UNIVERSITI TUN HUSSEIN ONN MALAYSIA

STATUS CONFIRMATION FOR UNDERGRADUATE PROJECT REPORT

IMPLEMENTATION OF A MANUFACTURING EXECUTION SYSTEM FOR A LEARNING FACTORY

ACADEMIC SESSION: 2022/2023

I, **MATTIS TOM RITTER**, agree to allow this Undergraduate Project Report to be kept at the Library under the following terms:

1. This Undergraduate Project Report is the property of the Universiti Tun Hussein Onn Malaysia.
2. The library has the right to make copies for educational purposes only.
3. The library is allowed to make copies of this report for educational exchange between higher educational institutions.
4. ** Please Mark (√)

☐	CONFIDENTIAL	(Contains information of high security or of great importance to Malaysia as STIPULATED under the OFFICIAL SECRET ACT 1972)
☐	RESTRICTED	(Contains restricted information as determined by the Organization/institution where research was conducted)
☑	FREE ACCESS	

Approved by

Mattis Ritter

PROF. MADYA DR. LOW CHENG YEE
Jabatan Kejuruteraan Mekanik
Fakulti Kejuruteraan Mekanikal dan Pembuatan
Universiti Tun Hussein Onn Malaysia

Mattis Tom Ritter

Permanent Address:

19, Jalan Kencana 1A/26,

83300 Pura Kencana

BATU PAHAT,

JOHOR

Date: 04.07.2023

Prof. Madya Ir. Dr. Low Cheng Yee

Date: 05. 07. 2023

NOTE:

** If this Undergraduate Project Report is classified as CONFIDENTIAL or RESTRICTED, please attach the letter from the relevant authority/organization stating reasons and duration for such classifications.

IMPLEMENTATION OF A MANUFACTURING EXECUTION SYSTEM FOR A LEARNING FACTORY

MATTIS TOM RITTER

A thesis submitted in partial
fulfilment of the requirement for the award of the
Bachelor's Degree of Mechanical Engineering with Honours

Faculty of Mechanical and Manufacturing Engineering
Universiti Tun Hussein Onn Malaysia

AUGUST 2023

I hereby declare that the work in this project report is my own except for quotations and summaries which have been duly acknowledged

Student : Mattis Ritter
Mattis Tom Ritter

Date : 04.07.2023

Supervisor : ……
Prof. Madya Ir. Dr. Low Cheng Yee

ACKNOWLEDGEMENT

I would like to use this space to give a special acknowledgment for Dr. Low Cheng Yee, who always supported me in every matter.

I would like to also thank all the persons working in the UTHM-Frauenhofer IEM Innovation Lab. It was great working together with very helping people. It was a great experience being part of the project learning factory. Especially one person I like to thank, Moritz Höhnel.

ABSTRACT

In times of the fourth industrial revolution data is the most valuable resource of a company. Manufacturing Execution Systems use digitalised production site information to improve the production processes. This report presents the implementation of such a system in order to provide a demonstrator for small and medium-sized enterprises. The focus is on the data gathering, storage and visualization. The project is conducted by using Arcstone software and shows the data flow from shopfloor level to the Manufacturing Execution System in detail. As the concept is not established in local companies, this report serves as a knowledge transfer platform for their next step of industrial revolution. It will show them the benefits of a more detailed observed production with a learning factory as example. Furthermore, companies can use the step-by-step explanations as a guideline to implement the system in their own production.

ABSTRAK

Pada masa revolusi industri keempat, data adalah sumber yang paling berharga bagi sesebuah syarikat. Sistem Perlaksanaan Pembuatan menggunakan maklumat tapak pengeluaran digital untuk menambah baik proses pengeluaran. Laporan ini membentangkan pelaksanaan sistem sedemikian untuk menyediakan penunjuk perasaan untuk perusahaan kecil dan sederhana. Tumpuan adalah pada pengumpulan data, penyimpanan dan visualisasi. Projek ini dijalankan dengan menggunakan perisian Arcstone dan menunjukkan aliran data dari peringkat shopfloor ke Sistem Perlaksanaan Pembuatan secara terperinci. Memandangkan konsep itu tidak diwujudkan dalam syarikat tempatan, laporan ini berfungsi sebagai platform pemindahan pengetahuan untuk langkah seterusnya revolusi perindustrian mereka. Ia akan menunjukkan kepada mereka faedah pengeluaran yang diperhatikan dengan lebih terperinci dengan kilang pembelajaran sebagai contoh. Tambahan pula, syarikat boleh menggunakan penjelasan langkah demi langkah sebagai garis panduan untuk melaksanakan sistem dalam pengeluaran mereka sendiri.

CONTENTS

	TITLE		i
	DECLARATION		ii
	ACKNOWLEDGEMENT		iii
	ABSTRACT		iv
	ABSTRAK		v
	LIST OF TABLES		vii
	LIST OF FIGURES		ix
	LIST OF SYMBOLS AND ABBREVIATIONS		xi
	LIST OF APPENDICES		xii
CHAPTER 1	INTRODUCTION		1
	1.1	Background Study	1
	1.2	Problem Statement	2
	1.3	Real World Example	3
	1.4	Objectives	4
		1.4.1 General Description of the Product	4
	1.5	Scope of Study	6
		1.5.1 Functional Requirements	7
		1.5.2 Non-Functional Requirements	9
CHAPTER 2	LITERATURE REVIEW		10
	2.1	MES	11
		2.1.1 Functionality of a MES	11
		2.1.2 MES in the Context of Industry 4.0	14
		2.1.3 Arcstone’s MES solutions	15

