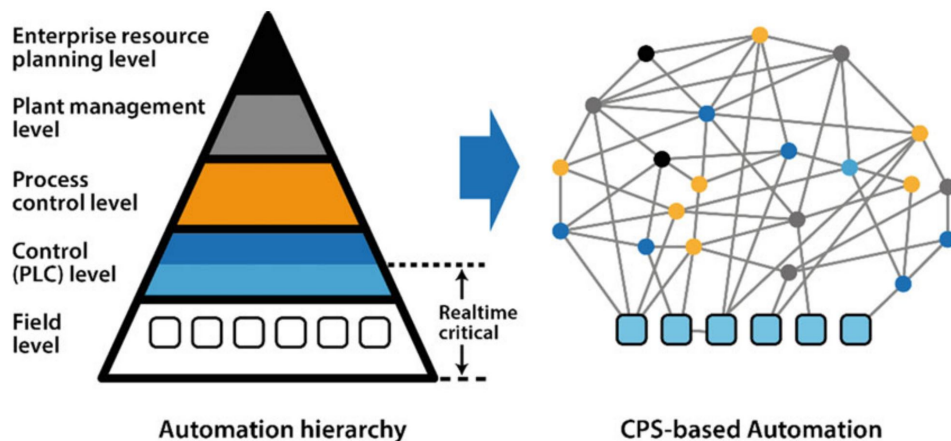


Figure 2.1: Decomposition of the automation hierarchy [10]

This creates a network in which all services, data, and hardware components are distributed in the nodes. The interaction of use and the provision of decentralized services create an automation cloud wherein the nodes abstract into functional modules. While real-time-critical controls remain at the field level close to the process, research is being done on communication protocols being able to transfer data to the cloud in real time. For this purpose, the open platform communications (OPC) foundation is investigating an extension for the OPC unified architecture (UA) in combination with time-sensitive networking (TSN) [11].

2.1.2 Internet of Things

The emergence of the IoT offers space for innovations, product functionalities, and efficiencies in value creation processes [1]. New networking structures are necessary, in order for machines to be able to communicate with other devices, for people to interact with the machine, and for resources such as intelligent workpieces to influence their processes. Furthermore, the networking of suppliers, other plants, marketing, and sales, as well as the customer, ought to be expanded. With conventional IT systems, this can only be achieved to a limited level [12]. In the future, system architectures have to be able to connect different networks in a service-oriented way.

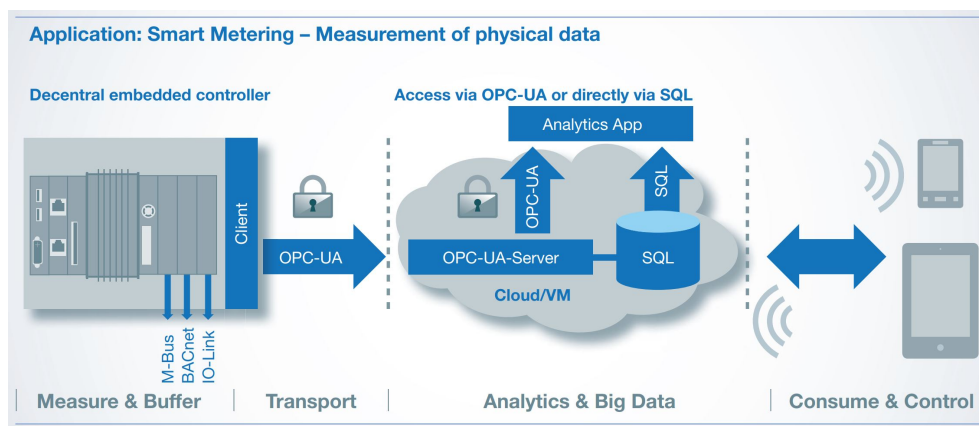


Figure 2.2: Smart Metering – Measurement of physical data [13]

During the development and implementation of applications for the IoT, the large amounts of data pose a challenge [6]. Especially at the field level, enormous amounts of data are generated in real time. Fig. 2.2 shows an application of the IoT implemented by Beckhoff Automation. In this example, the controller can communicate directly with the cloud via OPC UA connection, and the data is available for other devices for consumption or control purposes. Also, an analysis app whose evaluation has a positive effect on the efficiency of the system is used. In the IoT context, the term digital twin has become established for a virtual representation of a real plant or service [3]. The digital twin combines different types of information in order to represent an entire system as completely as possible.

One of the major tasks is to manage the data overflow. Moreover, the data can only be used profitably with appropriate analysis. Existing data repositories and architectures are going to reach their limits in terms of data amount and analysis speed.

2.2 Predictive Maintenance

Maintenance procedures can be divided into three different concepts: reactive, preventive, and predictive maintenance. Even though the latter concept is gaining importance due to the increasing spread of IoT and big data, the others will still exist in the future. For this reason, they are also briefly explained.

Reactive maintenance is the least complex concept. The system only reacts when an equipment failure occurs. However, this is only possible when the equipment can be replaced cost-effectively and does not cause major damage. A more sophisticated concept is preventive maintenance. As a result of the experience or recommendations of the system manufacturers, fixed maintenance intervals are planned. This can significantly reduce the risk of a breakdown, but the replacement of operational parts can increase costs, and no current plant-specific conditions are considered.

Industry 4.0 addresses this issue and improves current concepts with the possibilities of predictive maintenance. An important contribution to this is the expansion of existing machines and systems with appropriate sensors. Thus, machine data can be obtained in real time, and its evaluation contains information on the current operating status, the degree of wear, or any imminent defects. This makes it possible to predict errors and machine failures in advance and to initiate and plan countermeasures at an early stage [6]. Another advantage of this method is the continuous monitoring of all components. While regular maintenance can result in breakdowns before the maintenance interval is reached and high damage is caused, predictive maintenance reduces this risk [12]. In addition to the many advantages of predictive maintenance, there is also a disadvantage of this approach. Often the algorithms for data analysis reach a high complexity, and the implementation is associated with an immense engineering effort.

An essential stage in the implementation of predictive maintenance concepts is the selection of data. Besides sensor data such as temperature and vibrations, information like environmental conditions or machine operating time can also be used for evaluations. The methods can range from the calculation of simple condition indicators to the use of complex machine learning algorithms, allowing error patterns, trends, and correlations to be identified and a prognosis for the next maintenance time to be made. In general, the following three different approaches are used to detect possible abnormal behavior of a plant at an early stage:

Isolation of defective components

Data recorded at runtime can detect faulty behavior of individual components or provide indications of a trend towards an imminent failure. For example, vibration measurements can reveal the state of a component;

Isolation of causes of defect

Measured values from the system can also detect conditions, possibly leading to incorrect behavior. For example, the temperature monitoring of lubricants can increase the reliability of motors;

Isolation of environmental conditions

The last approach includes the change of environmental conditions for the prediction when and how faulty behavior may occur [14].

For efficient production, every company needs to maximize the uptime of its equipment, resulting in a lack of error occurrences in industrial data. Yet, this data would be worth knowing for optimal identification and prognosis algorithms. Past projects have shown that especially the interdisciplinary cooperation of engineers, product experts, and data analysis improves the value of the acquired information and the service quality [6].

2.3 Programmable Logic Controller

Control technology is one of the main areas of automation. In industry, programmable logic controllers (PLC) have become indispensable. As the basis for the advancing development of industry, controllers are becoming more and more powerful and taking on a wider range of tasks. While control systems were developed mostly proprietary for local and real-time critical applications, the manufacturers are now changing their strategy [15]. In the following, the definition of a controller and the principle of its operation are described. Last but not least, the necessary adaptations to controllers for Industry 4.0 are considered.

2.3.1 Definition of a Control System

In a control circuit, actuating values are set via defined regularities by measuring the process states [16]. A typical control circuit is shown in Fig. 2.3.

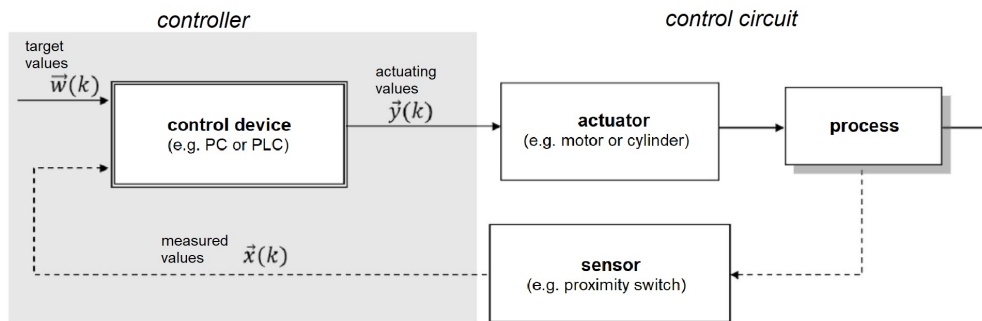


Figure 2.3: Schematic structure of a control circuit [16, adapted from Fig. 1.1]

The regularities are defined in the control algorithm allowing the controller to generate actuating values $\vec{y}(k)$ from the target values $\vec{w}(k)$. All values are written or read at the discrete sampling time k . The actuating values control the actuators of the plant affecting the process, which is called an open loop control circuit. Additionally, the state of the process is often monitored by sensors, and the information $\vec{x}(k)$ is transferred to the controller. Thus, this is known as a closed control loop.

2.3.2 Functioning of a PLC

In a PLC, the control task is set in the program as a result of instructions [17], being stored in the working memory of the PLC and holding their own address. During execution, the instructions are loaded into the central processing unit (CPU) according to their address. After processing, the next instruction is loaded by incrementing the address counter. This is called cyclic program processing.

A controller works pursuant to the input-processing-output (IPO) model [18]. Accordingly, a cycle essentially consists of three work steps. In the first step, the process image of the inputs is created, involving evaluating all input channels and writing the current values to the controller memory. During the second step, the program is processed sequentially, and at the same time, the process image of the outputs is created. Finally, the process image of the outputs is forwarded to the hardware.

Furthermore, cyclic program processing can be interrupted periodically or event-driven [19]. In this way, input and output (IO) signals can be processed acyclically or time-critical instructions can be executed.

2.3.3 The PLC in Terms of Industry 4.0

If conventional controllers need to be networked with each other, proprietary transmission control protocol/internet protocol (TCP/IP) or standardized fieldbuses such as Profinet or Modbus TCP will be commonly used in automation. These standard technologies usually distinguish significantly from the communication methods used by the Internet. As the fourth industrial revolution progresses, manufacturers of control systems have to enhance their products. The following points are mentioned as requirements for Industry 4.0 compatible controllers.

- Autonomy, reconfigurability and agility (Plug & Work),
- Overcoming the strict information encapsulation of controllers,
- Introduction of the service paradigm in production automation (production services),
- Networking in local and global networks,
- Interoperability between heterogeneous control systems,
- Dependencies have to be changeable dynamically during runtime,
- Use of models for the development of “higher-quality” control approaches,
- Orchestration of heterogeneous controllers [15].

In order to ensure the achievement of these requirements, manufacturers have to address two key issues emerging from state of the art.

First of all, basic web technologies need to be introduced. While newer models have already implemented web servers and hypertext markup language (HTML) pages for accessing process variables and configurations, further adaptation of the controllers is necessary. In most cases the connection via hypertext transfer protocol is relatively slow and therefore unsuitable for large amounts of data. Besides, most solutions are proprietary and open web interfaces are not available.

Secondly, the process data has to be accessible in the global or local network. For the integration of PLCs in supervisor, management, and coordination systems, the data transfer to the latter systems is essential. Often the communication does not take place from the controllers themselves but is taken over by human machine interfaces (HMI) which are located one level higher in the automation hierarchy. The resulting latency times are not always predictable but are usually slower than the process itself. Lean and standardized protocols such as MQTT and AMQP can enable communication directly from the controller. The OPC foundation has also recognized this and is developing a publish/subscribe-based specification to make the data available in a cloud [20].

2.4 Open Platform Communications Unified Architecture

OPC UA is a communication standard for the data exchange of mainly industrial components. By introducing such standard solutions, the integration of information systems can be realized more cost-effectively and time-saving in the long-term run [21]. The communication interface is independent of the manufacturer, the programming language or the operating system.

One of the essential innovations to the previous version is the extension of the information model. OPC UA continues to enable horizontal integration, allowing different automation devices to communicate with each other. Furthermore, the new specification can also be applied to vertical integration, including the connection between the various automation levels [22]. Accordingly, the new standard enables communication from the controllers at the field level to the associated servers at the production planning level. For this OPC UA uses a TCP-based, optimized, binary protocol [23].

As a communication basis, OPC UA follows the design paradigm of service-oriented architecture (SOA), meaning that a service provider receives requests, processes them, and returns the result in the form of a response [24]. For this task, the OPC UA specification defines different groups of services. Flexibility is achieved through the information model. Any complex extensions can optimize the basic model for the desired application without affecting the interoperability [23]. The OPC UA layer model is shown in Fig. 2.4.

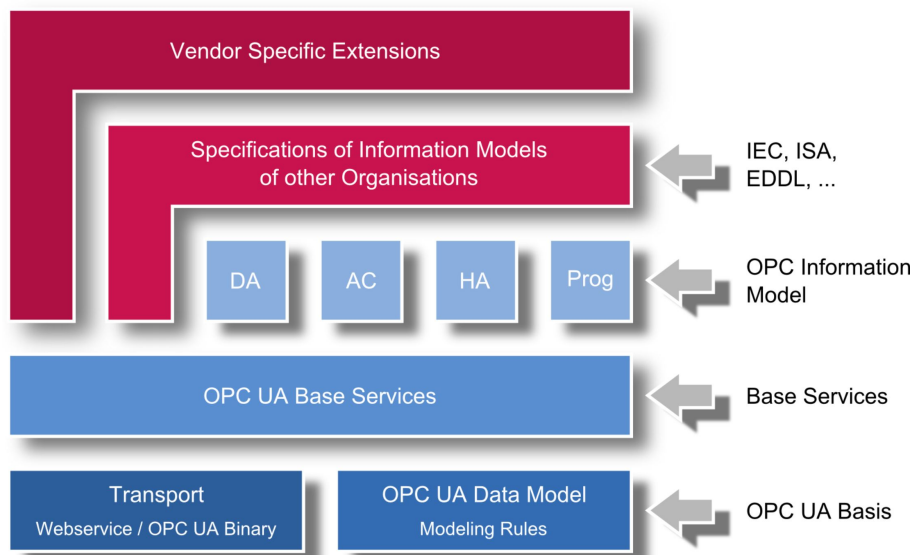


Figure 2.4: OPC UA layer model [25]

2.4.1 Generic OPC UA Information Models

According to the requirements of the industry, four standard information models have been developed. Their purpose is explained in the list below.

Data access (DA)

The OPC DA interface provides real-time access to process data, allowing to read, write and monitor variables. It is the most widely used interface of the standard models [26]. In this use case, a client establishes a connection to the server. The variables are usually updated after a fixed cycle or based on event occurrence;

Alarms and conditions (AC)

The AC model defines the handling of events and alarms. If a change of a condition causes an event, the clients having subscribed to this event will receive a notification. Also, accompanying values for corresponding states can be sent;

Historical access (HA)

Historical Access allows a client to access past variable values or events. The data is timestamped and stored in a database, archive, or other memory. A unique feature is the possibility to preprocess the information on the server [21];

Programs

In a program, access to functions on a server is described. The programs are structured as state machines whose end of processing or state transitions are transmitted to the client.

In addition, the standard models can be adapted to the models of other organizations or any other specifications. This is the only way to establish the basis for a standardized data exchange.

2.4.2 Address Space Model

The address space model is the number of objects that an OPC UA Server offers to the connected clients. Its purpose is to establish a standard method for the server to provide objects for the clients. The object information is mapped to nodes which can be simple variables or complex data types. References can represent relationships between multiple objects.

2.5 Profinet

Profinet is a communication protocol optimized for industrial applications. Based on Ethernet, it is a comprehensive standard fulfilling all requirements for the use in automation technology [27]. A distinction is made between Profinet IO and Profinet component-based automation (CBA). Latter is designed for the realization of modular applications as well as machine-to-machine communication and is not further considered.

Profinet IO is optimized for communication between a controller and decentralized field devices. It has established itself as one of the leading industrial Ethernet networks [28]. Optimized for the fast transmission of IO signals in real time, it also offers the possibility to send data via TCP/IP. All signals are read in, processed and retransmitted in each cycle without being requested by a communication partner.

2.5.1 Profinet IO Real-Time Behavior

Profinet IO distinguishes between real-time (RT) communication and isochronous real-time (IRT) communication. In the former case, RT capability is ensured by prioritizing the IO telegrams over the data telegrams, containing, for example, parameters or configurations. IRT communication is a synchronized transmission method for the cyclic exchange of data. For this purpose, a specific bandwidth is reserved in the transmission cycle, making the transmission of IRT data insensitive to a high network load and thus enabling clock-synchronous data exchange [29].

2.5.2 Profinet IO-System

Profinet IO does not only refer to the communication of individual controllers with their decentralized peripherals but can also be applied to entire systems. Fig. 2.5 shows the components of such a system.

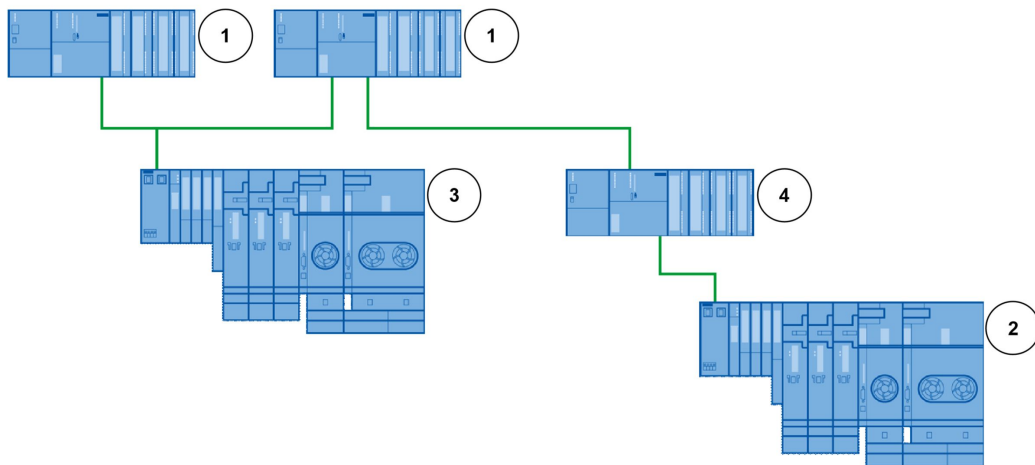


Figure 2.5: Profinet IO system [29]

According to their number, the elements from Fig. 2.5 are explained in the following list.

1. IO-Controller

The IO-Controller is a PLC executing an automation program and addresses connected IO-Devices. Therefore, the IO-Controller exchanges IO signals with the decentralized field device.

2. IO-Device

An IO-Device is a decentralized field module, usually assigned to an IO-Controller. For example, it can be a valve terminal, a distributed IO module or any other device with Profinet IO functionality. In most cases, the manufacturer provides so-called general station description (GSD) files allowing the user to utilize these IO-Devices in the configuration tool of the IO-Controller.

3. Shared device

If an IO-Device contains relevant data for several IO-Controllers, it can be declared as a shared device and therefore provides the data for more than one controller.

4. I-Device

Allowing any IO-Controller to be used as an IO-Device, the I-Device function enables the possibility of realizing subordinate Profinet IO networks. Simultaneously, the possibility to use the I-Device as a shared device is maintained [29].

2.6 Modbus/TCP-Communication

Modbus is a simple communication protocol for industrial control systems and is still widely used today [30]. Transmitting data via Modbus over TCP/IP, the Modbus slave acts as a TCP server and thus the Modbus master like a TCP client. The server waits for the client's request and returns its message in response. In this form of transmission, the Modbus packets are embedded in the TCP segments (see Fig. 2.6).

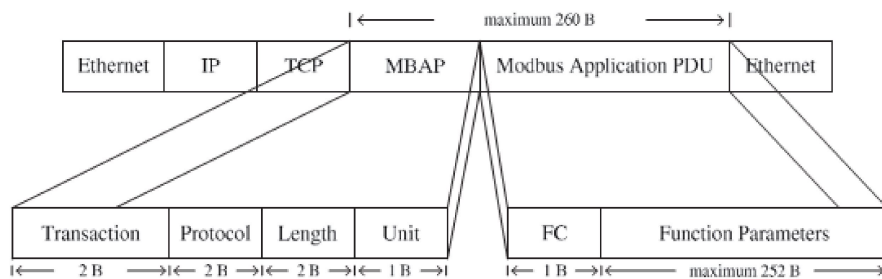


Figure 2.6: Modbus frame encapsulation in TCP segments [31]

The Modbus telegram consists of a seven-byte Modbus application header (MBAP) and the Modbus protocol data unit (PDU) with a size of up to 253 bytes. The MBAP contains a two-byte transaction identifier, a two-byte protocol identifier set to 0x0000 for Modbus, a two-byte length field, and a one-byte unit identifier. In the first byte of the PDU segment, a function parameter between one and 255 is set, whereby entries 128-255 are reserved for exceptions. During transmission, these parameters specify the type of action to be executed by the server. Besides, additional information for processing can be added to the message.

2.7 Machine Learning

As the amount of available data increases, the analytical methods also need to evolve. Machine learning approaches this challenge by enabling the automated analysis of data. In [32], machine learning is defined as a set of methods for recognizing patterns, allowing to predict future behavior or to make other decisions based on these revealed patterns.

In the industrial sector, time series patterns can indicate possible error states in a system. Potential failure causes can be identified with the automatic classification of these patterns. A conventional algorithm for this application is neural networks. In the following, both are explained in detail.

2.7.1 Classification

The supervised learning approach has the goal to find a mapping from given inputs x to outputs y . Therefore the training set

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N \quad (2.1)$$

contains labeled input and output pairs that are used for training. \mathcal{D} is called a training set and N is the number of available training examples.

The input values x_i may consist of more complex data such as images or time series, although in a simpler variant it may consist of vectors with numbers representing the attributes of the input signal. For instance, the attributes of a time series can be values such as mean value or variance.

The outputs y_i are part of a finite set

$$y_i \in \{1, \dots, C\}. \quad (2.2)$$

Assuming the set of possible values of y represents different categories in which the input values are assigned, this is referred to as classification or pattern recognition. In this case C corresponds to the number of classes.

In the case of $C = 2$, it is called a binary classification. If more than two classes are available, it will be called multi-class classification. Typically the input is assigned to exactly one class, otherwise it is called multi-label classification [32].

The formalization of the problem can be seen in its simplest form as function approximation. The equation

$$y = f(x) \quad (2.3)$$

contains the unknown function f which ought to be determined by training with the labeled datasets. Finally, the function has to be sufficiently approximated to correctly classify future unknown data.

2.7.2 Neural Network

Artificial neural networks are information processing systems whose function is modeled similar to a human brain. The structure resembles a nervous system consisting of a large number of parallel units, the neurons [33].

The communication flow proceeds via directional connections that are weighted. The actual learning process does not influence the active neurons but changes the adaptive weights of the connections. If a neuron receives a sufficiently high stimulation through these weighted connections, it will become active and send a signal to another neuron. Perceptrons are introduced to reproduce this behavior, summarizing the weighted inputs and forming their outputs by the activation function

$$y = \begin{cases} 1, & \text{falls } \sum_{i=1}^n w_i x_i \geq \theta \\ 0, & \text{sonst.} \end{cases} \quad (2.4)$$

whereby n corresponds to the number of inputs x_i with the weight w_i and y corresponds to the output. A threshold value θ is assigned to each perceptron.

One or more perceptrons form a layer in a neural network consisting of an input layer, one or more hidden layers, and the output layer. The most common is the feedforward structure, where each perceptron only has connections to the next higher layer. Fig. 2.7 shows a feedforward neural network for classification.

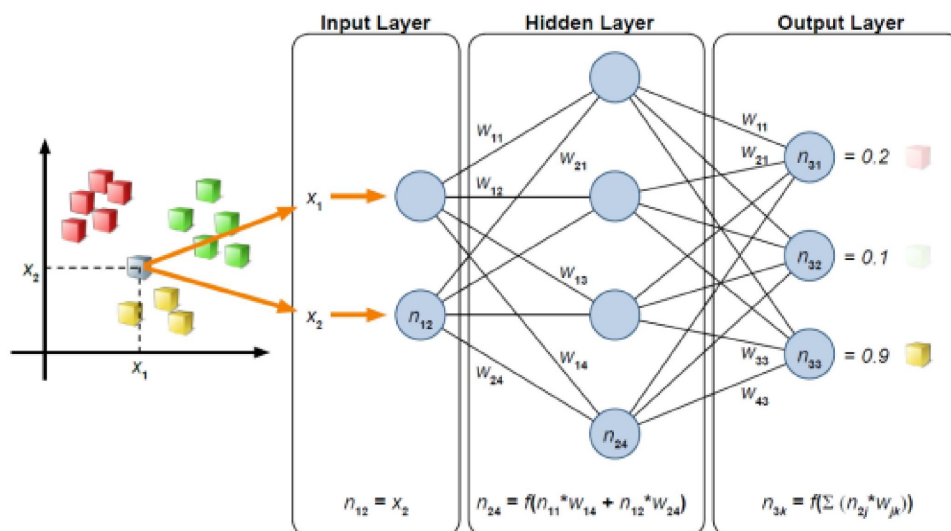


Figure 2.7: Neural network for classification [34]

By specifying the coordinates, an unknown object has to be assigned to one of the three classes. Using the input and output vectors of the known cubes, the weights of the connections are optimized to minimize the error between the output vector of the network, and the target output vector of the training data. The actualization of the weights follows the used training algorithm of the network.

2.8 Development Tools

In this section, the used development tools are briefly explained, including the Totally Integrated Automation Portal (TIA Portal) for programming the controllers, the Siemens OPC UA Modeling Editor (SiOME) for defining a custom information model, the Machine Advisor for data monitoring in the cloud, the ComX Box used for uploading the data to the Machine Advisor, KUKA.WorkVisual for the programming of the robot, and finally, Matlab for additional analysis of the collected data.

2.8.1 Totally Integrated Automation Portal

The TIA Portal allows the implementation of complex automation tasks from digital planning through integrated engineering to transparent operation, combining all engineering functions into a single framework [35].

For the realization of this thesis, software STEP 7 is used. This extension enables the planning, programming, and commissioning of projects with Siemens controllers. The programming languages are based on the open international standard IEC 61131-3. Especially due to its high integration capability and the support of industry standards such as OPC UA, Siemens has taken one of the market-leading positions in automation.

2.8.2 Siemens OPC UA Modeling Editor

The SiOME is used to define custom OPC UA information models. The information models are generated as extensible markup language (XML¹) files and can be imported directly into the TIA Portal. This allows variables to be mapped from the control program to the server interface, making them uniformly accessible to the clients. In Fig. 2.8, the operating principle of the SiOME is illustrated. In addition to the self-defined information models, standardized OPC UA models can be imported and adapted to the individual application.

¹XML is a markup language for the platform-independent exchange of data

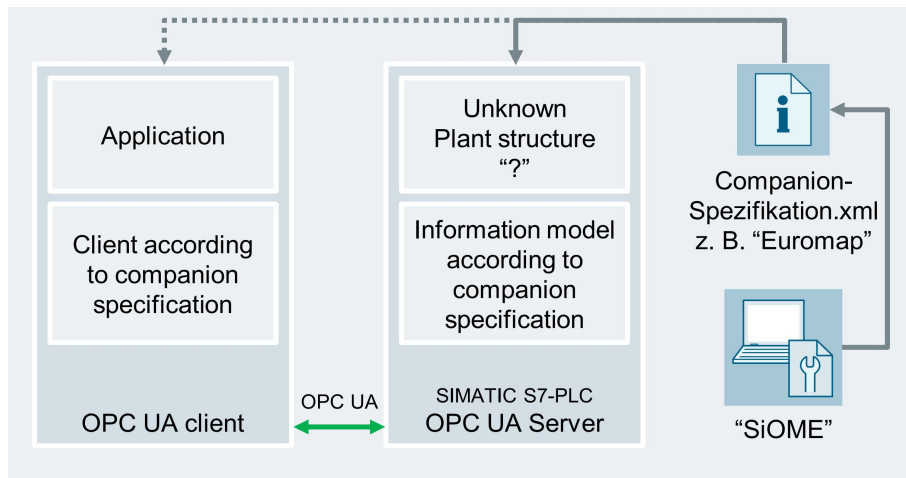


Figure 2.8: SiOME operating principle [36]

2.8.3 KUKA.WorkVisual

The KUKA.WorkVisual environment enables intuitive configuration, programming, commissioning, and diagnosis of robot systems. Besides offline programming, the software offers extensive online diagnostic options for safe operation. The development environment is optimized for the dedicated KRC4 controllers and enables simple connection to other control systems by supporting fieldbuses such as Profinet, Profibus, or EtherCAT.

2.8.4 Machine Advisor

The Machine Advisor, developed by Schneider Electric, offers a cloud-based service platform for machine manufacturers to monitor performance data. Multiple machines can be registered and located in the cloud, maintenance schedules can be created, machine documents can be uploaded, and tickets for the operator can be prepared. A significant advantage is the availability of the data as it is accessible at any time and any place. By creating dashboards, the data can be visually presented and, for example, compared with data from the previous year. A machine manufacturer can use this tool to monitor all his machines and observe trends at an early stage. The aim is to increase the availability of systems while reducing support costs. Fig. 2.9 shows the possible use of a dashboard in the Machine Advisor.



Figure 2.9: Machine Advisor [37]

2.8.5 ComXBox

The ComXBox represents an interface between the data collection devices and the cloud platform. It stores the data locally and loads it into the cloud at a minimum interval of fifteen minutes. The box can be configured via web interface allowing the user to parameterize communication settings. Modbus/TCP is used for collecting data from the controllers, and the file transfer protocol (FTP) is applied for uploading data to the cloud. Besides, alarms for the collected data can be defined. Fig. 2.10 shows the use of the ComXBox.

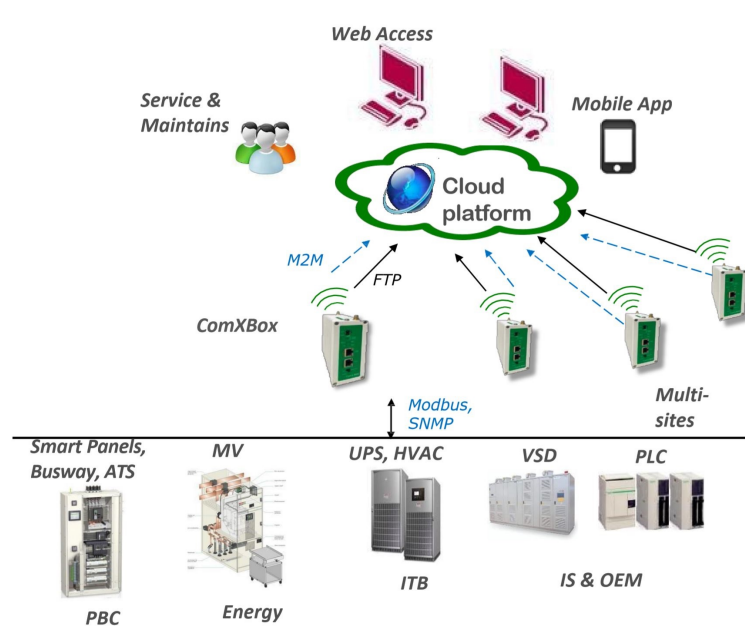


Figure 2.10: ComXBox [38]

2.8.6 MATLAB

The MATLAB desktop environment developed by MathWorks is based on a proprietary programming language. Combining numerical analysis, matrix calculation, signal processing, and visualization, the tool is designed to solve mathematical problems. By applying toolboxes, the functionality can be extended in various aspects.

One of the used toolboxes is the OPC Toolbox enabling access to OPC data directly from MATLAB. Thus, the data from a PLC can be read, written, and logged. As well as accessing data from live servers using OPC DA, it also supports OPC HA allowing historical data to be obtained.

Another toolbox used is the Deep Learning Toolbox allowing neural networks to be designed and implemented. Executable apps provide simple configuration of network architectures and training parameters as well as monitoring of the training progress in appropriate diagrams. Additionally, pre-trained models can be used to improve the results of small data sets.

3 Industry 4.0 Production Line

This chapter introduces the Industry 4.0 production line of Lucas-Nuelle which will be enhanced in this thesis (see Fig. 3.1). The plant belongs to the laboratory of Shanghai-Hamburg College and provides training of students.

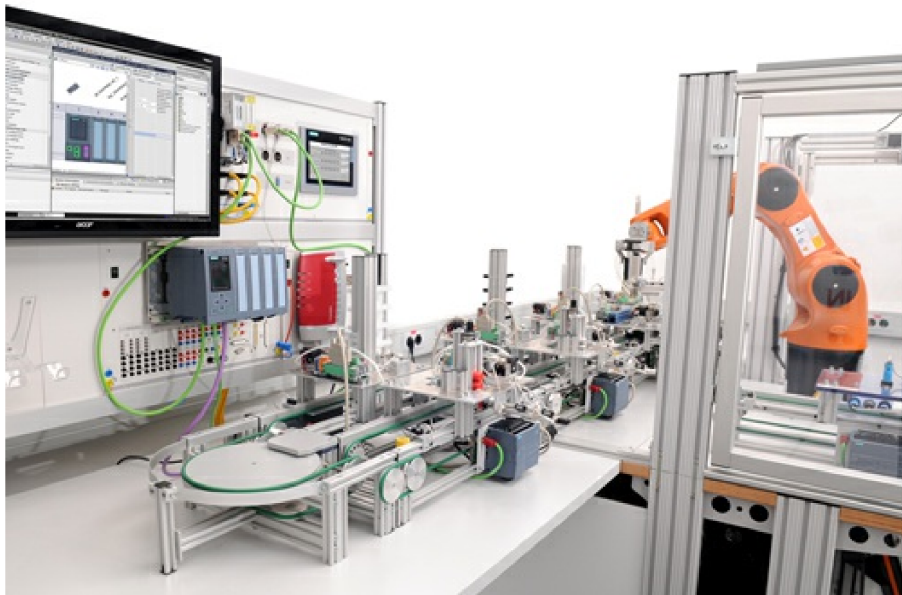


Figure 3.1: Industry 4.0 production line [39]

In the following sections, the production process and the associated components of the plant are explained. If all parts of the plant are known, reference will be made to the application of the aspects of Industry 4.0. Predictive maintenance is particularly highlighted as a single aspect here.

3.1 Technical Description

The Lucas-Nuelle production line offers highly flexible production and individualization of products up to batch size one. The system informs the user about the current status of production at any time. Before the associated components and their structure are explained in detail, the dynamic production process is described.

3.1.1 Process Specification

The manufacturing purpose of the plant is the production of workpieces. In this process, the bottom and top parts, shown in Fig. 3.2, are joined together and optionally pinned. According to the individual request of the customer, the products will be automatically assembled.



Figure 3.2: Workpieces [39]

The system has an OpenCart¹ webshop for the customer to place an order. The customer can intuitively place a request according to his purposes. After selecting a workpiece with or without bolt, a selection of the desired composition can be made. Generated in OpenCart, the orders are transmitted into the manufacturing execution system (MES) database and will be produced if the system is ready.

The production of the workpieces requires a total of eight stations. Furthermore, there is an extension for checking the workpieces, accessible with a robot manufactured by KUKA. Fig. 3.3 illustrates the structure of the plant.

¹Open source online store management program

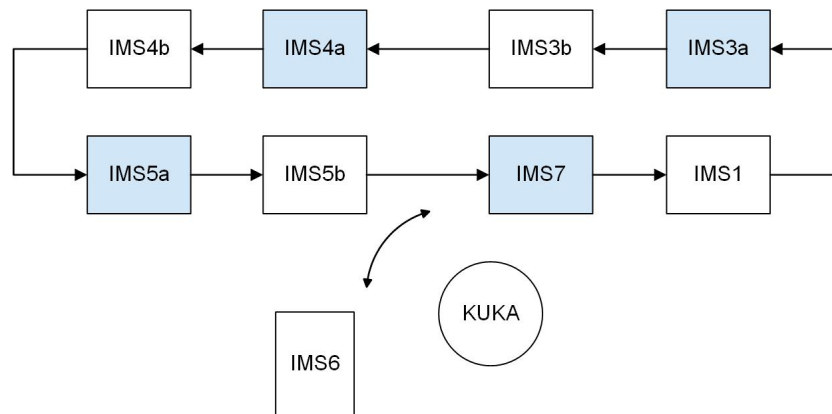


Figure 3.3: Plant schematics

When the system is in idle state, three workpiece carriers are integrated into the system. The waiting positions are behind industrial mechatronic system (IMS) 7 and IMS1 as well as in front of station IMS3a. With radio-frequency identification (RFID) read and write units, mounted on the stations marked in blue, each carrier is assigned a unique ID and the corresponding product information. As a result, the system does not only know the position of the carriers in the system but also the current production status of each order. Being referred to as an intelligent workpiece under Industry 4.0 criteria, the workpiece has self-knowledge. More precise, it knows the desired final state and the necessary production sequence.

As soon as the customer has entered an order, the system starts producing automatically. If there is a free carrier in the waiting position in front of IMS3a, the RFID unit can write the product information. Consequently, all stations to be approached are known. In the station IMS3a the black bottom parts are assembled, and the same applies to the white bottom parts in station IMS3b. In the following, the top parts are assembled in black (IMS4a) or white (IMS4b). Each workpiece has to consist of at least one bottom and one top part. Furthermore, the customer can decide whether both parts shall be fixed with a bolt. For this purpose, the IMS5a station contains plastic bolts, and the IMS5b station contains metal bolts. If no bolts are selected, the carrier will pass through both stations without action. Before the carrier reaches the handling station and the production process is finished, the operator decides whether the workpiece should be checked or not. For this, the KUKA robot can be selected by a HMI touch panel. During the checking process, the carrier stops behind IMS5b, and the robot grabs the workpiece for transport to the testing station. The testing station, equipped with several sensors, compares the actual properties of the workpiece with the properties desired by the customer and displays the result on the HMI touch panel. If the evaluation is negative, the workpiece will be removed from the process. Therefore, the robot places it in one of three intended deposit positions. If the inspection is successful, the workpiece will be reintegrated into the process, and the carrier can move to the last station.

At the handling station, the correct parts are lifted out of the process with a vacuum suction cup and are thus completed for the customer.

3.1.2 Components

As the process is known, the most important components working together are explained. In addition to the description of the stations with all their sensors and actuators, the ERP-Lab for Industry 4.0 by Lucas-Nuelle are also described.

IMS1: Conveyor Belt with PLC

The conveyor belt with PLC forms the basis for all other stations. The workpiece carriers are transported between the stations with a double strap. A DC motor is available for this purpose, making it possible to rotate the belt in both directions. Besides, two inductive proximity switches monitor the end positions of the belt. The conveyor belt can be seen in Fig. 3.4.



Figure 3.4: IMS1: Conveyor belt with PLC

All sensors and actuators are directly connected to a Siemens S7-1200 PLC. The IO signals are listed in Tab. 3.1.

Table 3.1: IMS1: Conveyor Belt with PLC

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

IMS3x: Assembly Station of the Bottom Part

The station for the assembly of the bottom parts is, like all other stations, placed on a conveyor belt (see Fig. 3.5). If the carrier has to be equipped with a bottom part, it will travel through the station until a stopper prevents it from continuing. Now a bottom part can be dropped onto the carrier from the stock containing a maximum of five elements. Releasing the stopper enables the carrier to continue.

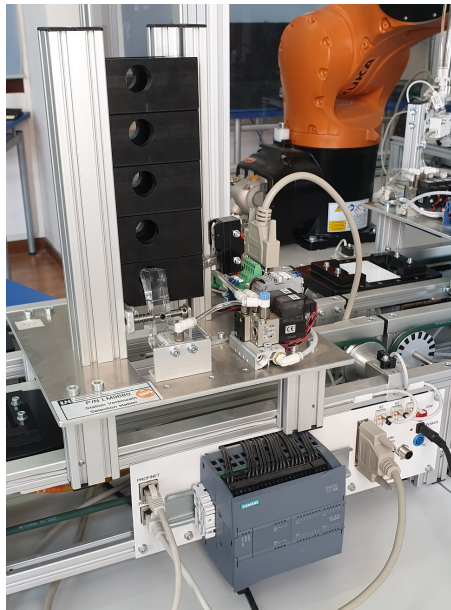


Figure 3.5: IMS3x: Assembly station of the bottom part

To realize the described process, the assembly station owns the IO signals listed in Tab. 3.2. Additionally to the sensors and actuators of the conveyor belt, the station consists of a magnetic sensor for monitoring the stopper position, a mechanical position switch for magazine occupancy and two electro-pneumatic valves for controlling the cylinders for separating the parts and controlling the stopper.

Table 3.2: IMS3x: Assembly Station of the Bottom Part

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

IMS4x: Assembly Station of the Top Part

The assembly station for the mounting of the top parts mainly corresponds to the structure of the IMS3x stations. Only mechanical adjustments to the size of the parts are required. For this reason, the IO signals match the elements in Tab. 3.2. The station is shown in Fig. 3.6.

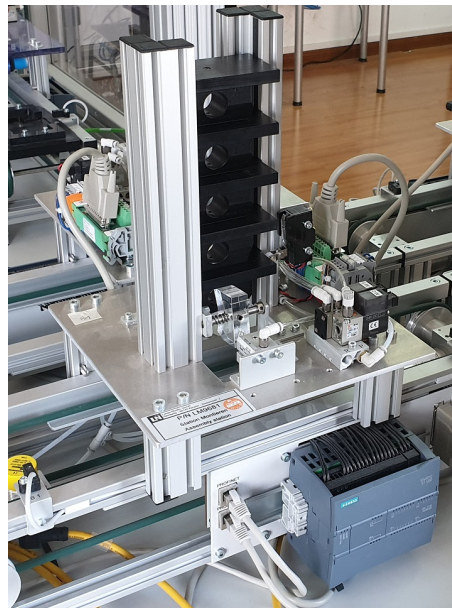


Figure 3.6: IMS4x: Assembly station of the top part

IMS5x: Assembly Station of the Bolt

In the stations IMS5x, top and bottom parts are pinned together. Fig. 3.7 shows the station for mounting the plastic bolts. Again, a stopper secures the correct positioning of the workpiece. Next, a pneumatic cylinder presses the pin from the stock into the intended fit in the workpiece. The assembly process is now complete, and the carrier can continue after the stopper is retracted.

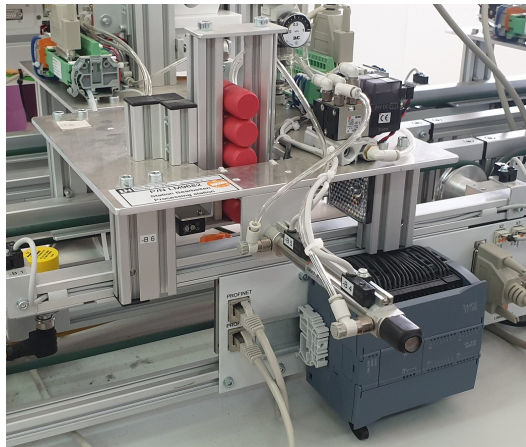


Figure 3.7: IMS5x: Assembly station of the bolt

In contrast to the monitoring of the stock occupancy at the IMS3x and IMS4x stations, light barriers with reflectors are used here. Besides, the press cylinder has two magnetic sensors for observing the end positions. Tab. 3.3 shows the overview of all IO signals of the IMS5x stations.

Table 3.3: IMS5x: Assembly Station of the Bolt

Description	ID	IO
End position sensor - left	B1	%I1.3
End position sensor - right	B2	%I1.4
Conveyor belt - right		%Q1.0
Conveyor belt - left		%Q1.1
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 pressing cylinder	M2	%Q0.2

IMS6: Testing Station

The testing station receives the workpiece from the KUKA robot. During the checking process, the carrier travels to the stopper and remains there for a short moment until the result is determined. Finally, the carrier returns to the end position of the conveyor belt, where the robot can pick up the workpiece (see Fig. 3.8).

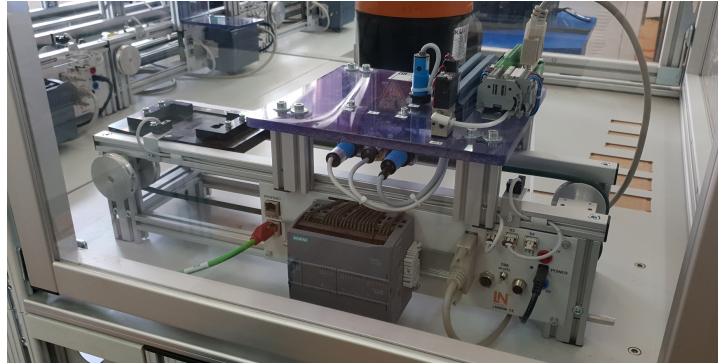


Figure 3.8: IMS6: Testing station

The station IMS6 has to be able to identify all possible variations of the workpieces. Therefore, it has four sensors whose functions are explained in the following paragraph.

Two optical sensors are positioned in a manner that one directs towards the bottom part and the other towards the top part. The sensors return a logical one for a white part and a logical zero for a black part. While the capacitive sensor is used to check whether a bolt is integrated into the workpiece, the inductive sensor can check if the material is out of plastic or metal. The hardware addresses can be found in Tab. 3.4.

Table 3.4: IMS6: Testing Station

Description	IO
End position sensor - left	%I1.3
End position sensor - right	%I1.4
Conveyor belt - right	%Q1.0
Conveyor belt - left	%Q1.1
Magnetic sensor - stopper at top	%I0.2
Optical sensor - bottom part	%I0.3
Inductive sensor	%I0.4
Capacitive sensor	%I0.5
Optical sensor - top part	%I0.6
Lower stopper	%Q0.0

IMS7: Handling Station

The handling station represents the last step in the production process. Equipped with a swivel table and a lifting cylinder, this station can pick up the workpieces from the process. If the carrier is correctly positioned, the swivel table can be actuated and moves into the pick-up position. When it arrives, the lifting cylinder extends, and the suction cups are placed on the workpiece. By activating the vacuum, the workpiece is fixed and can be transported. The lifting cylinder retracts, and the swivel table moves to the deposit. To place the workpiece in the storage position, the lifting cylinder extends once more, and the vacuum is deactivated. A maximum of two workpieces can be positioned on the deposit. The station IMS7 is shown in Fig. 3.9.



Figure 3.9: IMS7: Handling station

A sensor monitors the vacuum to ensure that the workpiece is suctioned. Furthermore, the swivel table is equipped with end position sensors indicating the pick-up and deposit position for the workpieces. Tab. 3.5 lists all signals for the IMS7.

Table 3.5: IMS7: Handling Station

Description	ID	IO
End position sensor - left	B1	%I1.3
End position sensor - right	B2	%I1.4
Conveyor belt - right		%Q1.0
Conveyor belt - left		%Q1.1
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	%I0.6
Swivel table from 0° to 90°	M1	%Q0.0
Lower stopper	M2	%Q0.1
Lower cylinder	M3	%Q0.2
Vacuum on	M4	%Q0.3

KUKA Robot

A 6-axis articulated manipulator is used to transport the workpieces. The robot is a KR6 R700 from KUKA. The system also includes a KRC4 controller and a Smartpad. In the idle position, the robot is waiting above station IMS5b (see Fig. 3.10). If the system informs the robot that a part is ready for transport, the robot will grab and transport it to the testing station. During the inspection process, the robot is waiting for the result. If the workpiece passes the check, the robot will reintegrate the workpiece into the production line. The robot places negatively checked workpieces in one of three designated deposit positions and then returns to its idle position. Once all three storage positions are occupied, the system operator has to clear them and confirm this on the touch panel.



Figure 3.10: KUKA robot

ERP-Lab for Industry 4.0

The ERP-Lab forms the basis for equipment management. It is structured as a cloud having no connection to the Internet and only being accessed from the plant's network. The data storage and the hardware are provided by the central computer which is the host for this application. The cloud runtime operates on a virtual Linux system. Fig. 3.11 shows the structure of this configuration.

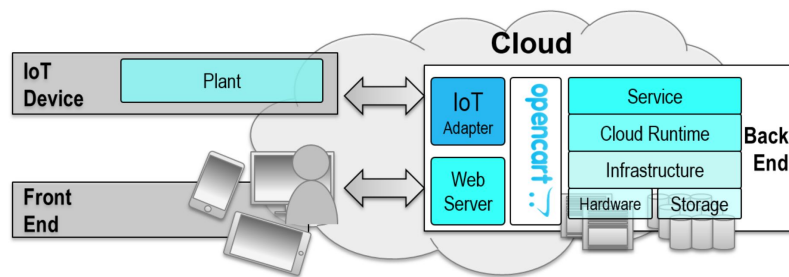


Figure 3.11: Structure of the ERP-Lab [40]

In this context, the Industry 4.0 plant represents the IoT device and is connected to the cloud service by an IoT adapter provided by the PLC. On the server side, the constrained application protocol (CoAP) is used for reading and replying to communication requests from the controllers. For the controller, only the interface specification is of importance [40]. The data is exchanged with the user datagram protocol (UDP) which is a transport protocol for sending and receiving data in IP-based networks.

A further interface to the cloud is the web server which can read and write data via HTML. It is the user interface and enables the operator to place orders. The system can also be configured and extended here. The web server offers various views whose functionality is summarized in the following list.

Order

In the order overview, all transacted orders are listed. Besides, information such as production duration, time of completion or order ID, the ordered workpieces can be viewed here.

Stock

In this tab, the plant operator can view the current status of each stock. It can be filled up to the maximum level of five by pressing the corresponding buttons.

SCADA

The supervisory control and data acquisition (SCADA) view is shown in Fig. 3.12. The

current status of the stations being integrated into the production line can be viewed here. This is the most important screen for the operator of the plant, as all error messages of the system appear here. As soon as the error has been corrected, it is possible to reset the station.

MES

The MES system is the interface between the webshop and the production line. In this menu, the layout of the system is configured, and the functions of the individual stations are defined. Furthermore, the workpiece carriers are added and the corresponding items assigned to the stations (see Fig. 3.2). An action table determines which stations are activated for each OpenCart order.

ERP

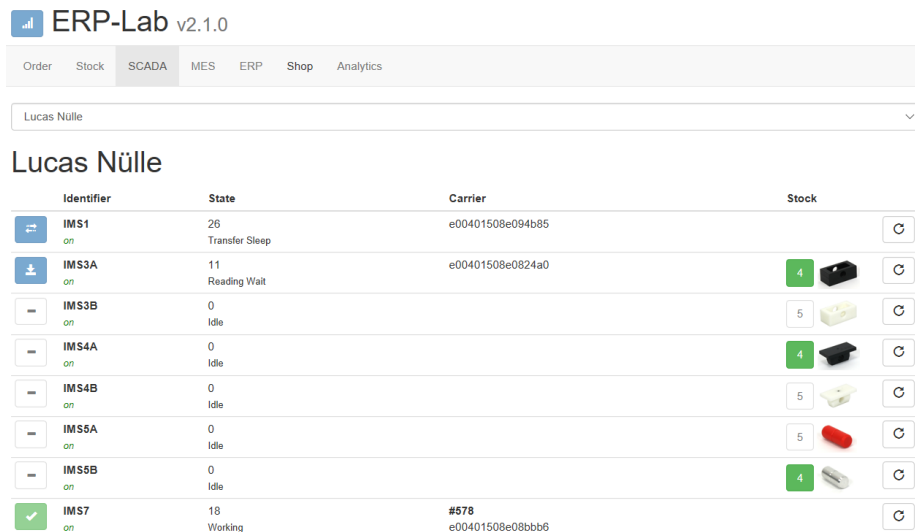
The enterprise resource planning (ERP) view allows the configuration of the webshop. Aside from creating new products, the existing ones can be modified. Further functions are design adjustments of the website or customer relationship management.

Shop

The webshop is used to generate new orders for the plant. The handling is similar to a standard e-commerce solution. All orders placed by the customer are transmitted into the MES database. The resulting action table contains all the information required for production.

Analytics

In Analytics, the data can be loaded from the ERP Lab database into a time series analysis software. As a result, data can be observed over a certain time period and additional alarms for the signals can be set.



ERP-Lab v2.1.0

Order Stock **SCADA** MES ERP Shop Analytics

Lucas Nülle

Lucas Nülle







Identifier	State	Carrier	Stock
IMS1 on	26 Transfer Sleep	e00401508e094b85	
IMS3A on	11 Reading Wait	e00401508e0824a0	4 
IMS3B on	0 Idle		5 
IMS4A on	0 Idle		4 
IMS4B on	0 Idle		5 
IMS5A on	0 Idle		5 
IMS5B on	0 Idle		4 
IMS7 on	18 Working	#578 e00401508e08bbb6	

Figure 3.12: SCADA tab of the ERP-Lab

3.2 Application of Industry 4.0

Distributed as a plant corresponding to the aspects of Industry 4.0, the considered production line applies modern production strategies. From self-organization to self-optimization, it works almost without human intervention and offers a high level of individualization of the desired customer products. The system informs the operator about the status of production and also assists in control.

The IMS stations form the technical basis of the entire system. Since they represent a combination of software and mechanical components communicating with the cloud via Ethernet interface of the controller, the IMS stations can be identified as CPS. In this case, the cloud represents the IoT and contains the virtual representation of the physical CPS. Due to the availability of the data, the virtual production planning can optimize the process by making decisions based on this information. For example, if a CPS reports an empty stock, workpieces that require parts from that stock will be queued, and other orders will be prioritized. As a result, more efficient production can be guaranteed while reducing the workload of operators [40].

So far, the focus of the collected plant data has been on pure production information. Thus, it is not possible to monitor the condition of the plant. For this reason, concepts for the implementation of predictive maintenance and plant monitoring are developed and applied in this thesis.

4 Requirements Analysis

In this chapter, the requirements placed on the topic of this thesis are defined, structured and summarized in tabular form. Section 4.2 compares the software applications for the realization of these requirements.

4.1 Functional Requirements

The following requirements relate to the extension of the Lucas-Nuelle Industry 4.0 plant. For the realization, it is necessary to select relevant data, to adapt the control technology while maintaining modularity, and to extend the communication by Modbus/TCP and OPC UA. In Tab. 4.1 all requirements are listed. The numbering specifies the priority, whereby number one is assigned the highest and number nine the lowest priority.

Table 4.1: Functional requirements

Number	Requirement
1	Transmission and display of relevant live data
2	Accessibility of historical data
3	Setting of simple thresholds for monitoring
4	Feedback to the operator about the current condition
5	Acknowledgment of current error states by the operator
6	Web interface for accessing collected information from the Internet
7	Enhancement opportunities by the implementation of machine learning
8	Support of open communication standards
9	Issue specific maintenance tickets

First of all, it is important to implement data transmission and display live data in appropriate software. The availability of the data in the network forms the foundation of concepts for live monitoring and predictive maintenance. On this basis, historical data of the plant also ought to be accessible. Many changes in the data, indicating possible error conditions, are only recognizable over a longer observation period. Hence, the insight of historical data can be used profitably in this context.

Since the availability of data alone is not sufficient for reliable monitoring, the possibility of setting simple thresholds has to be given in the first step. Violating these limits requires notification of the operator about the current situation. At the same time, the operator has to be able to acknowledge these alarms or issue maintenance tickets to remedy the causes.

All relevant plant's information should be accessible via web interface. Furthermore, the user ought to be able to configure the display according to his needs.

If more complex maintenance concepts have to be applied to this plant, the system shall be capable of enhancement by introducing machine learning. As a result, predictive maintenance, using appropriate algorithms, leads to benefits, like the provision of early warning of error conditions. In order to enable simple integration of additional devices into the existing system, the solution has to support open communication standards.

4.2 Comparison between Machine Advisor and MATLAB

In this section, the applicability of both development tools for the pre-defined requirements is reviewed. Focusing on the theoretical possibility of implementation and not on assessing feasibility, the results are listed in Tab. 4.2.

Table 4.2: Comparison between Machine Advisor and MATLAB

Number	Machine Advisor	MATLAB
1	✓	✓
2	✓	✓
3	✓	✓
4	✓	✓
5	✓	✓
6	✓	✓
7	X	✓
8	X	✓
9	✓	X

As shown in Tab. 4.2, neither application is able to satisfy all requirements individually. For this reason, both solutions are developed simultaneously. To keep the workload within the scope of this master thesis, the requirements are split between both tools. Only the accessibility of the live data, being the fundament for each further work step, has to be available in both solutions.

The Machine Advisor, promoted as a cloud-based service platform for monitoring performance data, provides a configurable web interface. The website ought to display live machine data and furthermore enable access to historical data. Based on this information, the plant's operator has to be informed about the violation of defined limit values and can acknowledge them. If the cause is identified, maintenance tickets can be created and scheduled using the web interface.

MATLAB stands out in particular due to its wide range of capabilities. By providing various toolboxes, it offers plenty of room for future enhancements. With the data available in MATLAB, a more detailed assessment of the condition of the system can be made. Applying machine learning algorithms ought to extend the analysis of the data.

5 Hardware Configuration

Before the software development can begin, the structure of the network and the necessary hardware configuration are explained. Once all participants are known, the setup of the communication for the data transfer is described step by step. First of all, the connection between the different controllers and the robot is established, allowing access to all data on a central controller. Secondly, it is explained how the data is transferred to the analysis and visualization software Machine Advisor and MATLAB.

5.1 Plant Networking

In this section, the structure of the network, including all its participants, is explained. For visualization Fig. 3.3 is extended. In addition to the known components, Fig. 5.1 contains the personal computer (PC), the *PLC_1*, and the ComX Box. The black lines correspond to the production-relevant connection of the stations, and the blue lines indicate the communication connections.

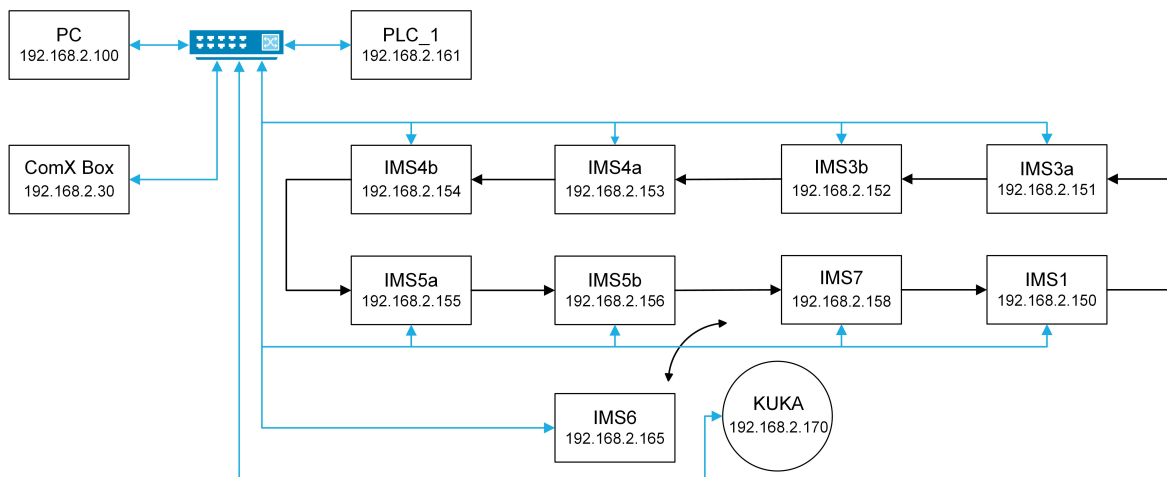


Figure 5.1: Plant network

The *PLC_1*, a Siemens S7-1500 controller, assumes an essential role for data collection, storing all relevant plant information provided by the controllers and the robot. Furthermore, it is an interface to other participants of the network not using the TIA Portal development tool. For the connection to MATLAB, running as an application on the PC, the *PLC_1* provides an onboard OPC UA server. Simultaneously, it operates as a Modbus/TCP server for communicating with the ComX Box.

5.2 Communication between Controllers

Since each station has a controller linked to the sensors and actuators, simple deterministic communication is essential. The I-Device function is used for the transfer of all necessary information to the *PLC_1* acting as an IO-Controller. Thus, the stations become IO-Devices and can send and receive data via Profinet IO.

With this type of communication, configurable virtual IO is added to the controllers. In this application, only the I-Device ought to send data to the IO-Controller, therefore a unidirectional connection is sufficient. The configuration is done in the TIA Portal and is shown in Fig. 5.2.

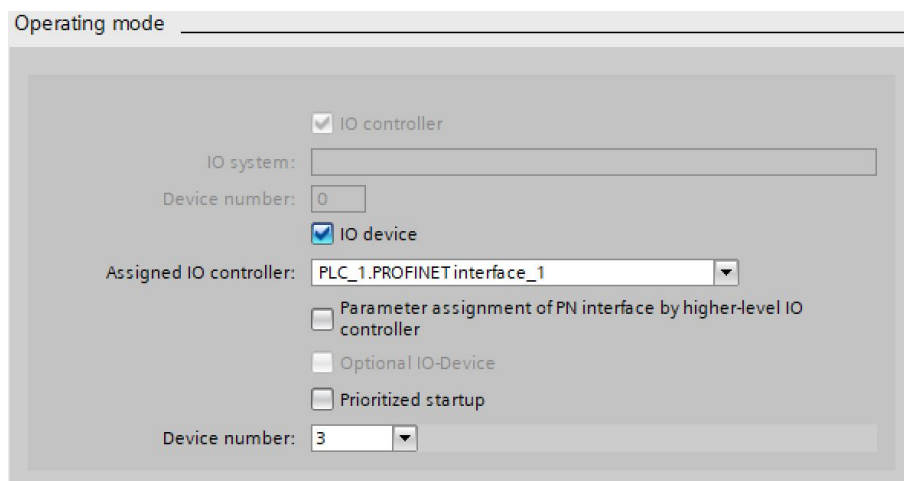


Figure 5.2: Operating mode of I-Device



At first, the I-Device functionality has to be activated by setting the check mark, allowing to select and assign an IO-Controller. It is crucial that the device number is unique in the network. Otherwise, error messages will occur. This setting is necessary for each station of the system. Thereby all PLCs are assigned to *PLC_1*, the IO-Controller in this application.

In the next step, the transfer area for the communication between the IO-Controller and the I-Device has to be defined. As an example, this is done for station IMS1. The configuration is documented in Fig. 5.3.

Details of the transfer area

Transfer area: DATA_TO_MASTER

Type of transfer area: CD

Partner:  Local: 

Data exchange between: PLC_1 and ims_1

Subslot: 1002 (Partner) / 1002 (Local)

Address type: I (Partner) / Q (Local)

Start address: 500 (Partner) / 200 (Local)

Organization block: --- (Automatic update) (Partner) / --- (Automatic update) (Local)

Process image: Automatische Aktualisieru... (Partner) / Automatische Aktualisieru... (Local)

Length: 100 Bytes

Comment: Used for data collection

Enable PROFinergy communication

Figure 5.3: Configuration of the transfer area

More than one transfer area can be created and identified by its individual name. For data collection, the identifier *DATA_TO_MASTER* is used. Below, both communication partners are displayed. To ensure unidirectional communication between both devices, the signals of *ims_1* are configured as Q¹ and the signals of *PLC_1* as input (I).

The complete transfer area can be defined by the starting address and the length. The local start address is mapped to the start address of the communication partner, causing a direct change in the address range of the IO-Controller if there is any status change in the address range of the I-Device. For the data collection, 100 bytes are allocated. In case this amount of data is not sufficient, it can be extended here. These settings only have to be made locally on the I-Device and are automatically applied by the IO-Controller.

Following this procedure, the configuration for all further stations is executed. The structure of the I-Device communication is schematically shown in Fig. 5.4.

¹ Abbreviation for output in the TIA Portal

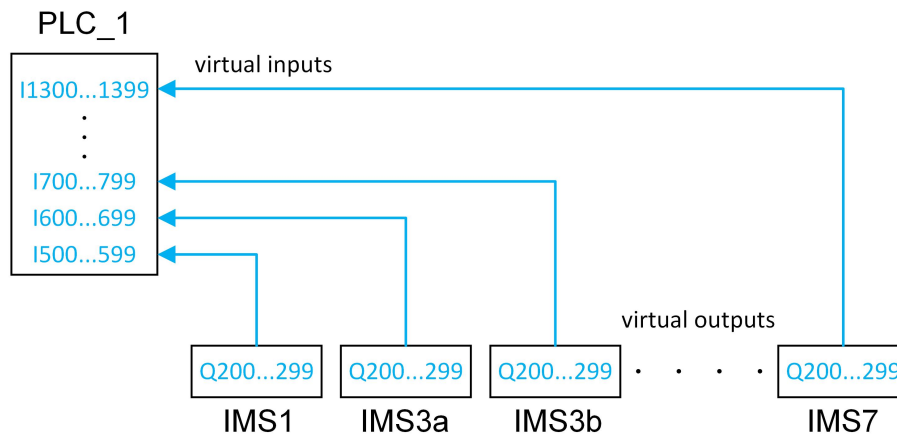


Figure 5.4: I-Device communication

In order to maintain the modularity of the control software, the PLCs of the stations are configured identically. To prevent data overlapping in the IO-Controller, the information of the individual stations are located in successive address areas. The complete assignment is documented in Tab. 5.1.

Table 5.1: Addresses of I-Device communication

Station	PLC_1 Address	Local Address	Size
IMS1	I 500...599	Q 200...299	100 bytes
IMS3a	I 600...699	Q 200...299	100 bytes
IMS3b	I 700...799	Q 200...299	100 bytes
IMS4a	I 800...899	Q 200...299	100 bytes
IMS4b	I 900...999	Q 200...299	100 bytes
IMS5a	I 1000...1099	Q 200...299	100 bytes
IMS5b	I 1100...1199	Q 200...299	100 bytes
IMS6	I 1200...1299	Q 200...299	100 bytes
IMS7	I 1300...1399	Q 200...299	100 bytes

5.3 Data Exchange with KUKA Robot

As already described in Section 3.1.2, the KUKA robot has a KRC4 controller. The program of this controller programmed using KUKA.WorkVisual has to be adapted for the data collection.

Since the robot has to be specified as an IO-Device with a fixed IO module size in the TIA Portal, the data being transferred is selected first. Subsequently, the configuration in KUKA.WorkVisual and the TIA Portal is described.

5.3.1 Selection of Robot Transfer Data

Aim of monitoring the robot is to obtain information about the state of each axis. To give a better overview of the 6-axis articulated manipulator, it is shown in Fig. 5.5.

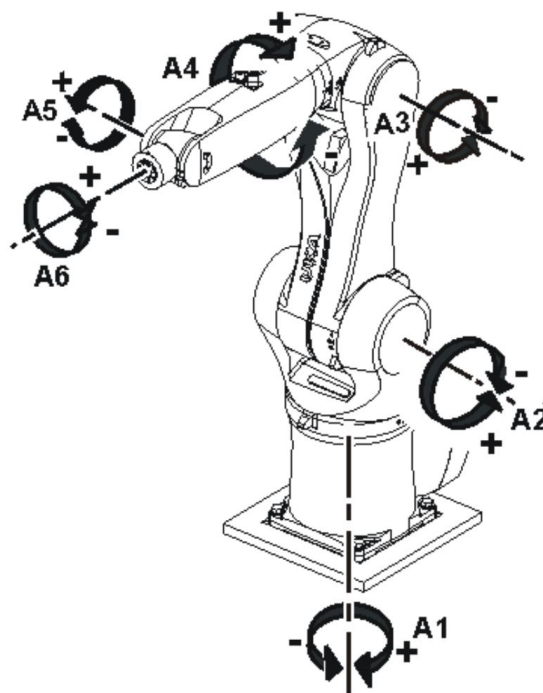


Figure 5.5: Structure of the ERP-Lab [41]

The robot is mounted on a rotating base driven by the axis. The unit between the arm and the base is called linking arm which is controlled by axis A2 and contains the motor for moving the arm (A3). Ultimately, the arm connects the 3-axis central joint consisting of axes A4, A5, and A6.

Information about all axes can be collected by using the system variables of the KRC4 controller. These variables contain configuration parameters, motion parameters, and reference values as well as information on current positions, velocity, and temperatures. The following variables are used to monitor the robot.

$\$CURR_ACT[AxisNumber]$

Returns the actual current of each robot axis as a percentage of the maximum amplitude from -100.0% to +100.0%. The data type is a REAL floating-point number with a length of 32 bits.

 $\$MOT_TEMP[AxisNumber]$

Returns the current temperature of each motor, measured in Kelvin with a tolerance of ± 12 K. The data type is an integer (INT) with a length of 16 bits.

 $\$TORQUE_AXIS_ACT[AxisNumber]$

Returns the current torque of the motor for each robot axis in Newton meters. The data type is a REAL floating-point number with a length of 32 bits.

 $\$VEL_AXIS_ACT[AxisNumber]$

Returns the current motor speed of each robot axis as a percentage of the maximum speed from -100.0% to +100.0%. The data type is a REAL floating-point number with a length of 32 bits.

The label *AxisNumber* has to be replaced by the corresponding axis number for use in the program code. Axis A1 corresponds to number one, Axis A2 corresponds to number 2, and the other identifiers are following this pattern.

During the execution of this thesis, overlaps have occurred in the transfer. These overlaps have only occurred in the address range of the motor temperature. Increasing the variable to 32 bits, similar to the other variables, has solved this problem. As a result of the four system variables selected for each axis,

$$32 \text{ bits} \cdot 6 \cdot 4 = 768 \text{ bits} \quad (5.1)$$

are required to receive and interpret all data in the controller. To provide a reserve for possible extension by additional variables, the module with 1536 IO signals is selected.

5.3.2 Configuration in KUKA.WorkVisual

In KUKA.WorkVisual the Profinet communication has to be added to the bus structure, allowing the Profinet IO connection to be configured. Before the variables can be mapped, the size of the Profinet device has to be specified. For this purpose, the Profinet settings have to be opened (see Fig. 5.6).

PROFINET	
Device name:	kukaIn
PROFINET device	
<input checked="" type="checkbox"/>	Activate PROFINET device stack
Number of safe I/Os:	0
Number of I/Os:	1536
Profinet version:	v8.3, PNet 3.3
Update time:	8 ms
Bus timeout:	20000 ms
<input type="checkbox"/>	Display diagnostic alarm as message
<input type="checkbox"/>	Transmit device alarms to PLC
PROFINET controller	
Update time:	2 ms
Bus timeout:	20000 ms

Figure 5.6: KUKA Profinet configuration

The device name defines the name for the robot in the network. Placing the check mark for activation the Profinet device stack leads to the S7-1500 being able to send control data to the robot. Also, the required number for the IO signals can be specified. As defined in 5.3.1, the module with 1536 IO is selected here. Finally, the Profinet version has to be set to v8.3 and PNet 3.3, and the other settings remain with the default settings.

According to this configuration, the KUKA controller has internal IO that have to be linked to the IO of the Profinet fieldbus. For this purpose, KUKA.WorkVisual offers an IO mapping editor. As shown in Fig. 5.7, the internal IO has to be selected on the left and the Profinet fieldbus on the right.

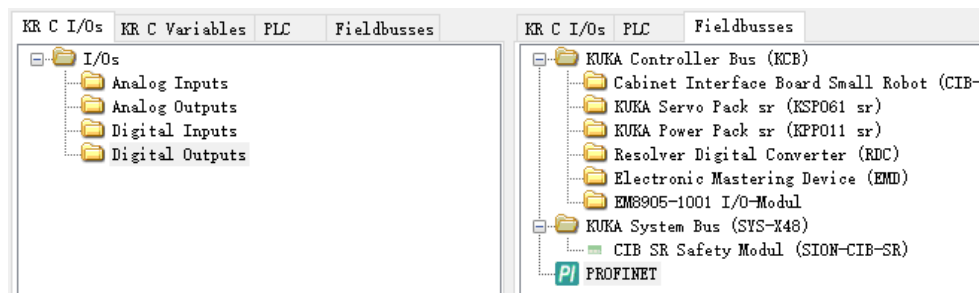


Figure 5.7: IO mapping editor

The window from Fig. 5.7 is followed by the actual Mapping Editor, consisting of three tables. Whilst the upper table shows all already linked signals, the other two tables are showing

the internal and the fieldbus IO signals. By default, only the individual IO are displayed as boolean data types. Nevertheless, since 32 bits are necessary for the transmission of the individual variables, the corresponding data area has to be grouped before linking.

The internal signals of the KRC4 controller can be combined to different data types by marking corresponding blocks of signals. Therefore, 32 bit are marked and combined to a double word (DWORD) for each variable. As the default settings already map system variables to internal IO, the lowest address used for data collection is 2000.

The integrated signal editor is used for the clustering of the fieldbus signals, allowing the combination of 32 boolean output signals to a DWORD by drag and drop. In order to keep a certain distance to the control signals, the address of the first data entry is 20. The configured IO mapping editor is shown in Fig. 5.8.

Name	Type	Description	I/O	I/O	Name	Type	D...	Addr...
\$OUT[2000]#G	DWORD		→	→	CURR_ACT_Axis1	DWORD		20
\$OUT[2032]#G	DWORD		→	→	CURR_ACT_Axis2	DWORD		24
\$OUT[2064]#G	DWORD		→	→	CURR_ACT_Axis3	DWORD		28
\$OUT[2096]#G	DWORD		→	→	CURR_ACT_Axis4	DWORD		32
\$OUT[2128]#G	DWORD		→	→	CURR_ACT_Axis5	DWORD		36
\$OUT[2160]#G	DWORD		→	→	CURR_ACT_Axis6	DWORD		40
\$OUT[2192]#G	DWORD		→	→	MOT_TEMP_Axis1	DWORD		44
\$OUT[2224]#G	DWORD		→	→	MOT_TEMP_Axis2	DWORD		48
\$OUT[2256]#G	DWORD		→	→	MOT_TEMP_Axis3	DWORD		52
\$OUT[2288]#G	DWORD		→	→	MOT_TEMP_Axis4	DWORD		56
\$OUT[2320]#G	DWORD		→	→	MOT_TEMP_Axis5	DWORD		60

Name	Type	Description	I/O	I/O	Name	Type	D...	Ad...
\$OUT[2000]#G	DWORD		→	→	CURR_ACT_Axis1	DWORD		20
\$OUT[2032]#G	DWORD		→	→	CURR_ACT_Axis2	DWORD		24
\$OUT[2064]#G	DWORD		→	→	CURR_ACT_Axis3	DWORD		28
\$OUT[2096]#G	DWORD		→	→	CURR_ACT_Axis4	DWORD		32
\$OUT[2128]#G	DWORD		→	→	CURR_ACT_Axis5	DWORD		36
\$OUT[2160]#G	DWORD		→	→	CURR_ACT_Axis6	DWORD		40
\$OUT[2192]#G	DWORD		→	→	MOT_TEMP_Axis1	DWORD		44
\$OUT[2224]#G	DWORD		→	→	MOT_TEMP_Axis2	DWORD		48
\$OUT[2256]#G	DWORD		→	→	MOT_TEMP_Axis3	DWORD		52
\$OUT[2288]#G	DWORD		→	→	MOT_TEMP_Axis4	DWORD		56
\$OUT[2320]#G	DWORD		→	→	MOT_TEMP_Axis5	DWORD		60
\$OUT[2352]#G	DWORD		→	→	MOT_TEMP_Axis6	DWORD		64
\$OUT[2384]#G	DWORD		→	→	TORQUE_ACT_Axis1	DWORD		68
\$OUT[2416]#G	DWORD		→	→	TORQUE_ACT_Axis2	DWORD		72

Figure 5.8: Mapping fieldbus IO signals

The internal signals are thus directly coupled to the signals of the fieldbus and can be transmitted. Tab. 5.2 shows an overview of all configured addresses of the variables.

Table 5.2: Configuration of robot variables

Identifier	Output Address	Fieldbus Address	Size
CURR_ACT_Axis1	2000	20	32 bytes
CURR_ACT_Axis2	2032	24	32 bytes

Table 5.2: Configuration of robot variables

Identifier	Output Address	Fieldbus Address	Size
CURR_ACT_Axis3	2064	28	32 bytes
CURR_ACT_Axis4	2096	32	32 bytes
CURR_ACT_Axis5	2128	36	32 bytes
CURR_ACT_Axis6	2160	40	32 bytes
MOT_TEMP_Axis1	2192	44	32 bytes
MOT_TEMP_Axis2	2224	48	32 bytes
MOT_TEMP_Axis3	2256	52	32 bytes
MOT_TEMP_Axis4	2288	56	32 bytes
MOT_TEMP_Axis5	2320	60	32 bytes
MOT_TEMP_Axis6	2352	64	32 bytes
TORQUE_ACT_Axis1	2384	68	32 bytes
TORQUE_ACT_Axis2	2416	72	32 bytes
TORQUE_ACT_Axis3	2448	76	32 bytes
TORQUE_ACT_Axis4	2480	80	32 bytes
TORQUE_ACT_Axis5	2512	84	32 bytes
TORQUE_ACT_Axis6	2544	88	32 bytes
VEL_ACT_Axis1	2576	92	32 bytes
VEL_ACT_Axis2	2608	96	32 bytes
VEL_ACT_Axis3	2640	100	32 bytes
VEL_ACT_Axis4	2672	104	32 bytes
VEL_ACT_Axis5	2704	108	32 bytes
VEL_ACT_Axis6	2736	112	32 bytes

5.3.3 Configuration in TIA Portal

In order for the PLC to access the fieldbus variables, the KUKA robot has to be adapted in the TIA portal. Before the robot can be inserted as an IO-Device, the GSD file for the robot controller has to be imported. After a successful installation, the robot is added to the hardware catalog and can be integrated into the project (see Fig. 5.9).

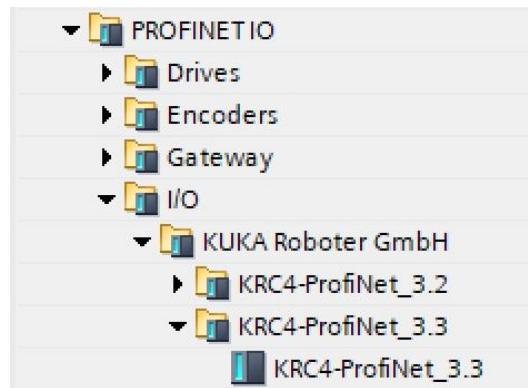


Figure 5.9: KUKA IO-Device in TIA Portal

The KRC4 controller can be inserted into the TIA Portal network view via drag and drop. In this view the connection to the Profinet interface of the *PLC_1* is established. This connection is indicated in the complete network view in Fig. 5.10 by the green line.

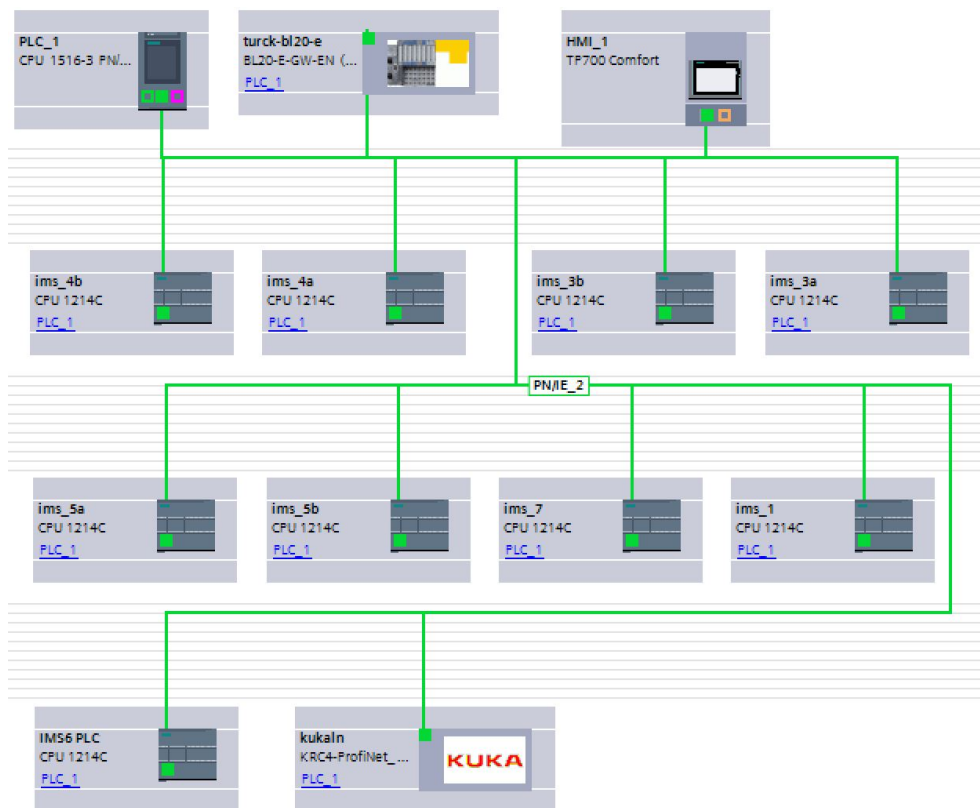


Figure 5.10: TIA Portal network view

The IO module for the KUKA controller is selected in the device view, having to match the selection in KUKA.WorkVisual. Before the variables from the robot can be used within the TIA Portal program, they have to be read from the fieldbus. The defined address range maps the data to the addresses 150 to 341. These addresses are treated in the program, similar to the virtual inputs of the I-Devices, as regular hardware inputs.

5.4 Connection to Machine Advisor

In the next step, the data has to be transferred from the TIA Portal to the analysis software. Therefore, the communication to the Machine Advisor ought to be established first. In the following, the enhancements in the TIA Portal as well as the configuration of the ComX Box and the Machine Advisor are explained.

5.4.1 Setup of Modbus TCP/IP Server

The *PLC_1* running the Modbus TCP/IP server, provides specific instructions for this application, implementing the functionality of the Modbus communication and thus just require parameterization. No additional hardware module is necessary for the application, because communication with the network is possible directly via the Profinet interface.

The server receives connection requests from the client as well as Modbus function requests, and sends response messages. The ComX Box is the client sending the command to read the holding register and receives the content as a response (see Fig. 5.11).

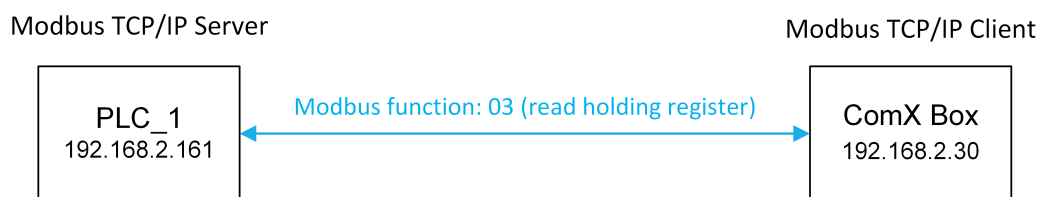
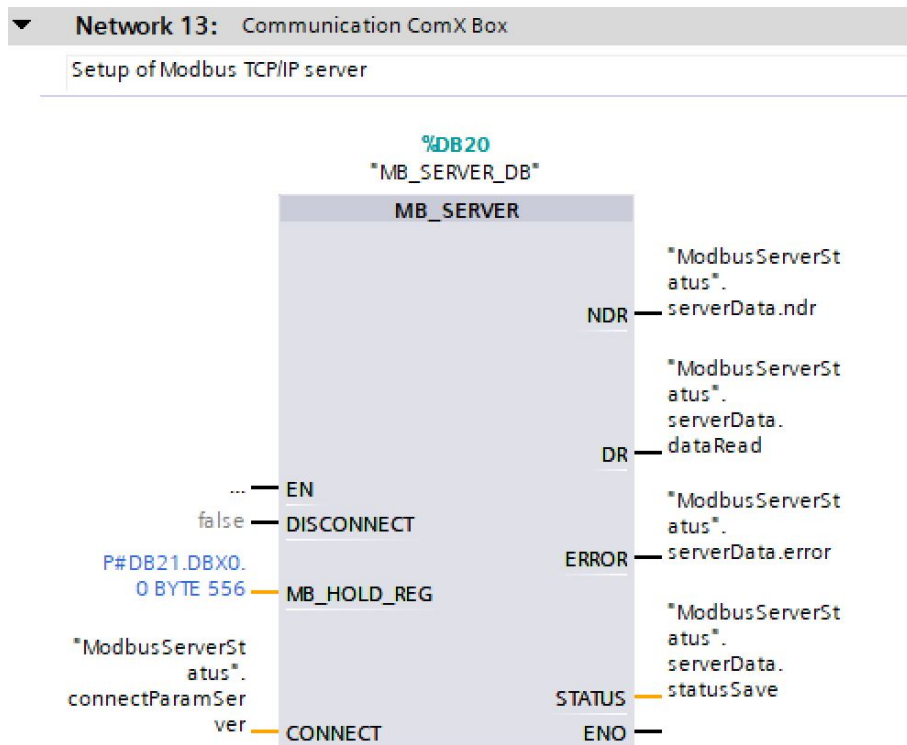


Figure 5.11: Communication between Modbus TCP/IP server and client

The *MB_SERVER* instruction is used to activate the server on the controller. To ensure continuous execution, it is called cyclically in the organization block *OB1*. The corresponding network is shown in Fig. 5.12.

Figure 5.12: *MB_Server* function block

The block has predefined IO ports requiring appropriate parameterization. The holding register *MB_HOLD_REG*, containing all relevant data of the plant, is implemented using a data block. Therefore, the data block *ModbusServerData* with the number 21 is created. The attributes of the block have to be modified as shown in Fig. 5.13 to enable access by a pointer.

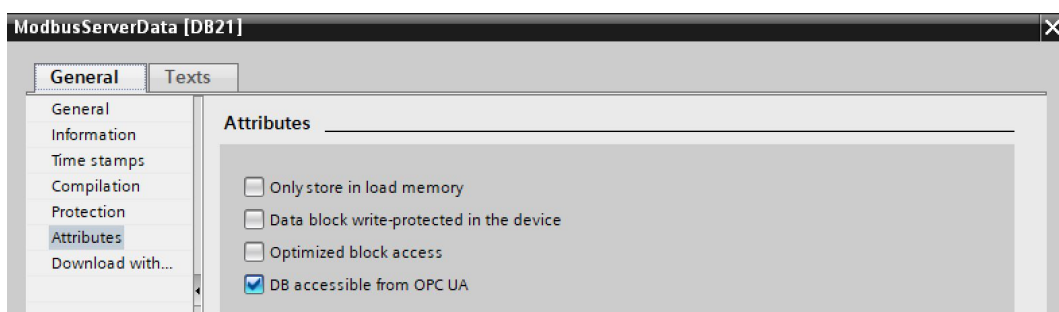


Figure 5.13: Data block attributes for Modbus server data

By default, the optimized module access is activated, reducing the memory requirement as

it is without a fixed structure. Yet, communication with the ComX Box needs a fixed structure assigning a fixed address to the data. Consequently, optimized access has to be deactivated. The expression *P#DB21.DBX0.0 Byte 556* creates a pointer to the first element of the data block with a length of 556 bytes and assigns the content of the *ModbusServerData* to the holding register. The length has to be chosen according to the size of the data block.

The other parameters of the *MB_SERVER* instruction are summarized in the data block *ModbusServerStatus* (see Fig. 5.14).

ModbusServerStatus							
	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	▼ Static			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	▼ connectParamServer	TCON_IP_v4		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3	InterfaceId	HW_ANY	64	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
4	ID	CONN_OUC	2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
5	ConnectionType	Byte	11	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	ActiveEstablished	Bool	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	▼ RemoteAddress	IP_V4		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
8	▼ ADDR	Array[1..4] of Byte		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
9	ADDR[1]	Byte	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
10	ADDR[2]	Byte	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
11	ADDR[3]	Byte	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
12	ADDR[4]	Byte	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
13	RemotePort	UInt	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
14	LocalPort	UInt	502	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
15	serverData	Struct		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 5.14: Connection parameters for Modbus communication

The parameter *connectParamServer*, including all necessary variables for communication, is particularly important. For that reason, the individual values are explained in the following list.

InterfaceId

The interface ID is the hardware identifier of the local interface. Since the Profinet interface [X1] is used for communication, this hardware ID has to be used. The identifier can be found in the device view of the *PLC_1* in the list of hardware system constants.

ID

The ID identifies the connection and has to be unique within the CPU. Both server and client ought to have the same ID for Modbus communication.

ConnectionType

The connection type is determined by 11 (decimal), representing a TCP connection.

ActiveEstablished

ActiveEstablished is the identifier for the type of connection setup. A logical *false* indicates a passive connection setup and a logical *true* indicates an active connection setup. As the client controls the connection setup in this case, the value is set to *false*.

RemoteAddress

The remote address contains the IP address of the connection partner. Using *0.0.0.0* as the address, the Modbus server allows connection requests from any communication partner.

RemotePort

The remote port determines the port used for communication by the remote partner. In order for the Modbus server to accept requests from any port of the client, the port is set to the value 0.

LocalPort

The local port defines the port used to receive the client's connection requests. Accordingly, the port number of the client, port 502, has to be used.

This parameterization enables the Modbus client to access the data within the holding register. The retentivity is set to ensure that the configuration is permanently maintained.

Besides the variables for the connection configuration, the structure *serverData* is created, containing information of the current status of the connection, including, for example, the display of incorrect parameterization or other error messages.

The Modbus TCP/IP server is now completely set up in the TIA Portal. During programming, it is only necessary to ensure that the data is written to the data block *ModbusServerData* and thus transferred to the holding register.

5.4.2 Configuration of ComX Box and Machine Advisor

Before the ComX Box functions as an interface between the *PLC_1* and the Machine Advisor, several configurations have to be made.

First, a parameter template ought to be created, as this is necessary for the configuration of Modbus communication. As soon as all data to be transmitted have been defined, the template can be updated. Until the table will be completed in Chapter 6, it is used without data entries.

The table has to contain predefined columns, allowing the entered information to be interpreted by the Machine Advisor. From the left, the columns are data-name, parameter category, monitoring type, alarm level, curve type, short name, ID, unit, scale factor, data type,

transmission type, transmission frequency, address, size, upload, and remarks. The final table can be found in the appendix of this thesis. In the platform management of the Machine Advisor, the created Excel table can be uploaded as a parameter template (see Fig. 5.15).

All Templates Download Upload

Template Type	Template Name	Upload User	Upload Date	Operation
Parameter Template	USST industrylab4.0.xlsx	jianqiangshen	2019-06-19 16:46	Download Upload Record

Figure 5.15: Parameter template

As the ComX Box maps the collected data from the holding register to the entries in the table, it is important that the parameter template has the same structure as the data block in the *PLC_1*. Otherwise, the entries in the Machine Advisor may not be assigned to the appropriate devices of the plant.

During the next step, the Modbus communication is configured. Therefore, a device is added to the equipment management of the ComX Box. The parameters to be set are shown in Fig. 5.16.


所属机柜	<input type="text" value="Industry 4.0 Lab"/>	应用包	设备名称	Industry 4.0 Lab
设备描述	Industry 4.0 Lab		设备ID	2
设备总称	USST		设备类型	Machine
设备型号	A1		数据模板	USST industrylab4.0
通讯协议	Modbus TCP/IP		设备地址	255
IP	192.168.2.161		端口	502

Figure 5.16: ComX Box Modbus TCP/IP configuration

The device description *Industry 4.0 Lab* as well as the overall name of the equipment *USST* are shown in the left half. Below, the communication protocol and the IP address of the Modbus TCP/IP server have to be selected. The IP address corresponds to the address of the *PLC_1*. The right half contains, among other parameters, the name of the device and the device ID which has to match the ID set in the Modbus server. Furthermore, the communication port 502 and the parameter template *USST industrylab4.0* are selected here.

These settings enable the ComX Box to read the data from the controller and map it to the entries of the parameter template. The local stored data ought to be uploaded into the cloud and as a result be available in the Machine Advisor. The ComX Box has a subscriber identity

module (SIM) card allowing the connection to the Internet. The operating mode has to be set to 3G in the network settings (see Fig. 5.17) to use this interface.



Operating mode Ethernet 1 Ethernet 2 3G

Cloud platform connection

Ethernet 1

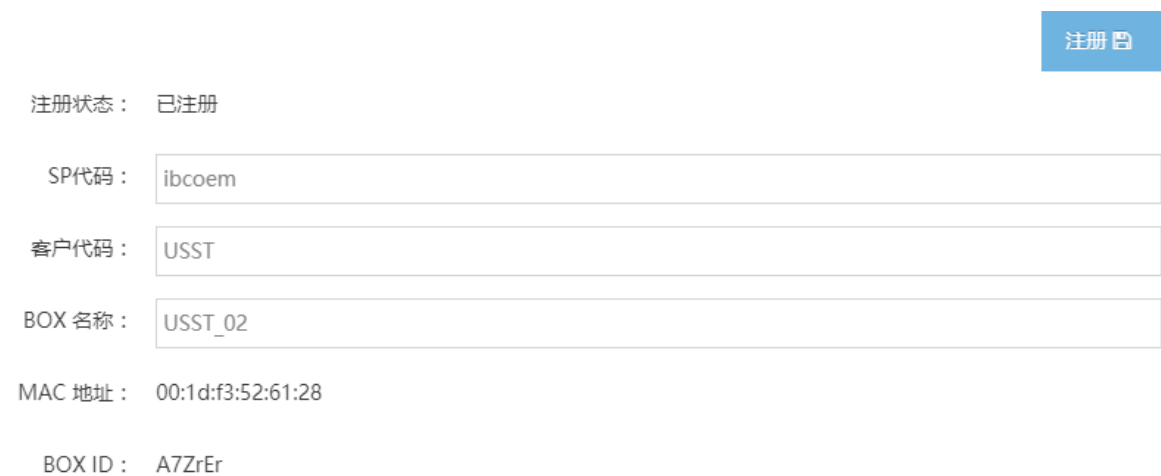
Ethernet 2


3G

save 

Figure 5.17: Operating mode of the ComX Box

Next, the cloud platform communication has to be configured. In Fig. 5.18, the name of the box and the customer name, as well as the MAC address and ID of the box, are defined. After the configuration has been uploaded, these settings are assumed by the Machine Advisor.



注册 

注册状态 : 已注册

SP代码 :

客户代码 :

BOX 名称 :

MAC 地址 : 00:1d:f3:52:61:28

BOX ID : A7ZrEr

Figure 5.18: Definition of the gateway

The actual upload settings have to be configured in the FTP settings (see Fig. 5.19). Here, the address of the server and the FTP port for the connection to the Machine Advisor are entered as well as memory and upload cycles are set. Both cycles are at the lowest time

interval of 15 minutes in order to provide as many data as possible to the Machine Advisor. In the last entry, the type of access to the server address is defined. The selection allows active or passive access, whereby active access has to be selected for this application.

服务器地址: ftp.energymost.com

FTP端口: 21

保存周期: 15分钟

上传周期: 15分钟

模式: 主动 被动

Figure 5.19: Upload settings

Once all settings have been made, the configuration has to be uploaded to the cloud platform. This can be done in the configuration synchronization tab of the cloud platform settings.

In the Machine Advisor, below the gateway tab, the ComX Box can be selected to display the available devices, including the configured ComX Box with the gateway name *USST_02*. Clicking on this gateway opens the window in Fig. 5.20.

ComX BOX Gateway Detail

Gateway Name	Gateway ID	Gateway Status	Software Version
USST_02	A7ZrEr	Online	Latest Version 1.4.1 Latest Version 1.6.3

Sync configuration with cloud Upload alarm data Upload real-time data Gateway Offline Alarm SMS

SIM Card	LAN1	LAN2	WIFI
Operator: China Unicom Network Standard: LTE Used Flow: 538MB Flow Threshold: --	Mac Address LAN1 Internet Access: 00:1d:f3:52:61:28 IP Address LAN1 Internet Access: 192.168.1.30	Mac Address LAN2 Internet Access: 00:1d:f3:52:61:29 IP Address LAN2 Internet Access: 192.168.2.30	Mac Address Wireless Network Card: -- IP Address Station Model: --

Figure 5.20: Configured ComX Box gateway

The ComX Box is now linked to the Machine Advisor and transmits the information of the Industry 4.0 plant. Fig. 5.20 shows the current connection status of the SIM card and enables the activation and deactivation of configuration synchronization, the upload of alarm data, and the upload of real-time data. Furthermore, a short message service (SMS) can be send to the plant operator, if the gateway is offline. The operator of the plant to be contacted in case of error occurrences can be defined in the alarms view of the production line.

5.5 Establishment of OPC UA Communication

The communication between MATLAB and *PLC_1* is done via OPC DA. Accordingly, a client-server structure has to be set up. For the server, the onboard OPC UA server functionality of the Siemens S7-1500 controller is used, and the OPC UA client is implemented in MATLAB.

5.5.1 Activation of OPC UA Server

The onboard OPC UA functionality of the *PLC_1* controller has to be activated in the device configuration. In the settings for OPC UA, the accessibility of the server can be activated by setting a check mark (see Fig. 5.21).

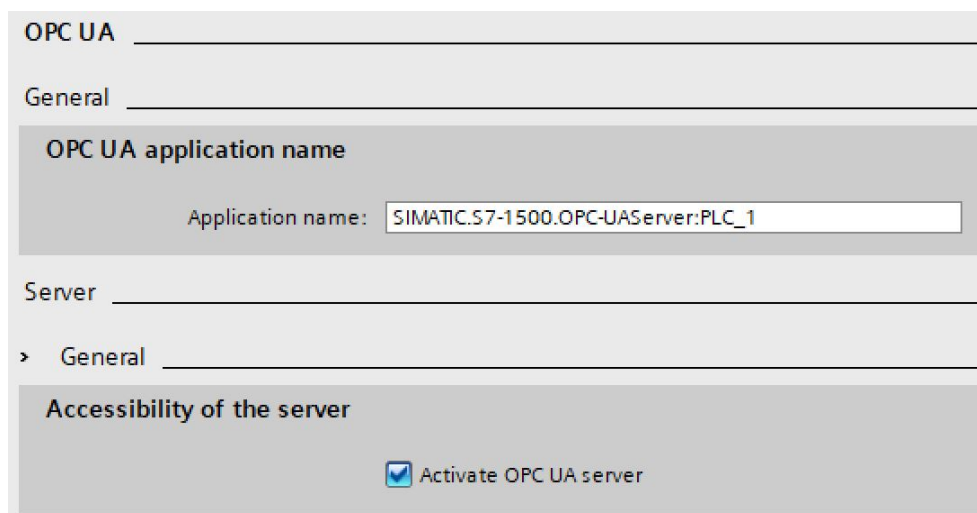
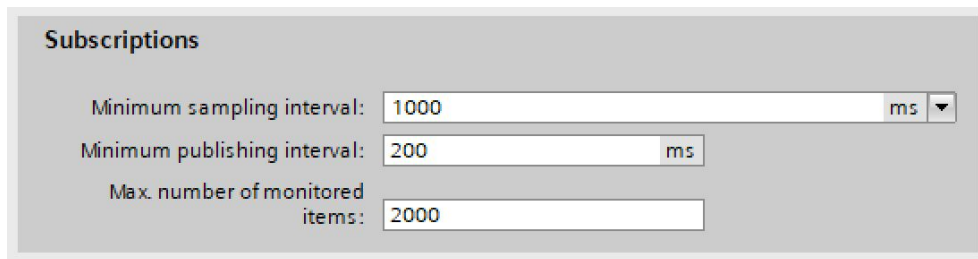


Figure 5.21: Activation of the OPC UA Server

Due to this setting, the OPC UA server is running on the controller and is available in the network. The accessibility from OPC UA check mark has to be activated to ensure visibility

of the variables for the client. This option can be selected directly in the variable list whilst creating the variable or can be defined for entire data blocks.

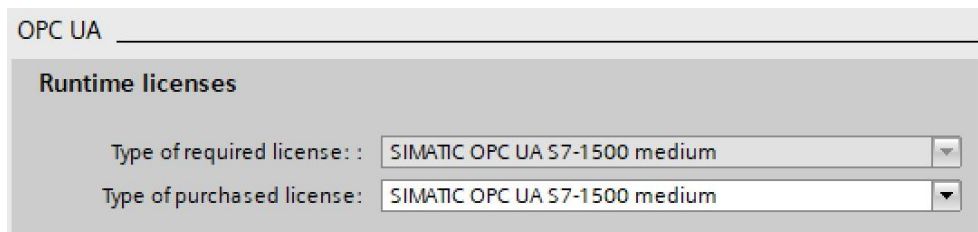
Furthermore, the intervals for the sampling and publishing of the server can be adjusted. The settings are shown in Fig. 5.22. Particularly important is the publishing interval which is set to the minimum value of 200 ms. Accordingly, each 200 ms, the client can read new data from the controller.



The screenshot shows a configuration window titled "Subscriptions". It contains three input fields: "Minimum sampling interval:" with a value of 1000 and a unit dropdown set to "ms"; "Minimum publishing interval:" with a value of 200 and a unit dropdown set to "ms"; and "Max. number of monitored items:" with a value of 2000.

Figure 5.22: OPC UA subscriptions

In the last step, a runtime license needs to be selected for the execution of the server. The *SIMATIC OPC UA S7-1500 medium* license is required for the used controller (see Fig. 5.23).



The screenshot shows a configuration window titled "OPC UA" with a sub-section "Runtime licenses". It contains two dropdown menus: "Type of required license:" and "Type of purchased license:", both set to "SIMATIC OPC UA S7-1500 medium".

Figure 5.23: Selection of the runtime license

For a uniform interface to access the plant data, an information model is created in SiOME and imported into the TIA portal, enabling MATLAB to access the same data being available in the Machine Advisor. Since all collected data has to be available, the creation of the information model is described in Chapter 6.

5.5.2 Creation of an OPC UA Client

MATLAB provides OPC UA client objects for the connection to an OPC UA Server, allowing existing functions to be used whilst programming. The establishment of the connection can

thus be realized in a simple form. Listing 5.1 shows the creation of an OPC UA client and the connection to the server.

```
1 %Query for accessible OPC UA servers
2 serverInfo = opcuaserverinfo('192.168.2.161');
3 %Construct an OPC UA Client object
4 opcClient = opcua(serverInfo);
5
6 connect(opcClient); %Connect to the server
```

Listing 5.1: Creation of an OPC UA client

In the second line, the information of the available OPC UA servers is retrieved and stored in the variable *serverInfo*. The variable contains the hostname, the port for communication and the name of the OPC UA server. Based on this information, an OPC UA client associated with the referenced server is created in the fourth line. By using the *connect* command the connection to the specified server is established, and the client can access the contained nodes with appropriate functions.

6 Program Design and Implementation

In this chapter, software development is explained. Starting with data collection, the enhancement of the control software in the TIA Portal and KUKA.WorkVisual is described. Subsequently, the data monitoring and alarm management of the Machine Advisor is established. Finally, MATLAB is used for a more detailed analysis of the data.

6.1 Data Collection

The data collection forms the basis for the beneficial monitoring of a plant. Resulting data has to provide as much information about the current status as possible. Based on the previously configured communication interfaces, this section explains the provision of data from the individual IMS stations and the KUKA robot.

6.1.1 Selection of IMS Stations Data

Before programming the data collection, the information to be collected from all IMS stations needs to be defined. The available information is selected according to the component description in Section 3.1.2 and is documented in Tab. 6.1.

Table 6.1: IMS stations data

Name	Data type	Description
current	Real	Analog value of the current
voltage	Real	Analog value of the voltage
power	Real	Calculated value of the power
motorOperatingHours	LTime	Operating time of the motor
countSenBeltLeft	DInt	Number of activations: Sensor belt left
countSenBeltRight	DInt	Number of activations: Sensor belt right
countMotorIsActive	DInt	Number of activations: Belt motor
countCylStopper	DInt	Number of activations: Stopper cylinder
countCylSort	DInt	Number of activations: Sort cylinder

Table 6.1: IMS stations data

Name	Data type	Description
countSenStopperTop	DInt	Number of activations: Sensor stopper Top
countSenMagazine	DInt	Number of activations: Sensor magazine
countCylPressing	DInt	Number of activations: Pressing cylinder
countSenPressCylNotActuated	DInt	Number of activations: Sensor pressing cylinder not actuated
countSenPressCylActuated	DInt	Number of activations: Sensor pressing cylinder actuated
countSenOptBottom	DInt	Number of activations: Optical sensor bottom part
countSenOptTop	DInt	Number of activations: Optical sensor top part
countSenInd	DInt	Number of activations: Inductive sensor
countSenKap	DInt	Number of activations: Capacitive sensor
countCylSwivelTable	DInt	Number of activations: Swivel table cylinder
countCylLift	DInt	Number of activations: Lift-cylinder
countVacuumValve	DInt	Number of activations: Vacuum Valve
countSenSwivelTable0	DInt	Number of activations: Sensor swivel table 0 degree
countSenSwivelTable90	DInt	Number of activations: Sensor swivel table 90 degree
countSenVacuumMonitoring	DInt	Number of activations: Sensor vacuum monitoring
countSwitchLiftTop	DInt	Number of activations: Switch lift-cylinder top

The monitoring of the plant consists of signals from the binary sensors and the power consumption measurement. Not all entries of Tab. 6.1 are available for each station, as the sensors are different.

Since the time course of the binary sensor signals is uninteresting for the condition of the plant, only the number of activations of the respective sensor is counted. The same applies to the actuation of the pneumatic cylinders and motors, whereby the motor running time is additionally recorded.

The measurement of the power consumption consists of analog values of the current and the voltage as well as the calculated power. These variables are available as floating point numbers for each station.

6.1.2 Introduction of User-Defined Data Types

In order to create a uniform interface for data transfer, the two user-defined data types *DATA_TO_MASTER* and *KUKADATA_TO_MASTER* are introduced. Especially for the IMS stations, this can be used beneficially, as all controllers can access the same data type and thus the information is uniformly known.

The data types contain all variables being essential for the data collection and have to be sent to the *PLC_1*. Since the transfer data has been determined for the stations and the robot, a tabular representation is omitted. Only one change was made to the data of the IMS stations. In contrast to the S7-1500 controller, the S7-1200 controller does not support the data type *LTime* which is intended to provide a sufficiently large time window for recording the motor runtime. For this reason, the data type *DATA_TO_MASTER* does not contain the variable *motorOperatingHours* but the boolean variable *motorIsActive*, allowing the S7-1500 controller to record the motor runtime. Both data types can be viewed in the appendix of this thesis.

The user-defined data types have to be added to the variable lists of the respective controller to use them for the transfer of information. As an example, the interpretation of the incoming data is explained for the *PLC_1*.

PLC tags						
	Name	Tag table	Data type	Address	Retain	Accessible from HMI/OPC UA
67	DATA_IMS1	Standard-...	*DATA_TO_MASTER*	%I500.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
68	DATA_IMS3a	Standard-...	*DATA_TO_MASTER*	%I600.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
69	DATA_IMS3b	Standard-...	*DATA_TO_MASTER*	%I700.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
70	DATA_IMS4a	Standard-...	*DATA_TO_MASTER*	%I800.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
71	DATA_IMS4b	Standard-...	*DATA_TO_MASTER*	%I900.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
72	DATA_IMS5a	Standard-...	*DATA_TO_MASTER*	%I1000.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
73	DATA_IMS5b	Standard-...	*DATA_TO_MASTER*	%I1100.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
74	DATA_IMS6	Standard-...	*DATA_TO_MASTER*	%I1200.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
75	DATA_IMS7	Standard-...	*DATA_TO_MASTER*	%I1300.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
76	DATA_KUKA	Kuka	*KUKADATA_TO_MASTER*	%I155.0	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 6.1: User-defined data type in the PLC tags

According to the address ranges assigned in the hardware configuration, the *PLC_1* receives all data as virtual inputs. By applying the user-defined data types in the variable list shown in Fig. 6.1, all virtual inputs can be directly assigned to the variables in the custom data

type. In this process, it is crucial to ensure that the sent data has the same order as the interpreting data type. For the stations, this is ensured by the fact that both the I-Devices and the IO-Controller use the same data type. When communicating with the KUKA robot, it has to be ensured by the configuration of the fieldbus variables.

6.1.3 Program Extension of IMS Stations

The programs, running on the individual controllers of the stations, contain information about all process operations. In this way, each controller is able to fulfill all functionalities, allowing the stations to be easily exchanged without having to change the conveyor belt. The MES system, containing the plant layout configuration, transmits the functionality to perform to each controller.

Consequently, the data collection has to be able to recognize the station and the available sensors and actuators. For this purpose, the identifier *"BELT".IMS* is used, containing a number corresponding to the function of the module. A breakdown of the numbers can be seen in Tab. 6.2.

Table 6.2: *"BELT".IMS* identifier

Identifier	Station
0	IMS1
3	IMS3a, IMS3b
4	IMS4a, IMS4b
5	IMS5a, IMS5b
6	IMS6
7	IMS7

The now known available sensors and actuators can be transferred to the function block for data collection. A module with a uniform interface is programmed to avoid each station having to use its individual module for data collection. The module is included several times in the program code and is linked to the corresponding signals for each station. The identifier then decides which block has to be executed. Fig. 6.2 shows the function block for recording data from IMS7.

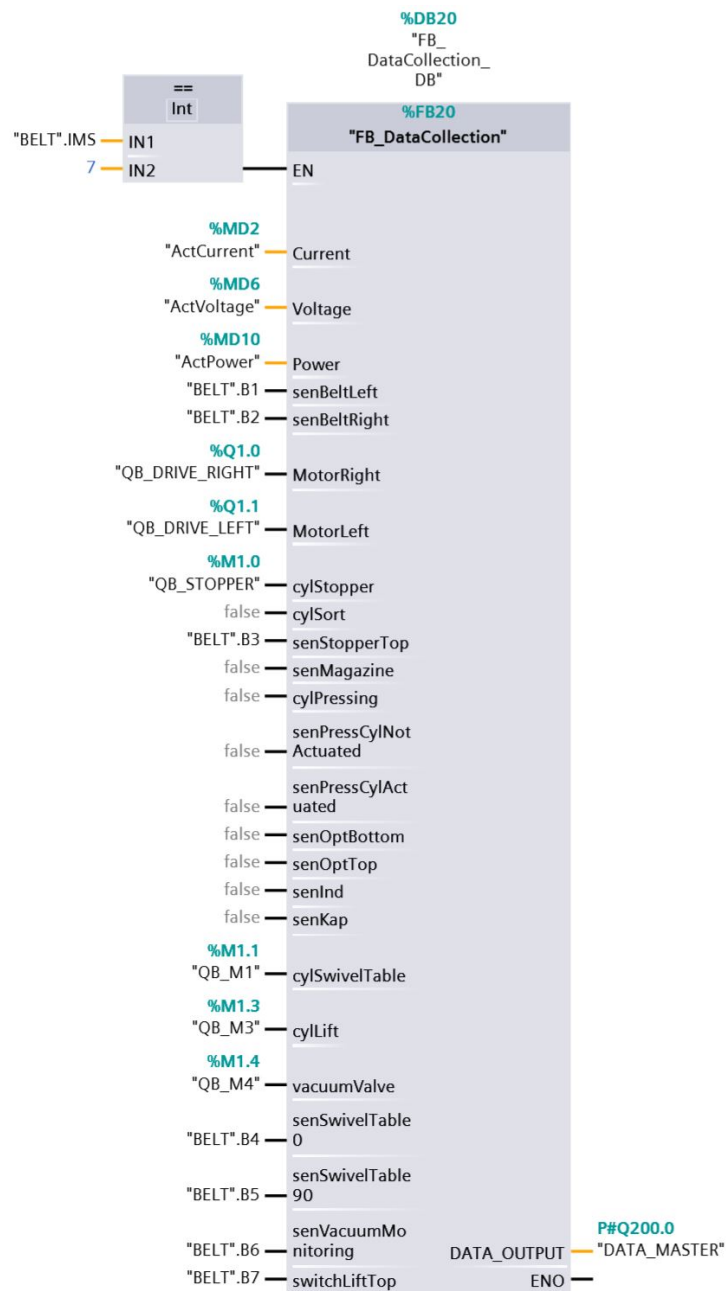


Figure 6.2: Data collection function block

The input *EN* enables the execution of the block and is activated when the identifier is equal to seven. All other input parameters are linked to the corresponding signals of the sensors and actuators of the station. Signals that are not available are set to false to avoid any effect

on the data collection. The output, summarizing all relevant data in the user-defined data type, is forwarded directly to the output address for communication to *PLC_1*.

The structured control language (SCL) is used to program the data collection within the function block. The incoming signals are either transmitted directly to the output signal or passed to an incremental counter. Listing 6.1 shows the processing of the signals provided by each station, including the current, voltage, and power values as well as the counters for the conveyor belt sensors. The boolean variable *motorIsActive* is set when the motor is activated.

```
1 // Data collection for all devices
2 // Allocation of measurement-values
3 #DATA_OUTPUT.Current := #Current;
4 #DATA_OUTPUT.Voltage := #Voltage;
5 #DATA_OUTPUT.Power   := #Power;
6
7 // Counters
8 "senBeltLeftCounter".CTU(CU:= #senBeltLeft,
9     R:= FALSE,
10    PV:= 0,
11    CV=> #DATA_OUTPUT.countSenBeltLeft);
12
13 "senBeltRight".CTU(CU := #senBeltRight,
14    R := FALSE,
15    PV := 0,
16    CV => #DATA_OUTPUT.countSenBeltRight);
17
18 // Check and count if motor is active
19 #DATA_OUTPUT.motorIsActive := #MotorRight OR #MotorLeft;
20
21 "motorIsActiveCounter".CTU(CU:=#MotorRight OR #MotorLeft,
22    R:=FALSE,
23    PV:=0,
24    CV=>#DATA_OUTPUT.countMotorIsActive);
```

Listing 6.1: Data collection function block

The remaining input signals are also assigned to incremental counters to provide the required data. If the sensor is not available, the input value *false* will cause the counter to remain at zero.

6.1.4 Program Extension of KUKA Robot

In the hardware configuration, outputs have already been linked to the fieldbus. Before these outputs can be used in the program code, signal declarations have to be created. For this purpose, KUKA provides an area for the declaration of user variables in the configuration file *config.dat* of the robot. Listing 6.2 shows the declaration of the current signals for the six axes of the robot.

```
1 ; Userdefined Variables
2 SIGNAL CURR_ACT_Axis1 $OUT[2000] TO $OUT[2031]
3 SIGNAL CURR_ACT_Axis2 $OUT[2032] TO $OUT[2063]
4 SIGNAL CURR_ACT_Axis3 $OUT[2064] TO $OUT[2095]
5 SIGNAL CURR_ACT_Axis4 $OUT[2096] TO $OUT[2127]
6 SIGNAL CURR_ACT_Axis5 $OUT[2128] TO $OUT[2159]
7 SIGNAL CURR_ACT_Axis6 $OUT[2160] TO $OUT[2191]
```

Listing 6.2: Signal declarations in the *config.dat*

Using this signal declaration, the defined variables are mapped to the corresponding output address ranges. Once this has been done for all variables, values can be assigned in the program code. The variables of the signal declaration are always interpreted as INT.

The KRC4 controller is running two tasks. On the one hand, the robot interpreter, executing motion programs for the robot, and on the other hand, the controller interpreter, executing a parallel control program. Latter ought to be used for the assignment of the variables, since it enables an independent monitoring of the parameters. Listing 6.3 shows the modification of the controller interpreter for assigning the system variables to the declared variables.

```
1 ;FOLD USER PLC
2 ;Make your modifications here
3 CURR_ACT_Axis1 = $CURR_ACT[1]*1000 ; %
4 CURR_ACT_Axis2 = $CURR_ACT[2]*1000 ; %
5 CURR_ACT_Axis3 = $CURR_ACT[3]*1000 ; %
6 CURR_ACT_Axis4 = $CURR_ACT[4]*1000 ; %
7 CURR_ACT_Axis5 = $CURR_ACT[5]*1000 ; %
8 CURR_ACT_Axis6 = $CURR_ACT[6]*1000 ; %
9 MOT_TEMP_Axis1 = $MOT_TEMP[1] ; kelvin
10 MOT_TEMP_Axis2 = $MOT_TEMP[2] ; kelvin
11 MOT_TEMP_Axis3 = $MOT_TEMP[3] ; kelvin
```

```
12 MOT_TEMP_Axis4 = $MOT_TEMP[4] ; kelvin
13 MOT_TEMP_Axis5 = $MOT_TEMP[5] ; kelvin
14 MOT_TEMP_Axis6 = $MOT_TEMP[6] ; kelvin
15 TORQUE_ACT_Axis1 = $TORQUE_AXIS_ACT[1]*1000 ; Nm
16 TORQUE_ACT_Axis2 = $TORQUE_AXIS_ACT[2]*1000 ; Nm
17 TORQUE_ACT_Axis3 = $TORQUE_AXIS_ACT[3]*1000 ; Nm
18 TORQUE_ACT_Axis4 = $TORQUE_AXIS_ACT[4]*1000 ; Nm
19 TORQUE_ACT_Axis5 = $TORQUE_AXIS_ACT[5]*1000 ; Nm
20 TORQUE_ACT_Axis6 = $TORQUE_AXIS_ACT[6]*1000 ; Nm
21 VEL_ACT_Axis1 = $VEL_AXIS_ACT[1]*1000 ; %
22 VEL_ACT_Axis2 = $VEL_AXIS_ACT[2]*1000 ; %
23 VEL_ACT_Axis3 = $VEL_AXIS_ACT[3]*1000 ; %
24 VEL_ACT_Axis4 = $VEL_AXIS_ACT[4]*1000 ; %
25 VEL_ACT_Axis5 = $VEL_AXIS_ACT[5]*1000 ; %
26 VEL_ACT_Axis6 = $VEL_AXIS_ACT[6]*1000 ; %
27 ;ENDFOLD (USER PLC)
```

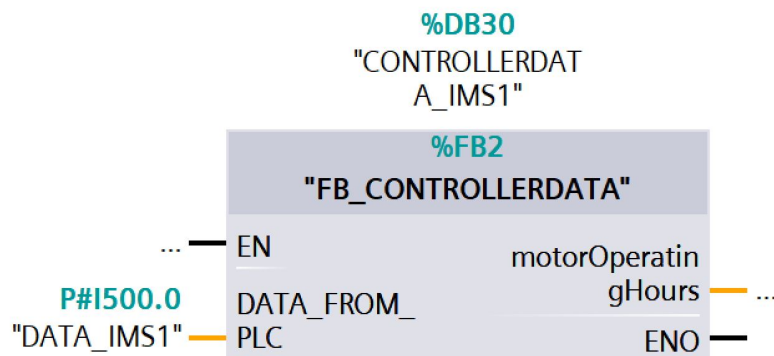
Listing 6.3: Assignment of variables in *sps.sub*

Since values for the current, torque and motor speed of each axis are floating-point numbers and the conversion to INT would mean a loss of information, they are multiplied by the factor 1000, shifting the decimal point, and previous decimal places can be transferred. When interpreting the values in the *PLC_1*, it is necessary to divide them by the same factor.

6.1.5 Program Extension of PLC_1

The control program of the *PLC_1* has to be extended to allow to read the data from the individual controllers as well as from the robot and to write them into the data block for communication with the Machine Advisor and MATLAB.

First, the data of the IMS stations ought to be read. For this purpose, a function block is created, expecting the user-defined data type *DATA_TO_MASTER* as input. When calling a function block, an individual instance data block is created, allowing to call the same function block for each station and supply it with the corresponding inputs. The resulting instance data blocks allow access to all data within the *PLC_1* program. The call of the block for the station IMS1 is shown in Fig. 6.3.

Figure 6.3: Function block *FB_CONTROLLERDATA*

Besides, the motor running time has to be recorded. The instruction time accumulator is being used, enabling the recording of time in a given period *PT*. As soon as *IN* turns to a logical one, the time recording is started. The parameterization of the counter is shown in Listing 6.4.

```

1 // Measurement of the Operating Hours of the Motor
2 #motorOperatingHoursTimer(
3     IN := #DATA_FROM_PLC.motorIsActive,
4     R := FALSE,
5     PT := LT#106750d,
6     ET => #TEMPmotorOperatingHours);
7
8 #motorOperatingHours := LTIME_TO_LINT(
    TEMPmotorOperatingHours) / (60 * (10 ** 9));

```

Listing 6.4: Recording of the motor runtime

The input parameter for the time accumulator is the signal monitoring the motor activity. The period *PT* is set to the largest possible value of the data type *LTime* to allow to record over a long period. Yet, the Machine Advisor does not support this data type. Thus, the output value has to be converted first. The function *LTIME_TO_LINT* converts the time signal into a 64 bit INT, representing the time in nanoseconds. For a more suitable representation in the Machine Advisor, the time is converted to minutes in line number eight.

Similarly, the data from the KUKA robot is processed in the *PLC_1*. Again, a function block is created for the data, storing all information in an instance data block. Within the function block, the variables have to be converted. Listing 6.5 shows the conversion of the data for axis one.

```
1 // Swap the DWord because Kuka handles litte endian
2 // Conversion to Ampere
3 #CURR_ACT_Axis1 := (DINT_TO_REAL(SWAP(#DATA_FROM_KUKA.
   CURR_ACT_Axis1)) / 1000) * (40 / 100);
4 // Conversion from Kelvin to Celsius
5 #MOT_TEMP_Axis1 := SWAP(#DATA_FROM_KUKA.MOT_TEMP_Axis1)
   -272;
6 // Conversion to Newton Meters
7 #TORQUE_ACT_Axis1 := DINT_TO_REAL(SWAP(#DATA_FROM_KUKA.
   TORQUE_ACT_Axis1))/1000;
8 // Conversion to Rounds per Minute
9 #VEL_ACT_Axis1 := (DINT_TO_REAL(SWAP(#DATA_FROM_KUKA.
   VEL_ACT_Axis1))/1000)*(6000/100);
```

Listing 6.5: Conversion of robot data

One change required for all data from the robot is to adjust the order of the bytes. KUKA uses the little-endian format, where the least significant byte (LSB) is stored at the lowest memory location, while Siemens uses the big-endian format with the most significant byte (MSB) at the lowest memory location. The *SWAP* function performs this swap operation.

The current, motor temperature and torque have to be divided by 1000 to represent the contained information as a floating-point number. Additionally, the values have to be converted into their units. The temperature requires only a subtraction for the conversion from Celsius to Kelvin. For the current and motor speed, given as a percentage of their maximum, the nominal current and speed of the motor need to be determined. These can be found in the motor files, located in the project directory *Config\User\Common\Mada\NGAxis\A1....6.xml*. The current for axis A1 is 40 A, and the speed for the motor is 6000 rpm.

The collected data is now available in several data blocks in the *PLC_1* (see Fig. 6.4). Finally, the data has to be transferred to the data block *ModbusServerData* which is accessed by the Machine Advisor and MATLAB.

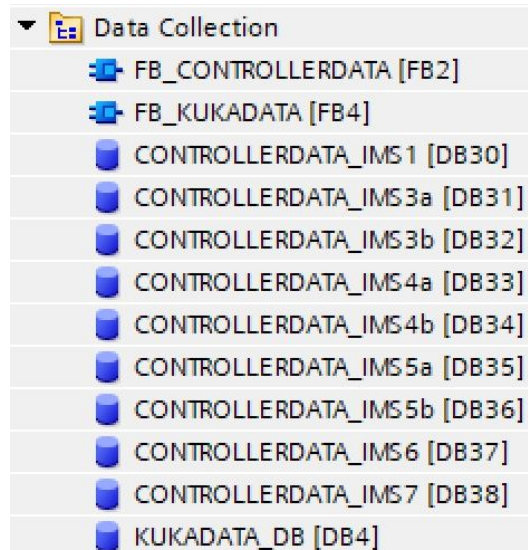


Figure 6.4: Data block structure in TIA Portal

All variables relevant for data collection have to be defined in the data block *ModbusServerData* before assignment. The complete table can be found in the appendix of this thesis. If all variables are defined, they will be assigned in the function *TransferControllerData*. The access to the variables is shown exemplarily for the power of the motor in Listing 6.6.

```
1 // Allocation of Motor Power for all Stations
2 "ModbusServerData".MotorPowerIMS1 :=
    "CONTROLLERDATA_IMS1".DATA_FROM_PLC.power;
3 "ModbusServerData".MotorPowerIMS3a :=
    "CONTROLLERDATA_IMS3a".DATA_FROM_PLC.power;
4 "ModbusServerData".MotorPowerIMS3b :=
    "CONTROLLERDATA_IMS3b".DATA_FROM_PLC.power;
5 "ModbusServerData".MotorPowerIMS4a :=
    "CONTROLLERDATA_IMS4a".DATA_FROM_PLC.power;
6 "ModbusServerData".MotorPowerIMS4b :=
    "CONTROLLERDATA_IMS4b".DATA_FROM_PLC.power;
7 "ModbusServerData".MotorPowerIMS5a :=
    "CONTROLLERDATA_IMS5a".DATA_FROM_PLC.power;
```



```

8  "ModbusServerData".MotorPowerIMS5b  :=
    "CONTROLLERDATA_IMS5b".DATA_FROM_PLC.power;
9  "ModbusServerData".MotorPowerIMS6   :=
    "CONTROLLERDATA_IMS6".DATA_FROM_PLC.power;
10 "ModbusServerData".MotorPowerIMS7   :=
    "CONTROLLERDATA_IMS7".DATA_FROM_PLC.power;

```

Listing 6.6: Function *TransferControllerData*

Within the function only simple assignments are necessary, writing the information from the data blocks of the individual stations or the robot into the data block *ModbusServerData*. By calling it in *OB1*, all variables are updated cyclically and can be captured by the analysis software.

6.2 Data Analysis in Machine Advisor

In order to display the data in the cloud and generate alarms based on this data, further configuration is necessary in the Machine Advisor and the ComX Box.

6.2.1 Interpretation of Data

The parameter template, introduced in Section 5.4, has to be completed for communication to the cloud platform. Therefore, the variables from the *ModbusServerData* data block are written into the Excel table for the parameter template. It is crucial to keep the same order, otherwise the data of the *PLC_1* will be mapped incorrectly.

The Machine Advisor provides predefined monitoring types which are identified by the corresponding indicator in the parameter template. The used data types are listed in Tab. 6.3.

Table 6.3: Monitoring types

Monitoring Type	Description
CUR	Current
VOL	Voltage
POW	Power
TIS	Number of repeats
TIM	Time
TEM	Temperature

Furthermore, the smallest possible size of a variable, specified in the parameter template, is a Word. Accordingly, the size for a 32 byte variable has to be two and for a 64 byte variable four.

The transmission type offers three options: a single transmission at the beginning of the connection, a transmission at a certain frequency or a transmission when the variable has changed. A fixed transmission interval is best suited for the transfer of the plant data, therefore a R is entered in the parameter template for the transmission type. The transmission frequency is set to the minimum value of five seconds.

In order to update a variable in the cloud, it has to be indicated by a Y in the upload column.

The final table has to be uploaded to the platform management of the Machine Advisor to overwrite the empty table. Once the new parameter template is activated, the data is transferred to the cloud.

6.2.2 Alarm Management

The alarm management is intended to offer the operator feedback on the status of the plant. The alarms are divided into the three categories low-level, medium-level, and high-level, whereby only a high-level alarm has the option of sending an SMS to the operator.

The configuration is set in the web interface of the ComX Box and is described using the temperature variables of the robot. In the data and alarm management view, all variables are listed, and alarms can be added. Fig. 6.5 shows the configuration window for the motor temperature of axis one.

启用	类型	阈值	级别	描述
<input checked="" type="checkbox"/>	高高报警	50	高	Temperature high
<input checked="" type="checkbox"/>	高报警	45	低	Temperature high warning
<input checked="" type="checkbox"/>	低报警	5	低	Temperature low warning
<input checked="" type="checkbox"/>	低低报警	0	高	Temperature low

死区 延时 秒

Figure 6.5: Alarm settings

The alarm can be set with two upper limits and two lower limits. The robot specification [42] defines a recommended operating temperature between $+5^{\circ}\text{C}$ and $+45^{\circ}\text{C}$. Reaching these limits generates a low-level alarm and if the temperature continues to rise or fall to $+50^{\circ}\text{C}$ or 0°C a high-level alarm will be triggered.

Once the setting has been confirmed, the alarms are automatically synchronized with the cloud. In the data and alarm management view the successful configuration is indicated by a tick in the last column (see Fig. 6.6).

编号	名称	当前值	单位	告警状态	质量	更新时间	启用告警
113	KUKA_MOT_TEMP_Axis1	31		告警	正常	7/31/2019, 12:29:08 PM	√
114	KUKA_MOT_TEMP_Axis2	30		正常	正常	7/31/2019, 12:29:08 PM	√
115	KUKA_MOT_TEMP_Axis3	31		正常	正常	7/31/2019, 12:29:08 PM	√
116	KUKA_MOT_TEMP_Axis4	31		正常	正常	7/31/2019, 12:29:09 PM	√
117	KUKA_MOT_TEMP_Axis5	32		正常	正常	7/31/2019, 12:29:09 PM	√
118	KUKA_MOT_TEMP_Axis6	31		正常	正常	7/31/2019, 12:29:09 PM	√

Figure 6.6: Data and alarm management view

6.3 Data Analysis in MATLAB

The first step in data analysis with MATLAB is to display the received data. Furthermore, a pattern recognition network is used to provide a perspective for the advanced analysis of plant data.

6.3.1 Creation of a server interface with SiOME

An information model is created to enable MATLAB to read the data from the OPC UA server. The SiOME allows to import TIA Portal projects and thus enables an easy selection of the data to be transferred. In the view of the program blocks, the data block *ModbusServerData* can be transferred by drag and drop into the objects directory of the information model (see Fig. 6.7).

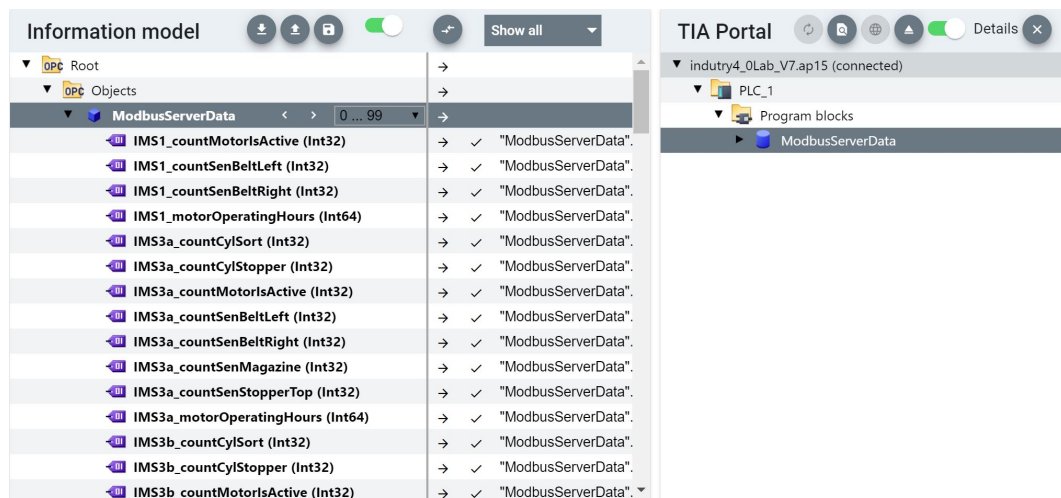


Figure 6.7: Information model for the plant data

As a result, an automatically generated XML file can be exported, mapping the program's internal variables to the server interface. The last step is to add this file to the *PLC_1* program in the OPC UA communication server interface.

6.3.2 Querying Data

With an existing connection to the OPC UA Server, the contained nodes can be read using the predefined function for an OPC UA client object. Listing 6.8 shows the reading of the data in the *ModbusServerData* data block.

```

1 % Recieve OPC UA Node array:
2 % 1:Server, 2:DeviceSet, 3:PLC_1, 4:ModbusServerData
3 nodeList = getNamespace(opcClient);
4 % Recieve the data nodes of the ModbusServerData
5 dataNodes = nodeList(4).Children;

```

Listing 6.7: Reading the OPC UA node array

The variable *dataNodes* contains all entries of the data block in the controller. Using the *readValue()* function, the current value can be read from the OPC UA server.

The focus of the more detailed analysis in Matlab is on the variables of the KUKA robot. Therefore, a variable *kukaProcessing* is transmitted to MATLAB to signal the start of the

robot process to ensure that a fixed timeframe is maintained when the robot data is recorded. The data nodes 80 to 103 contain the collected data for all robot axes and is written in a loop into a cell array (see Listing 6.8).

```
1 for i = 1:numberOfRecordings
2     % Wait for the kuka process to begin
3     while ~(readValue(kukaProcessing))
4     end
5
6     % Collect data
7     stopTime = datetime('now') + seconds(timeDuration);
8     while datetime('now') <= stopTime+seconds(timeInterval
9         )
10        % Start the time measurement
11        tic
12        % Collect the data of the KUKA robot
13        collectedData(count,:) = readValue(dataNodes
14            ((80:103)));
15        % If the elapsed time is smaller than the time
16        intervall pause the loop
17        elapsedTime = toc;
18        if elapsedTime < timeInterval
19            pause(timeInterval - elapsedTime);
20        else % Else create a warning for the operator
21            warning(['Required time to collect the data is
22                greater than the selected time interval. '
23                    ...
24                    'This can lead to data loss.']);
25        end
26        count = count + 1;
27    end
28 end
```

Listing 6.8: Reading the OPC UA node array

Since the data in the OPC UA Server is only updated every 200 milliseconds, the reading of the data is monitored. If the time interval is not reached, the system will wait for the respective time or, if the time interval is exceeded, will issue a warning, as this can lead to data loss.

The resulting cell array contains the recording of a working cycle of the robot and is saved in an Excel table for further evaluation. The variable *numberOfRecordings* determines the number of cycles to be recorded.

6.3.3 Pattern Recognition Network

A pattern recognition network is a neural network in feedforward structure. It can be trained to classify inputs into corresponding classes. For the KUKA robot, the time signals of the current ought to be evaluated to classify the state into one of the five classes regular operation, error axis one, error axis two, error axis three and error axis four.

The first step is to generate data for training the neural network. The method described under 6.3.2 is used to record the working process of the robot. For regular operation, the cycle is recorded 50 times, and for the individual failures, manually induced to operation, 30 cycles are recorded. The function *analyzeData* analyzes this data. It expects the time series of the current as input and returns a set of parameters describing the input signal. The contained parameters can be found in Listing 6.9.

```
1 param = [meanValue, ...
2         medianValue, ...
3         rmsValue, ...
4         variance, ...
5         peak, ...
6         peak2peakValue, ...
7         signalSkewness, ...
8         signalKurtosis, ...
9         crestfactor, ...
10        medianAbsoluteDeviation, ...
11        rangeOfCumulativeSum];
```

Listing 6.9: Output of the *analyzeData* function

Performing this for all training data, each data set will no longer contain the time series for a work cycle, but 44 parameters describing the current signals of axes one, two, three and four in that period. These parameters and the corresponding target class can be used to train the pattern net.

The neural network expects the input values and the corresponding target values in the form of a matrix, with each column representing one training example. In the target matrix, a one in the corresponding row represents the affiliation to this class. The first row stands for regular operation, the second row for an error axis one, the third row for an error axis two, the fourth row for an error axis three, and the fifth row for an error axis four. Listing 6.10 shows the declaration of the input and target matrix.

```
1 %% Parameter for the neural network
2 % Input matrix
3 x = [parameter, errorA1Parameter, errorA2Parameter,
      errorA3Parameter, errorA4Parameter];
4 % Target matrix
5 t = [ones(1,50), zeros(1,30), zeros(1,30), zeros(1,30),
      zeros(1,30);...
6      zeros(1,50), ones(1,30) , zeros(1,30), zeros(1,30),
      zeros(1,30);...
7      zeros(1,50), zeros(1,30), ones(1,30), zeros(1,30),
      zeros(1,30);...
8      zeros(1,50), zeros(1,30), zeros(1,30), ones(1,30),
      zeros(1,30);...
9      zeros(1,50), zeros(1,30), zeros(1,30), zeros(1,30),
      ones(1,30)];
```

Listing 6.10: Parameter for the neural network

The neural network has 44 input values and in the output layer five neurons, each representing one class. The parameterization and the training can be found in Listing 6.11.

```
1 %% Neural Network
2 % Choose a training function
3 trainFcn = 'trainscg'; % Scaled conjugate gradient
      backpropagation.
4 % Set the hidden layer size
5 hiddenLayerSize = 15;
6 % Create a pattern recognition network
7 net = patternnet(hiddenLayerSize, trainFcn);
8 % Choose a performance function
9 net.performFcn = 'crossentropy';
10 net.performParam.regularization = 0.1;
11 net.performParam.normalization = 'none';
12
13 % Setup division of data for training, validation,
      testing
14 net.divideParam.trainRatio = 70/100;
15 net.divideParam.valRatio = 10/100;
16 net.divideParam.testRatio = 20/100;
```

```
17  
18 % Train the network  
19 [net,tr] = train(net,x,t);
```

Listing 6.11: Creation of the pattern recognition network

Since this application is only intended to provide a perspective for the possible evaluation of the data, the parameterization of the neural network is not fully described. More detailed information about the parameterization can be found in [43].

The training function and the size of the hidden layer have to be defined to create a pattern net. The former is left with the standard function when using pattern recognition networks in MATLAB. Generally, when choosing the size of the hidden layer, a higher number of neurons is able to solve more complicated problems but tends to over-fit the data. At this point, a size of 15 has proved to be suitable by empirical determination. The input data is divided into three categories. The largest portion is the network training rate, followed by the test data. The validation data is completely independent data that is also used to test the network.

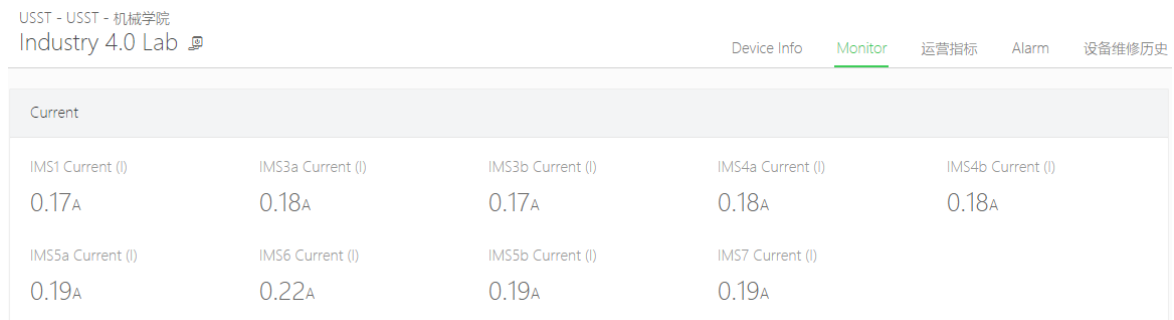
7 Functional Test

In the functional test, the developed software is tested relating to the requirements defined in Chapter 4, and the results are documented. The first step shows the data monitoring in the Machine Advisor and documents an error case of the system from the occurrence to the acknowledgment. Secondly, the data query is demonstrated in MATLAB, and the function of the pattern recognition network is tested.

7.1 Data Monitoring in Machine Advisor

The availability of data is a central requirement. The Machine Advisor provides monitoring of all transmitted variables, historical signal traces, and dashboards, displaying selected data. The different display formats are presented below.

The asset view of the Machine Advisor offers an overview of the plant's real-time data. Each variable can be combined into individual groups and display its current value, refreshed every five seconds. Fig. 7.1 shows the arrangement of the current for all stations.



The screenshot shows a web interface for 'Industry 4.0 Lab' with a navigation menu including 'Device Info', 'Monitor', '运营指标', 'Alarm', and '设备维修历史'. The 'Monitor' tab is active, displaying a table of current values for various stations.

Current				
IMS1 Current (I)	IMS3a Current (I)	IMS3b Current (I)	IMS4a Current (I)	IMS4b Current (I)
0.17A	0.18A	0.17A	0.18A	0.18A
IMS5a Current (I)	IMS6 Current (I)	IMS5b Current (I)	IMS7 Current (I)	
0.19A	0.22A	0.19A	0.19A	

Figure 7.1: Data monitoring view

By clicking on a variable, the historical data can be retrieved. The data is displayed in a graph that can be extended by additional variables. The user can vary the displayed period. Figure 7.2 shows the power consumption of all stations for one production day. Individual data points, available at 15-minute intervals, can be selected and displayed in the graph.

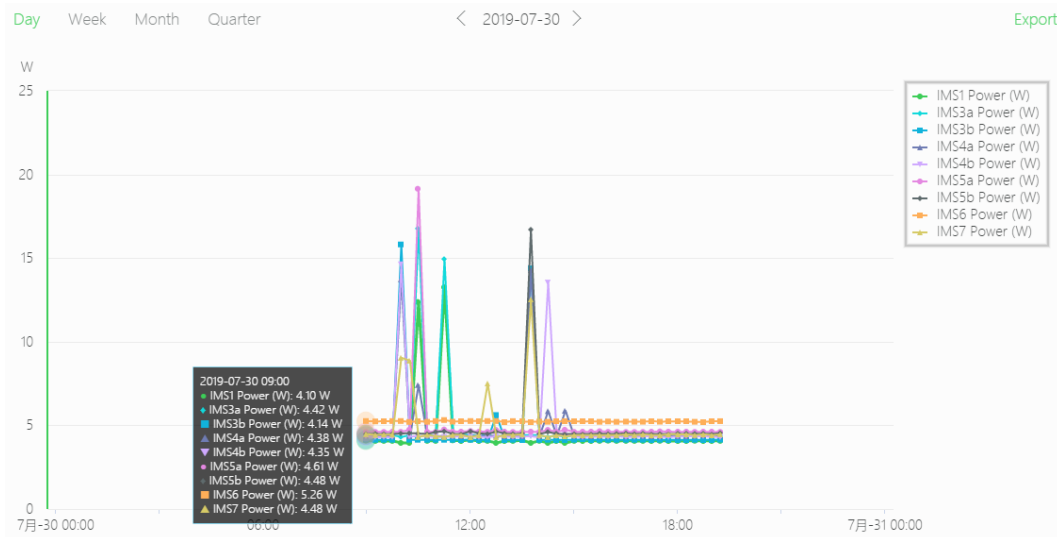


Figure 7.2: Historical data view

The historical data can also be exported as an Excel spreadsheet and used for further analysis. When displaying the historical data, the course of the signals appears almost arbitrary due to the 15-minute interval. Conspicuous features in the signal behaviour are difficult to detect with these data.

The last variant of the display options is the creation of a dashboard. Figure 7.3 shows the dashboard for selected data of the IMS1 station. The right half shows the current and voltage as well as the resulting power consumption over one day. The graphs on the left show the counter values for motor activation (top) and the activation of the conveyor belt's end position sensors.

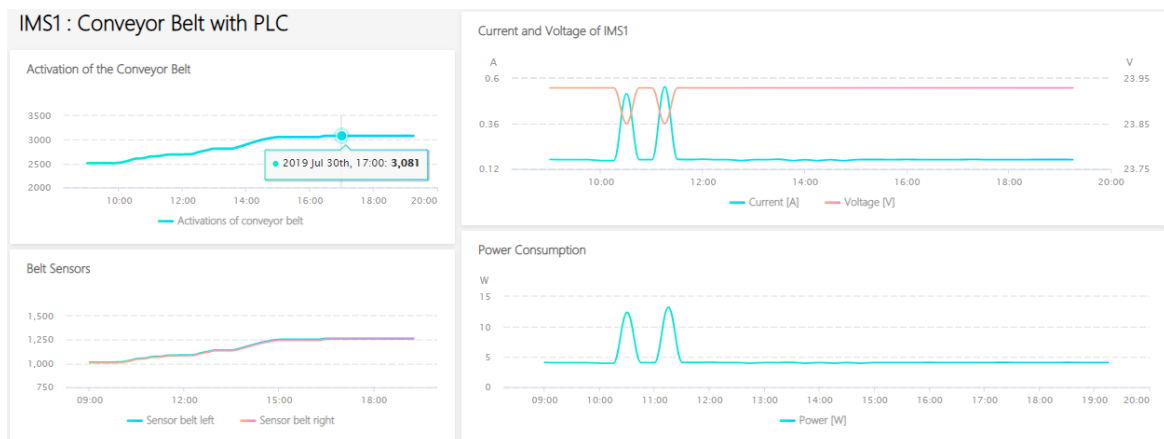


Figure 7.3: Dashboard for IMS1

Besides the display of the data, the supervision of these is of great importance for the achievement of the requirements. In the following, the occurrence and acknowledgement of an alarm as well as the occurrence of a high-level alarm is documented and described.

The upper limit of the motor temperature of axis one is set to a value of 30°C to generate an alarm. The error message appearing in the asset view is shown in Figure 7.4. In addition to current pending messages, past alarms can also be viewed.

Level	Time	Device	Type	Point	Actual Val...	Set Value	Operation
Low	2019-07-31 12:16:29	Industry 4.0 Lab	超上限	KUKA_MOT_TEMP_Axis1	31	30	Create 申请售后 Detail

Figure 7.4: Low-level Error occurrence

The error message contains information about the time of occurrence, the affected system, which variable causes the alarm, as well as the specified limit and current value.

With this information, the operator of the plant can already narrow down the error and react accordingly. If the alarm is classified as false or not relevant, it may be acknowledged under *Detail*. Otherwise, an alarm ticket can be created, containing all information about the error and the task to be performed (see Fig. 7.5).



Figure 7.5: Creation of an alarm ticket

Simultaneously, the ticket executor receives a notification in the form of an SMS with basic information about the alarm (see Fig. 7.6). The complete alarm ticket can be viewed in the app or on the website.

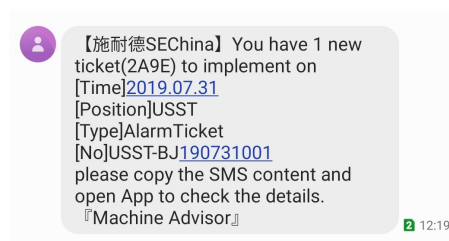


Figure 7.6: Notification of an open alarm ticket

As soon as the ticket is reported as finished, it can be seen in the *Detail* view shown in Fig. 7.7. The alarm can be acknowledged with *Release Alarm* and is then written to the archive.



Figure 7.7: Acknowledgment of an alarm

Apart from alarm tickets, tickets for one-time or cyclic maintenance can also be created and planned. Similarly, the ticket executor is informed about the work to be done and can finish the ticket at the end of the work.

All tickets are automatically entered in a calendar, which simplifies the planning of new maintenance tasks.

If a high-level alarm occurs in the system, a notification will automatically be sent to the contacts specified in the production line. The SMS, shown in Fig. 7.8, contains information about the error type and the affected plant.



Figure 7.8: Notification of a high-level alarm

Thus the implementation of the analysis in the Machine Advisor could meet the requirements 1-6 and 9 in Tab. 4.1.

7.2 Evaluation in MATLAB

The connection to MATLAB allows the recording of all data recorded by the *PLC_1*. In comparison to the Machine Advisor, the capability for more detailed analysis stands out. For this reason, this section will evaluate the error detection performed with the pattern recognition network.

The working process of the robot is recorded, including the transport of a workpiece to the inspection station and back to the IMS5b station. The current during the movement of axes 1-4 is recorded for 50 cycles in order to have regular operation data for the training of the neural network. Fig. 7.9 shows the current course of axis one. The small deviations can result from non-identical sampling times as well as external influences and the load on the robot.

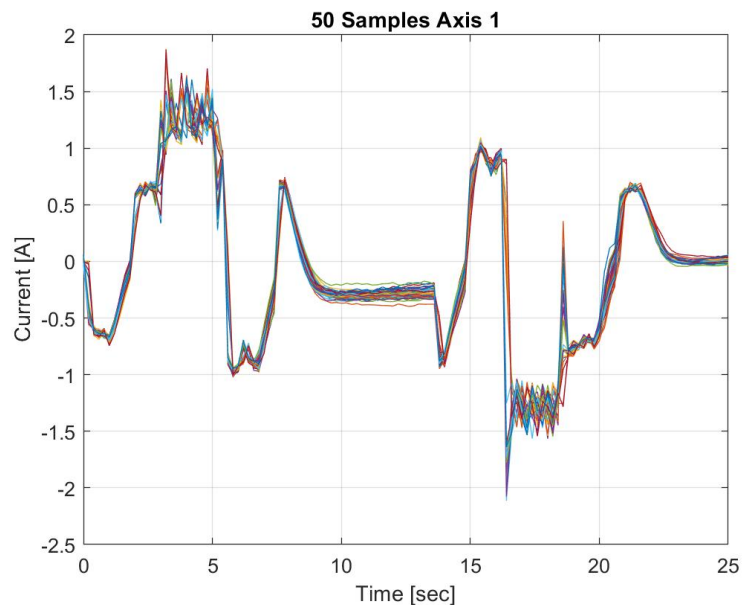


Figure 7.9: Regular samples of axis one

Errors are induced manually in the system to provide error data for the training. For each axis, 30 erroneous data sets are created. The current of the data set for axis one is shown in Fig. 7.10.

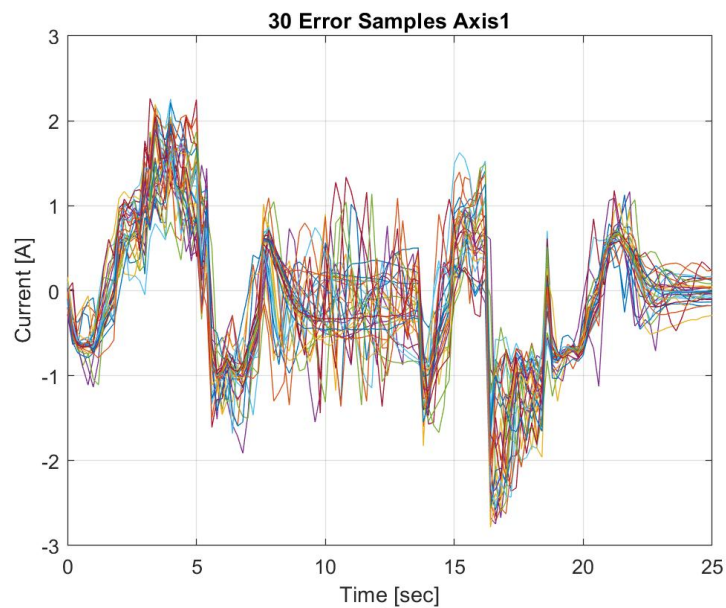


Figure 7.10: Error samples of axis one

The visual deviations of the two graphs are also reflected in the parameters given as input values to the pattern recognition network. Tab. 7.1 shows the maximum and minimum parameter values of all regular and error samples.

Table 7.1: Parameter comparison

Parameter	Regular Samples		Error Samples	
	max	min	max	min
mean	-0,0676	-0,0986	0,0775	-0,2322
median	-0,1762	-0,3102	0,0924	-0,3524
rms	0,7381	0,6990	0,9145	0,7296
variance	0,5379	0,4850	0,8374	0,5178
peak	1,8772	1,4596	2,2628	1,5980
peak to peak	3,7112	3,0400	4,9092	3,3968
skewness	0,4328	0,1264	0,5874	-0,5230
kurtosis	2,7280	2,4503	4,5153	2,2709
crestfactor	2,9026	2,1893	3,5057	2,3110
median absolute deviation	0,5900	0,5489	0,7258	0,5232
range of cumulative sum	31,5444	26,9832	44,3576	20,5492

The structure of the trained net is shown in Fig. 7.11. For each training example, the 44 input parameters are assigned to one of the five classes of the output layer. The hidden layer consists of 15 neurons.

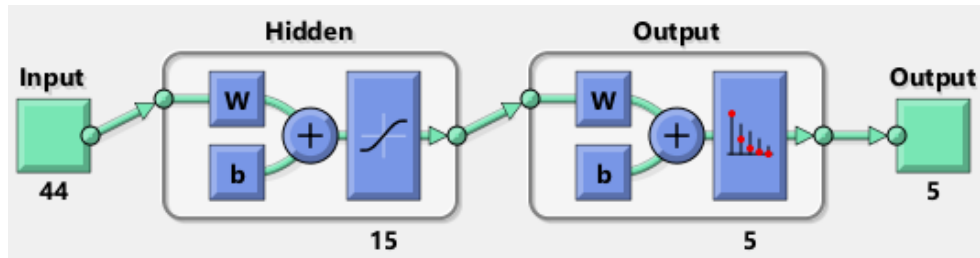


Figure 7.11: Network structure

When training the neural network, the error in the training data and the test data decreases as desired. The validation data ensures that the neural network is not only optimized for the test data. Before over-adaptation occurs, the neural network stops training and selects the best performance.

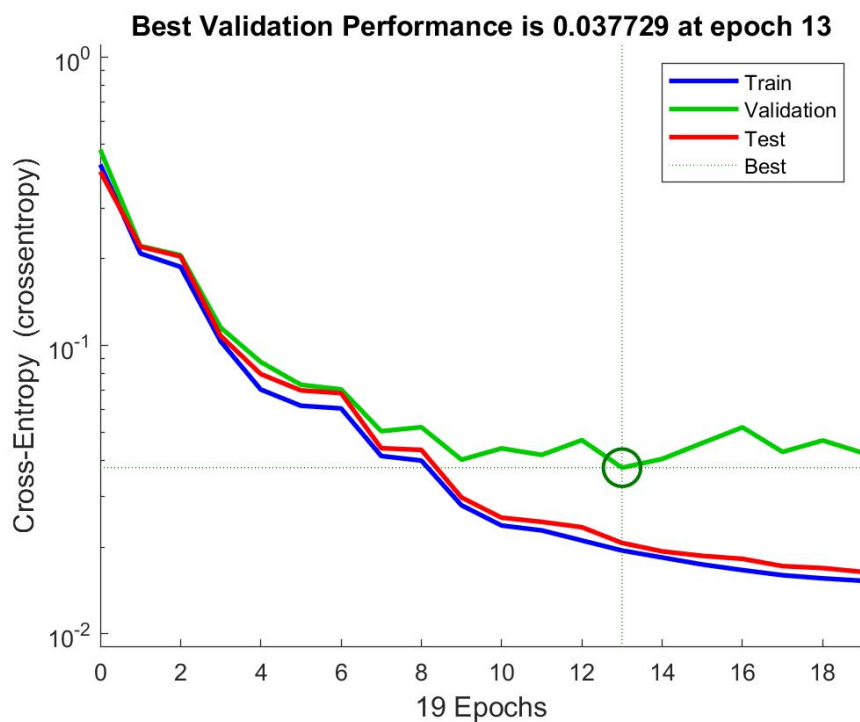


Figure 7.12: Performance

A confusion matrix (see Fig. 7.13) shows the classification of the input data according to the output classes and indicates the accuracy as a percentage. The main diagonal contains the correctly classified data sets and elements on secondary diagonals are incorrectly classified. Both the training and the test confusion matrix classify the data sets into the correct classes. Only during the validation, a training example is wrongly classified.

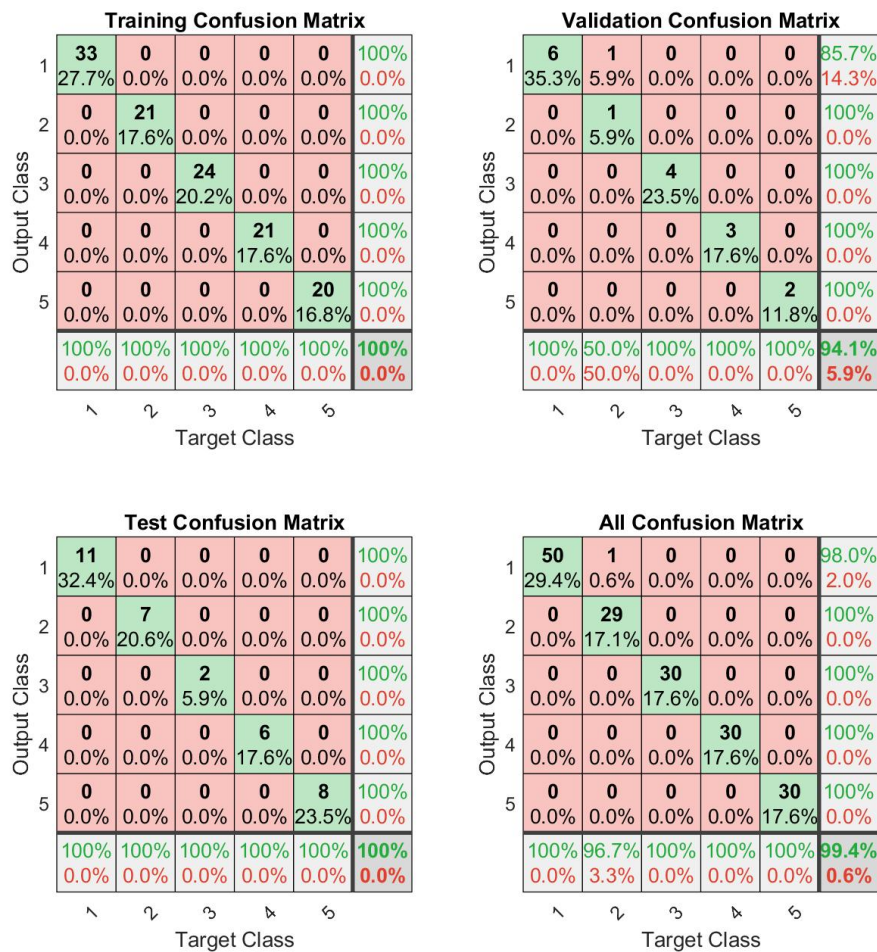


Figure 7.13: Confusion matrix

Thus, the classification of the validation data has a high accuracy rate of 94,1 % and the neural network is well suited for the classification of the input data into the error categories.

One possible reason for this very high rate of accuracy could be the separability of the error patterns. Due to the manual induction of the errors, the other axes are only slightly affected

(see Fig. 7.14). In the case of an error occurring during production, it cannot be assumed that this error will only affect one axis of the robot.

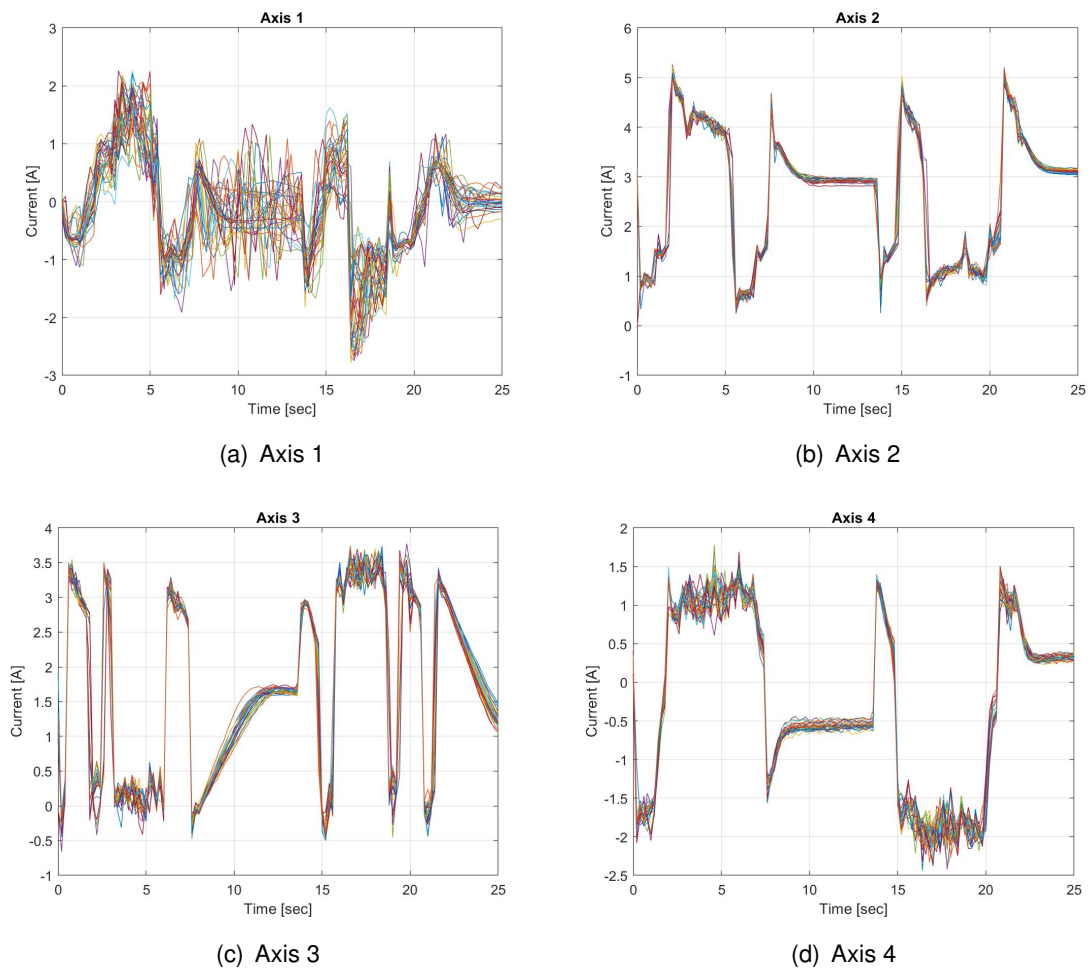


Figure 7.14: Influence of the error of axis one

This result is sufficient for a perspective on a possible analysis of the plant data, but it should be pointed out that for a reliable statement about the performance of the neural network significantly larger data sets are necessary.

The implementation in MATLAB meets the requirements 1, 6 and 7 from Tab. 4.1.

8 Conclusion

In the preceding thesis, an extension for an Industry 4.0 production plant located at SHC has been developed. Special attention is attached to the implementation of predictive maintenance concepts.

8.1 Summary

The first part establishes the necessary communication between the participants, including the communication between the individual stations and the robot with the main controller, as well as the connection to the cloud platform and the setup of an OPC UA server for data exchange with MATLAB. A precise analysis of the plant forms the basis for the identification of the collectible information. In the first step of the implementation, the data of the stations and the robot were collected and stored centrally on the main controller. The second step was to transfer the data to the Machine Advisor and MATLAB. The Machine Advisor is responsible for displaying the real-time as well as historical data and for the alarm management of the plant, whereby variables are monitored by setting simple thresholds. MATLAB allows time series to be displayed at a higher sampling rate and allows more detailed analysis. In this thesis, a pattern recognition network for classifying fault data was implemented as a perspective. The generated software was transferred to the model plant and tested according to the defined requirements.

The display of the historical data in the Machine Advisor proved to be insufficient for this process. Since data points can be uploaded at a minimum interval of 15 minutes, dynamic signal characteristics such as conveyor belt current or voltage are not fully recorded. The resulting data points, appearing almost arbitrary, lead to a more complicated analysis of the actual behavior.

Initially, no variables could be read from the server during the communication configuration between the OPC UA Server and MATLAB. Although the server setup allows access to the contained variables, they were not visible to the MATLAB client. Only the implementation of an information model allowed the client to access and read the variables.

During the execution of this thesis, the accurate description of the model plant turns out to be particularly valuable. Only if the interaction of the individual components is known, the obtained data can be interpreted correctly and contribute to the monitoring of the plant.

8.2 Outlook

One option for improving system monitoring is to extend it by adding appropriate sensors. For example, vibration sensors can additionally monitor the movement of the robot and are indicators of faulty behavior.

Furthermore, there is still room for improvement, especially in data analysis in MATLAB. Besides the classification of current data, future signal characteristics can be predicted, and the resulting mean time to failure can be determined. The required error data could be obtained from a MATLAB Simulink model of the plant, where error signals can be injected or the system failure dynamics modeled.

Another conceivable step would be to connect the analysis in MATLAB to a cloud. Integrated interfaces for cloud storage and databases enable convenient read and write access to data. Other systems could use the availability of the analyzed data in the cloud. For example, the information could be displayed for the plant operator using augmented reality glasses.

Bibliography

- [1] SENDLER, Ulrich: Industrie 4.0–Beherrschung der industriellen Komplexität mit SysLM (Systems lifecycle management). In: *Industrie 4.0*. Springer, 2013, S. 1–19
- [2] BAUERNHANSL, Thomas ; TEN HOMPEL, Michael ; VOGEL-HEUSER, Birgit: *Industrie 4.0 in Produktion, Automatisierung und Logistik: Anwendung-Technologien-Migration*. Springer, 2014
- [3] HUBER, Walter: *Industrie 4.0 kompakt–Wie Technologien unsere Wirtschaft und unsere Unternehmen verändern*. Springer, 2018
- [4] BRACHT, Uwe ; GECKLER, Dieter ; WENZEL, Sigrid: *Digitale Fabrik: Methoden und Praxisbeispiele*. Springer-Verlag, 2018
- [5] BETTENHAUSEN, Kurt D. ; KOWALEWSKI, Stefan: Cyber-physical systems: Chancen und Nutzen aus Sicht der Automation. In: *VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik (2013)*, S. 9–10
- [6] REINHEIMER, Stefan: *Industrie 4.0: Herausforderungen, Konzepte und Praxisbeispiele*. Springer-Verlag, 2017
- [7] FORSCHUNGSUNION WIRTSCHAFT-WISSENSCHAFT, Promotorengruppe K.: acatech–Deutsche Akademie der Technikwissenschaften e. In: *V.(Hrsg.) (2013)*
- [8] MÜLLER, Bernd ; HÄRTIG, Frank: Herausforderungen und Lösungsansätze zur einheitlichen Kommunikation von Messdaten für Industrie 4.0 und das Internet of Things. In: *Industrie 4.0*. Springer, 2017, S. 49–58
- [9] VOGEL-HEUSER, B: Herausforderungen und Anforderungen aus Sicht der IT und der Automatisierungstechnik [Challenges and requirements from the perspective of IT and automation technology]. In: *T Bauernhansl, M ten Hompel, B Vogel-Heuser (Edn.) Industrie 4 (2014)*
- [10] MONOSTORI, László ; KÁDÁR, Botond ; BAUERNHANSL, T ; KONDOH, S ; KUMARA, S ; REINHART, G ; SAUER, O ; SCHUH, G ; SIHN, W ; UEDA, K: Cyber-physical systems in manufacturing. In: *Cirp Annals 65 (2016)*, Nr. 2, S. 621–641

- [11] VEICHTLBAUER, Armin ; ORTMAYER, Martin ; HEISTRACHER, Thomas: OPC UA integration for field devices. In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)* IEEE, 2017, S. 419–424
- [12] ANDEFINGER, Volker P. ; HÄNISCH, Till: *Industrie 4.0: Wie cyber-physische Systeme die Arbeitswelt verändern*. Springer, 2017
- [13] BECKHOFF AUTOMATION GMBH: *From Sensor to IT Enterprise - Big Data & Analytics in the cloud*. ftp://ftp.beckhoff.com/Software/embPC-Control/Solution/Demo-IoT/Flyer-IoT-Sensor_to_Cloud.pdf. – Viewed on 20 June 2019
- [14] HABER, Peter ; LAMPOLTSHAMMER, Thomas ; MAYR, Manfred: *Data Science Analytics and Applications: Proceedings of the 1st International Data Science Conference iDSC2017 (German and English Edition)*. Springer Vieweg, 2017. – ISBN 3658192860, 9783658192860
- [15] LANGMANN, Reinhard ; ROJAS-PEÑA, Leandro F.: A PLC as an Industry 4.0 component. In: *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)* IEEE, 2016, S. 10–15
- [16] SEITZ, Matthias: *Speicherprogrammierbare Steuerungen für die Fabrik-und Prozessautomation: Strukturierte und objektorientierte SPS-Programmierung, Motion Control, Sicherheit, vertikale Integration*. Carl Hanser Verlag GmbH Co KG, 2015
- [17] KARAALI, Cihat: *Grundlagen der Steuerungstechnik: Einführung mit Übungen*. Springer-Verlag, 2018
- [18] BATHELT, Jens: *Entwicklungsmethodik für SPS-gesteuerte mechatronische Systeme*. Bd. 405. ETH Zurich, 2007
- [19] HEINRICH, Berthold ; LINKE, Petra ; GLÖCKLER, Michael: *Grundlagen Automatisierung: Sensorik, Regelung, Steuerung*. Springer-Verlag, 2017
- [20] GOLDSTEIN, Sven: *Beckhoff-Lösungen für Industrie 4.0 und IoT - Einfach, offen und standardisiert*. https://www.pc-control.net/pdf/022016/products/pcc_0216_industrie-40_d.pdf. – Viewed on 19 June 2019
- [21] HANNELIUS, Tom ; SALMENPERA, Mikko ; KUIKKA, Seppo: Roadmap to adopting OPC UA. In: *2008 6th IEEE International Conference on Industrial Informatics* IEEE, 2008, S. 756–761
- [22] HOPPE, Stefan ; SCHREIER, Jürgen: *Industrie 4.0 in der Praxis - IoT-Basics: Was ist OPC UA?* <https://www.industry-of-things.de/iot-basics-was-ist-opc-ua-a-727188/>. Version:2019. – Viewed on 24 June 2019

- [23] OPC UA-COMMUNITY: *OPC Unified Architecture – Wegbereiter der vierten industriellen (R)Evolution*. OPC Foundation, 2013
- [24] LEITNER, Stefan-Helmut ; MAHNKE, Wolfgang: OPC UA–service-oriented architecture for industrial applications. In: *ABB Corporate Research Center 48* (2006), S. 61–66
- [25] ASCOLAB GMBH: *OPC UA Concepts*. <http://www.ascolab.com/en/technology-unified-architecture/technology-aconcepts.html>. – Viewed on 25 June 2019
- [26] MAHNKE, Wolfgang ; LEITNER, Stefan-Helmut ; DAMM, Matthias: *OPC unified architecture*. Springer Science & Business Media, 2009
- [27] PIGAN, R. ; METTER, M.: *Automatisieren mit PROFINET: Industrielle Kommunikation auf Basis von Industrial Ethernet*. Wiley, 2015. – ISBN 9783895789496
- [28] HMS INDUSTRIAL NETWORKS AB: *Profinet*. <https://www.feldbusse.de/Profinet/profinet.shtml>. – Viewed on 14 July 2019
- [29] SIEMENS: *Umsetzung des Kommunikationskonzepts von PROFINET CBA mit PROFINET IO*. https://cache.industry.siemens.com/dl/files/355/60520355/att_117434/v1/60520355_dp_porting_profinet_cba_de.pdf. – Viewed on 14 July 2019
- [30] GOLDENBERG, Niv ; WOOL, Avishai: Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. In: *International Journal of Critical Infrastructure Protection* 6 (2013), Nr. 2, S. 63–75
- [31] VOYIATZIS, Artemios G. ; KATSIGIANNIS, Konstantinos ; KOUBIAS, Stavros: A Modbus/TCP fuzzer for testing internetworked industrial systems. In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)* IEEE, 2015, S. 1–6
- [32] MURPHY, Kevin P.: *Machine learning: a probabilistic perspective*. MIT press, 2012
- [33] KRUSE, Rudolf ; BORGELT, C ; KLAWONN, F ; MOEWES, C ; RUSS, G ; STEINBRECHER, M: *Computational Intelligence-Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 1. Auflage. Wiesbaden: Vieweg+ Teubner (2011)
- [34] WILLEY, Richard: *Choosing the Best Machine Learning Classification Model and Avoiding Overfitting*. <https://de.mathworks.com>. Version: Issued: 22 September 2011

- [35] SIEMENS: *Totally Integrated Automation Portal*. <https://new.siemens.com/global/de/produkte/automatisierung/industrie-software/automatisierungs-software/tia-portal.html>. – Viewed on 27 June 2019
- [36] SIEMENS: *Siemens OPC UA Modeling Editor - Functional description*. <https://support.industry.siemens.com/cs/ww/en/view/109755133>. – Viewed on 27 June 2019
- [37] SCHNEIDER ELECTRIC: *EcoStruxure Machine Advisor von Schneider Electric maximiert den Wert von Daten für OEMs*. <https://www.se.com/de/de/about-us/news/press-releases/2018/ecostruxure-machine-advisor.jsp>. – Viewed on 8 July 2019
- [38] SCHNEIDER ELECTRIC: *ComXBox product introduction and operation (Chinese language)*. – PDF file can be viewed on the CD
- [39] LUCAS-NÜLLE GMBH: *INDUSTRIE 4.0 PRODUKTIONSSTRASSE - Ausstattung*. www.lucas-nuelle.de. – Viewed on 5 June 2019
- [40] LUCAS-NÜLLE-AUTORENTEAM: *ILA-Kurs CSF 4: ERP-Lab für Industrie 4.0*. www.lucas-nuelle.de. – Viewed on 9 June 2019
- [41] NAGLIC, Marijan: *ILA-Kurs Inbetriebnahme des KUKA Roboters*. www.lucas-nuelle.de. – Viewed on 11 June 2019
- [42] KUKA DEUTSCHLAND GMBH: *KR AGILUS sixx - Mit W- und C-Variante - Spezifikation*. Issued: 19 April 2018. – Version: Spez KR AGILUS sixx V13 - PDF file can be viewed on the CD
- [43] MATHWORKS: *Pattern Recognition and Classification*. https://de.mathworks.com/help/deeplearning/pattern-recognition-and-classification.html?s_tid=CRUX_lftnav. – Viewed on 2 August 2019
- [44] REINHART, Gunther: *Handbuch Industrie 4.0: Geschäftsmodelle, Prozesse, Technik*. Carl Hanser Verlag GmbH Co KG, 2017
- [45] BECKER, Norbert ; EGGELING, Manfred: *ILA-Kurs CSF 1: Industrie 4.0*. www.lucas-nuelle.de. – Viewed on 6 May 2019

A Appendix

The appendix to the thesis is on CD and can be viewed at Prof. Dr.-Ing. Florian Wenck or Associate Prof. Dr.-Ing. Shen JianQiang.

A.1 Parameter Template: USST_industrylab4_0

数据名称	参数类别	国际类型	报警级别	脉冲曲线类型	缩略名	ID	单位	比例系数	类型	传输类型	传输速率	寄存器地址	寄存器大小	是否上传	备注
IMS1 Current (I)	实时数据	CUR			plc_meas01	1	A		Float	R	5	0	2	Y	Range:Enum;other;;
IMS3a Current (I)	实时数据	CUR			plc_meas02	2	A		Float	R	5	2	2	Y	Range:Enum;other;;
IMS3b Current (I)	实时数据	CUR			plc_meas03	3	A		Float	R	5	4	2	Y	Range:Enum;other;;
IMS4a Current (I)	实时数据	CUR			plc_meas04	4	A		Float	R	5	6	2	Y	Range:Enum;other;;
IMS4b Current (I)	实时数据	CUR			plc_meas05	5	A		Float	R	5	8	2	Y	Range:Enum;other;;
IMS5a Current (I)	实时数据	CUR			plc_meas06	6	A		Float	R	5	10	2	Y	Range:Enum;other;;
IMS5b Current (I)	实时数据	CUR			plc_meas07	7	A		Float	R	5	12	2	Y	Range:Enum;other;;
IMS6 Current (I)	实时数据	CUR			plc_meas08	8	A		Float	R	5	14	2	Y	Range:Enum;other;;
IMS7 Current (I)	实时数据	CUR			plc_meas09	9	A		Float	R	5	16	2	Y	Range:Enum;other;;
IMS1 Voltage (V)	实时数据	VOL			plc_meas11	10	V		Float	R	5	18	2	Y	Range:Enum;other;;
IMS3a Voltage (V)	实时数据	VOL			plc_meas12	11	V		Float	R	5	20	2	Y	Range:Enum;other;;
IMS3b Voltage (V)	实时数据	VOL			plc_meas13	12	V		Float	R	5	22	2	Y	Range:Enum;other;;
IMS4a Voltage (V)	实时数据	VOL			plc_meas14	13	V		Float	R	5	24	2	Y	Range:Enum;other;;
IMS4b Voltage (V)	实时数据	VOL			plc_meas15	14	V		Float	R	5	26	2	Y	Range:Enum;other;;
IMS5a Voltage (V)	实时数据	VOL			plc_meas16	15	V		Float	R	5	28	2	Y	Range:Enum;other;;
IMS5b Voltage (V)	实时数据	VOL			plc_meas17	16	V		Float	R	5	30	2	Y	Range:Enum;other;;
IMS6 Voltage (V)	实时数据	VOL			plc_meas18	17	V		Float	R	5	32	2	Y	Range:Enum;other;;
IMS7 Voltage (V)	实时数据	VOL			plc_meas19	18	V		Float	R	5	34	2	Y	Range:Enum;other;;
IMS1 Power (W)	实时数据	POW			plc_meas20	19	W		Float	R	5	36	2	Y	Range:Enum;other;;
IMS3a Power (W)	实时数据	POW			plc_meas21	20	W		Float	R	5	38	2	Y	Range:Enum;other;;
IMS3b Power (W)	实时数据	POW			plc_meas22	21	W		Float	R	5	40	2	Y	Range:Enum;other;;
IMS4a Power (W)	实时数据	POW			plc_meas23	22	W		Float	R	5	42	2	Y	Range:Enum;other;;
IMS4b Power (W)	实时数据	POW			plc_meas24	23	W		Float	R	5	44	2	Y	Range:Enum;other;;
IMS5a Power (W)	实时数据	POW			plc_meas25	24	W		Float	R	5	46	2	Y	Range:Enum;other;;
IMS5b Power (W)	实时数据	POW			plc_meas26	25	W		Float	R	5	48	2	Y	Range:Enum;other;;
IMS6 Power (W)	实时数据	POW			plc_meas27	26	W		Float	R	5	50	2	Y	Range:Enum;other;;
IMS7 Power (W)	实时数据	POW			plc_meas28	27	W		Float	R	5	52	2	Y	Range:Enum;other;;
IMS1_countMotorisActive	实时数据	TIS			plc_meas29	28			Int32	R	5	54	2	Y	Range:Enum;other;;
IMS1_countSenBeltLeft	实时数据	TIS			plc_meas30	29			Int32	R	5	56	2	Y	Range:Enum;other;;
IMS1_countSenBeltRight	实时数据	TIS			plc_meas31	30			Int32	R	5	58	2	Y	Range:Enum;other;;
IMS3a_countMotorisActive	实时数据	TIS			plc_meas32	31			Int32	R	5	60	2	Y	Range:Enum;other;;
IMS3a_countCylSorter	实时数据	TIS			plc_meas33	32			Int32	R	5	62	2	Y	Range:Enum;other;;
IMS3a_countCylSort	实时数据	TIS			plc_meas34	33			Int32	R	5	64	2	Y	Range:Enum;other;;
IMS3a_countSenBeltLeft	实时数据	TIS			plc_meas35	34			Int32	R	5	66	2	Y	Range:Enum;other;;
IMS3a_countSenBeltRight	实时数据	TIS			plc_meas36	35			Int32	R	5	68	2	Y	Range:Enum;other;;
IMS3a_countSenStopperTop	实时数据	TIS			plc_meas37	36			Int32	R	5	70	2	Y	Range:Enum;other;;

IMS5b_countSenPressCyInNotActuated	实时数据	TIS					plc_meas75	74				Int32	R	5	146	2	Y	Range::Enum;other;;
IMS5b_countSenPressCyActuated	实时数据	TIS					plc_meas76	75				Int32	R	5	148	2	Y	Range::Enum;other;;
IMS5b_countSenMagazine	实时数据	TIS					plc_meas77	76				Int32	R	5	150	2	Y	Range::Enum;other;;
IMS6_countMotorisActive	实时数据	TIS					plc_meas78	77				Int32	R	5	152	2	Y	Range::Enum;other;;
IMS6_countCyStopper	实时数据	TIS					plc_meas79	78				Int32	R	5	154	2	Y	Range::Enum;other;;
IMS6_countSenBeltLeft	实时数据	TIS					plc_meas80	79				Int32	R	5	156	2	Y	Range::Enum;other;;
IMS6_countSenBeltRight	实时数据	TIS					plc_meas81	80				Int32	R	5	158	2	Y	Range::Enum;other;;
IMS6_countSensTopperTop	实时数据	TIS					plc_meas82	81				Int32	R	5	160	2	Y	Range::Enum;other;;
IMS6_countSenOptBottom	实时数据	TIS					plc_meas83	82				Int32	R	5	162	2	Y	Range::Enum;other;;
IMS6_countSenInd	实时数据	TIS					plc_meas84	83				Int32	R	5	164	2	Y	Range::Enum;other;;
IMS6_countSenkap	实时数据	TIS					plc_meas85	84				Int32	R	5	166	2	Y	Range::Enum;other;;
IMS6_countSenOptTop	实时数据	TIS					plc_meas86	85				Int32	R	5	168	2	Y	Range::Enum;other;;
IMS7_countMotorisActive	实时数据	TIS					plc_meas87	86				Int32	R	5	170	2	Y	Range::Enum;other;;
IMS7_countCyStopper	实时数据	TIS					plc_meas88	87				Int32	R	5	172	2	Y	Range::Enum;other;;
IMS7_countCylSwivelTable	实时数据	TIS					plc_meas89	88				Int32	R	5	174	2	Y	Range::Enum;other;;
IMS7_countCylLift	实时数据	TIS					plc_meas90	89				Int32	R	5	176	2	Y	Range::Enum;other;;
IMS7_countSenBeltLeft	实时数据	TIS					plc_meas91	90				Int32	R	5	178	2	Y	Range::Enum;other;;
IMS7_countSenBeltRight	实时数据	TIS					plc_meas92	91				Int32	R	5	180	2	Y	Range::Enum;other;;
IMS7_countVacuumValve	实时数据	TIS					plc_meas93	92				Int32	R	5	182	2	Y	Range::Enum;other;;
IMS7_countSensTopperTop	实时数据	TIS					plc_meas94	93				Int32	R	5	184	2	Y	Range::Enum;other;;
IMS7_countSensSwivelTable0	实时数据	TIS					plc_meas95	94				Int32	R	5	186	2	Y	Range::Enum;other;;
IMS7_countSensSwivelTable90	实时数据	TIS					plc_meas96	95				Int32	R	5	188	2	Y	Range::Enum;other;;
IMS7_countSenVacuumMonitoring	实时数据	TIS					plc_meas97	96				Int32	R	5	190	2	Y	Range::Enum;other;;
IMS7_countSwitchLiftTop	实时数据	TIS					plc_meas98	97				Int32	R	5	192	2	Y	Range::Enum;other;;
IMS1_motorOperatingHours	实时数据	TIM					plc_meas99	98	m			Int64	R	5	194	4	Y	Range::Enum;other;;
IMS2a_motorOperatingHours	实时数据	TIM					plc_meas100	99	m			Int64	R	5	198	4	Y	Range::Enum;other;;
IMS3b_motorOperatingHours	实时数据	TIM					plc_meas101	100	m			Int64	R	5	202	4	Y	Range::Enum;other;;
IMS4a_motorOperatingHours	实时数据	TIM					plc_meas102	101	m			Int64	R	5	206	4	Y	Range::Enum;other;;
IMS4b_motorOperatingHours	实时数据	TIM					plc_meas103	102	m			Int64	R	5	210	4	Y	Range::Enum;other;;
IMS5a_motorOperatingHours	实时数据	TIM					plc_meas104	103	m			Int64	R	5	214	4	Y	Range::Enum;other;;
IMS5b_motorOperatingHours	实时数据	TIM					plc_meas105	104	m			Int64	R	5	218	4	Y	Range::Enum;other;;
IMS6_motorOperatingHours	实时数据	TIM					plc_meas106	105	m			Int64	R	5	222	4	Y	Range::Enum;other;;
IMS7_motorOperatingHours	实时数据	TIM					plc_meas107	106	m			Int64	R	5	226	4	Y	Range::Enum;other;;
KUKA_CURR_ACT_Axis1	实时数据	CUR					plc_meas108	107				Float	R	5	230	2	Y	Range::Enum;other;;
KUKA_CURR_ACT_Axis2	实时数据	CUR					plc_meas109	108				Float	R	5	232	2	Y	Range::Enum;other;;
KUKA_CURR_ACT_Axis3	实时数据	CUR					plc_meas110	109				Float	R	5	234	2	Y	Range::Enum;other;;
KUKA_CURR_ACT_Axis4	实时数据	CUR					plc_meas111	110				Float	R	5	236	2	Y	Range::Enum;other;;

KUKA_CURR_ACT_Axis5	实时数据	CUR				plc_meas112	111				Float	R	5	238	2	Y	Range::Enum;other;;
KUKA_CURR_ACT_Axis6	实时数据	CUR				plc_meas113	112				Float	R	5	240	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis1	实时数据	TEM				plc_meas114	113				Float	R	5	242	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis2	实时数据	TEM				plc_meas115	114				Float	R	5	244	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis3	实时数据	TEM				plc_meas116	115				Float	R	5	246	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis4	实时数据	TEM				plc_meas117	116				Float	R	5	248	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis5	实时数据	TEM				plc_meas118	117				Float	R	5	250	2	Y	Range::Enum;other;;
KUKA_MOT_TEMP_Axis6	实时数据	TEM				plc_meas119	118				Float	R	5	252	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis1	实时数据					plc_meas120	119				Float	R	5	254	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis2	实时数据					plc_meas121	120				Float	R	5	256	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis3	实时数据					plc_meas122	121				Float	R	5	258	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis4	实时数据					plc_meas123	122				Float	R	5	260	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis5	实时数据					plc_meas124	123				Float	R	5	262	2	Y	Range::Enum;other;;
KUKA_TORQUE_ACT_Axis6	实时数据					plc_meas125	124				Float	R	5	264	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis1	实时数据					plc_meas126	125				Float	R	5	266	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis2	实时数据					plc_meas127	126				Float	R	5	268	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis3	实时数据					plc_meas128	127				Float	R	5	270	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis4	实时数据					plc_meas129	128				Float	R	5	272	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis5	实时数据					plc_meas130	129				Float	R	5	274	2	Y	Range::Enum;other;;
KUKA_VEL_ACT_Axis6	实时数据					plc_meas131	130				Float	R	5	276	2	Y	Range::Enum;other;;

A.2 Data Type: DATA_TO_MASTER

Totally Integrated Automation Portal		
DATA_TO_MASTER		
DATA_TO_MASTER Properties		
General		
Name	DATA_TO_MASTER	Number 6
Language		Type UDT
Numbering		
Information		
Title		Author
Family		Version
		Comment
		User-defined ID
Name	Data type	Default value
Current	Real	0.0
Voltage	Real	0.0
Power	Real	0.0
countSenBeltLeft	DInt	0
countSenBeltRight	DInt	0
motorIsActive	Bool	false
countMotorIsActive	DInt	0
countCylStopper	DInt	0
countCylSort	DInt	0
countSenStopperTop	DInt	0
countSenMagazine	DInt	0
countCylPressing	DInt	0
countSenPressCylNotActuated	DInt	0
countSenPressCylActuated	DInt	0
countSenOptBottom	DInt	0
countSenOptTop	DInt	0
countSenInd	DInt	0
countSenKap	DInt	0
countCylSwivelTable	DInt	0
countCylLift	DInt	0
countVacuumValve	DInt	0
countSenSwivelTable0	DInt	0
countSenSwivelTable90	DInt	0
countSenVacuumMonitoring	DInt	0
countSwitchLiftTop	DInt	0

A.3 Data Type: KUKADATA_TO_MASTER

Totally Integrated Automation Portal		
KUKADATA_TO_MASTER		
KUKADATA_TO_MASTER Properties		
General		
Name	KUKADATA_TO_MASTER	Type UDT
Language		Numbering
Information		
Title		Author
Family		Version
		Comment
		User-defined ID
Name	Data type	Default value
CURR_ACT_Axis1	DInt	0
CURR_ACT_Axis2	DInt	0
CURR_ACT_Axis3	DInt	0
CURR_ACT_Axis4	DInt	0
CURR_ACT_Axis5	DInt	0
CURR_ACT_Axis6	DInt	0
MOT_TEMP_Axis1	DInt	0
MOT_TEMP_Axis2	DInt	0
MOT_TEMP_Axis3	DInt	0
MOT_TEMP_Axis4	DInt	0
MOT_TEMP_Axis5	DInt	0
MOT_TEMP_Axis6	DInt	0
TORQUE_ACT_Axis1	DInt	0
TORQUE_ACT_Axis2	DInt	0
TORQUE_ACT_Axis3	DInt	0
TORQUE_ACT_Axis4	DInt	0
TORQUE_ACT_Axis5	DInt	0
TORQUE_ACT_Axis6	DInt	0
VEL_ACT_Axis1	DInt	0
VEL_ACT_Axis2	DInt	0
VEL_ACT_Axis3	DInt	0
VEL_ACT_Axis4	DInt	0
VEL_ACT_Axis5	DInt	0
VEL_ACT_Axis6	DInt	0

A.4 Data Block: ModbusServerData

Totally Integrated Automation Portal					
ModbusServerData [DB21]					
ModbusServerData Properties					
General					
Name	ModbusServerData	Number	21	Type	DB
Language	DB	Numbering	Manual		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	
Name	Data type	Start value	Retain		
▼ Static					
MotorCurrentIMS1	Real	0.0			False
MotorCurrentIMS3a	Real	0.0			False
MotorCurrentIMS3b	Real	0.0			False
MotorCurrentIMS4a	Real	0.0			False
MotorCurrentIMS4b	Real	0.0			False
MotorCurrentIMS5a	Real	0.0			False
MotorCurrentIMS5b	Real	0.0			False
MotorCurrentIMS6	Real	0.0			False
MotorCurrentIMS7	Real	0.0			False
MotorVoltageIMS1	Real	0.0			False
MotorVoltageIMS3a	Real	0.0			False
MotorVoltageIMS3b	Real	0.0			False
MotorVoltageIMS4a	Real	0.0			False
MotorVoltageIMS4b	Real	0.0			False
MotorVoltageIMS5a	Real	0.0			False
MotorVoltageIMS5b	Real	0.0			False
MotorVoltageIMS6	Real	0.0			False
MotorVoltageIMS7	Real	0.0			False
MotorPowerIMS1	Real	0.0			False
MotorPowerIMS3a	Real	0.0			False
MotorPowerIMS3b	Real	0.0			False
MotorPowerIMS4a	Real	0.0			False
MotorPowerIMS4b	Real	0.0			False
MotorPowerIMS5a	Real	0.0			False
MotorPowerIMS5b	Real	0.0			False
MotorPowerIMS6	Real	0.0			False
MotorPowerIMS7	Real	0.0			False
IMS1_countMotorIsActive	DInt	0			False
IMS1_countSenBeltLeft	DInt	0			False
IMS1_countSenBeltRight	DInt	0			False
IMS3a_countMotorIsActive	DInt	0			False
IMS3a_countCylStopper	DInt	0			False
IMS3a_countCylSort	DInt	0			False
IMS3a_countSenBeltLeft	DInt	0			False
IMS3a_countSenBeltRight	DInt	0			False
IMS3a_countSenStopperTop	DInt	0			False
IMS3a_countSenMagazine	DInt	0			False
IMS3b_countMotorIsActive	DInt	0			False
IMS3b_countCylStopper	DInt	0			False
IMS3b_countCylSort	DInt	0			False

Totally Integrated Automation Portal			
Name	Data type	Start value	Retain
IMS3b_countSenBeltLeft	DInt	0	False
IMS3b_countSenBeltRight	DInt	0	False
IMS3b_countSenStopperTop	DInt	0	False
IMS3b_countSenMagazine	DInt	0	False
IMS4a_countMotorIsActive	DInt	0	False
IMS4a_countCylStopper	DInt	0	False
IMS4a_countCylSort	DInt	0	False
IMS4a_countSenBeltLeft	DInt	0	False
IMS4a_countSenBeltRight	DInt	0	False
IMS4a_countSenStopperTop	DInt	0	False
IMS4a_countSenMagazine	DInt	0	False
IMS4b_countMotorIsActive	DInt	0	False
IMS4b_countCylStopper	DInt	0	False
IMS4b_countCylSort	DInt	0	False
IMS4b_countSenBeltLeft	DInt	0	False
IMS4b_countSenBeltRight	DInt	0	False
IMS4b_countSenStopperTop	DInt	0	False
IMS4b_countSenMagazine	DInt	0	False
IMS5a_countMotorIsActive	DInt	0	False
IMS5a_countCylStopper	DInt	0	False
IMS5a_countCylPressing	DInt	0	False
IMS5a_countSenBeltLeft	DInt	0	False
IMS5a_countSenBeltRight	DInt	0	False
IMS5a_countSenStopperTop	DInt	0	False
IMS5a_countSenPressCylNotActuated	DInt	0	False
IMS5a_countSenPressCylActuated	DInt	0	False
IMS5a_countSenMagazine	DInt	0	False
IMS5b_countMotorIsActive	DInt	0	False
IMS5b_countCylStopper	DInt	0	False
IMS5b_countCylPressing	DInt	0	False
IMS5b_countSenBeltLeft	DInt	0	False
IMS5b_countSenBeltRight	DInt	0	False
IMS5b_countSenStopperTop	DInt	0	False
IMS5b_countSenPressCylNotActuated	DInt	0	False
IMS5b_countSenPressCylActuated	DInt	0	False
IMS5b_countSenMagazine	DInt	0	False
IMS6_countMotorIsActive	DInt	0	False
IMS6_countCylStopper	DInt	0	False
IMS6_countSenBeltLeft	DInt	0	False
IMS6_countSenBeltRight	DInt	0	False
IMS6_countSenStopperTop	DInt	0	False
IMS6_countSenOptBottom	DInt	0	False
IMS6_countSenInd	DInt	0	False
IMS6_countSenKap	DInt	0	False
IMS6_countSenOptTop	DInt	0	False
IMS7_countMotorIsActive	DInt	0	False
IMS7_countCylStopper	DInt	0	False
IMS7_countCylSwivelTable	DInt	0	False
IMS7_countCylLift	DInt	0	False
IMS7_countSenBeltLeft	DInt	0	False
IMS7_countSenBeltRight	DInt	0	False
IMS7_countVacuumValve	DInt	0	False

Totally Integrated Automation Portal			
Name	Data type	Start value	Retain
IMS7_countSenStopperTop	DInt	0	False
IMS7_countSenSwivelTable0	DInt	0	False
IMS7_countSenSwivelTable90	DInt	0	False
IMS7_countSenVacuumMonitoring	DInt	0	False
IMS7_countSwitchLiftTop	DInt	0	False
IMS1_motorOperatingHours	Lint	0	False
IMS3a_motorOperatingHours	Lint	0	False
IMS3b_motorOperatingHours	Lint	0	False
IMS4a_motorOperatingHours	Lint	0	False
IMS4b_motorOperatingHours	Lint	0	False
IMS5a_motorOperatingHours	Lint	0	False
IMS5b_motorOperatingHours	Lint	0	False
IMS6_motorOperatingHours	Lint	0	False
IMS7_motorOperatingHours	Lint	0	False
KUKA_CURR_ACT_Axis1	Real	0.0	False
KUKA_CURR_ACT_Axis2	Real	0.0	False
KUKA_CURR_ACT_Axis3	Real	0.0	False
KUKA_CURR_ACT_Axis4	Real	0.0	False
KUKA_CURR_ACT_Axis5	Real	0.0	False
KUKA_CURR_ACT_Axis6	Real	0.0	False
KUKA_MOT_TEMP_Axis1	Real	0.0	False
KUKA_MOT_TEMP_Axis2	Real	0.0	False
KUKA_MOT_TEMP_Axis3	Real	0.0	False
KUKA_MOT_TEMP_Axis4	Real	0.0	False
KUKA_MOT_TEMP_Axis5	Real	0.0	False
KUKA_MOT_TEMP_Axis6	Real	0.0	False
KUKA_TORQUE_ACT_Axis1	Real	0.0	False
KUKA_TORQUE_ACT_Axis2	Real	0.0	False
KUKA_TORQUE_ACT_Axis3	Real	0.0	False
KUKA_TORQUE_ACT_Axis4	Real	0.0	False
KUKA_TORQUE_ACT_Axis5	Real	0.0	False
KUKA_TORQUE_ACT_Axis6	Real	0.0	False
KUKA_VEL_ACT_Axis1	Real	0.0	False
KUKA_VEL_ACT_Axis2	Real	0.0	False
KUKA_VEL_ACT_Axis3	Real	0.0	False
KUKA_VEL_ACT_Axis4	Real	0.0	False
KUKA_VEL_ACT_Axis5	Real	0.0	False
KUKA_VEL_ACT_Axis6	Real	0.0	False

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Shanghai, 10. August 2019

Ort, Datum

Unterschrift