2.2 Artificial Intelligence 17
2.2.1 Neuronal Network in General 17
2.2.2 Convolutional Neuronal Network 20
2.2.3 Recurrent Neuronal Network 22
2.2.4 Summary Neuronal Networks 22
CHAPTER 3 METHODOLOGY 24
3.1 Flow Chart 24
3.2 Previous State of the Learning Factory 27
CHAPTER 4 RESULTS AND DISCUSSION 29
4.1 Dataflow from Automation Level to MES 29
4.1.1 Architecture of the Connection between Automation Software and Server 30
4.1.2 Pull Data from the Server to the MES Database 32
4.1.3 Access Data Inside the MES 36
4.2 Creating the Dashboard 39
4.3 How the AI-Lernfabrik Works 41
4.3.1 Applying the Neuronal Networks 42
CHAPTER 5 CONCUSSION AND RECOMMENDATIONS 49
5.1 MES 49
5.2 AI Lernfabrik 50
REFERENCE 53
APPENDIX 55

LIST OF TABLES

Table 1.1: MES Software 7
Table 1.2: Interfaces 8
Table 1.3: LI Learning Factory 8
Table 1.4: Non-Functional Requirements 9
Table 4.1: State of each station and the chutes 40
Table 4.2: Possible active state, code and target of each station 44
Table 4.3: Data 45

LIST OF FIGURES

Figure 2.1: Architecture of an enterprise with MES 13
Figure 2.2: Arcstone's MES solutions [6] 16
Figure 2.3: Arcstone in the context of the ISA 95 framework 16
Figure 2.4: Model of network [10] 18
Figure 2.5: Mathematical Neuron [10] 18
Figure 2.6: Pattern of number 2 [10] 19
Figure 2.7: CNN working principle [10] 21
Figure 2.8: Recurrent Neuronal Network [11] 22
Figure 3.1: MES Flowchart 25
Figure 3.2: AI-Learning Factory Flowchart 26
Figure 3.3: Fischertechnik Smart Factory [13] 27
Figure 3.4: Schematic Plan of the Learning Factory [15] 28
Figure 4.1: Dataflow from the automation software to the MES 29
Figure 4.2: Architecture examples of the connection 31
Figure 4.3: Used architecture for the learning factory 32
Figure 4.4: General connection setup of arc.quire profile 33
Figure 4.5: Source setup of arc.quire profile 34
Figure 4.6: Destination setup of arc.quire profile 35
Figure 4.7: Microsoft SQL Server Management Studio 36
Figure 4.8: Add new data source to a dashboard 37
Figure 4.9: Dashboard data source wizard 37
Figure 4.10: Bind data to grid 38
Figure 4.11: Calculated field 40
Figure 4.12: Dashboard 41
Figure 4.13: Sample message [15] 43
Figure 4.14: Sorted messages [15] 44

Figure 4.15: Input array (left) and output array (right) [15] 47
Figure 4.16: Multidimensional input array [15] 48
Figure 5.1: Controller overview [16] 50

LIST OF SYMBOLS AND ABBREVIATIONS

MES	-	Manufacturing Execution System
PLC	-	Programmable Logic Controller
AI	-	Artificial Intelligence
PC	-	Personal Computer
VDI	-	Verein Deutsche Ingenieure, *engl*. Association of German Engineers
ERP	-	Enterprise Resource Planning
ISA	-	International Society of Automation
CNN	-	Convolutional Neuronal Network
RNN	-	Recurrent Neuronal Network
OPCUA	-	Open Platform Communications United Architecture
URL	-	Uniform Resource Locator
SQL	-	Structured Query Language
JSON	-	JavaScript Object Notation
NFC	-	Near Field Communication
IoT	-	Internet of Things
MQTT	-	Message Queuing Telemetry Transport

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	AI-CAMPUS COURSE: LERNFABRIK	55
B	SOLUTION EXERCISES STEP BY STEP	152

CHAPTER 1

INTRODUCTION

1.1 Background Study

“The fourth industrial revolution is accelerating more and more. The foundations for a completely connected ‘smart factory’ are set. Digitalization, robotics, artificial intelligence […]. Today, these developments are changing industrial production fundamentally. […] Physical and digital processes are becoming increasingly intertwined. This also includes networking with customers: Their needs and demands are and remain the guiding principle of all of the company's work.” The Mercedes-Benz Group [1], one of the world’s leading car manufacturers, stating that the current industrialization is having an increasing effect. As three main pillars they name the digitalization, advancing robot technology and artificial intelligence (AI). All these pillars get more and more integrated into production processes, as they have positive effect on production costs.

One goal of Industry 4.0 is to connect workers and machines. Digitalization is providing a platform to collect production data from machines as well as humans. The collected information can be analysed and translated into languages machines and humans understand. Advancing robot technologies are helping to support and take over production steps. Especially those that humans can not handle, as example working in a hot environment. The third pillar, Artificial intelligence, is advancing the processing and analysis of production data.

The goal of this project is to work on a factory simulation that is capable of all the previous requirements for a future factory. The factory is completely operated by robots, to establish one pillar. For this project a Manufacturing Execution System

(MES) should be established. It is used to collect the sensor data, safe it to a cloud and then send it to inform the workers and machines that are part of the production. This means that it takes the pillar of Digitalization. The third pillar, the artificial intelligence, would be used to analyse the production data.
Apart from optimizing the production itself, the Mercedes-Benz Group outlines that the need for providing information to the customers also becomes more important. The manufacturing execution system has matching options for this purpose and shall be used to provide production information.

1.2 Problem Statement

While economy leading companies have already implemented important steps that are considered to be part of the fourth industrial industrialization, smaller local companies often lack those realization steps. They are still struggling to get the data as in most cases they are not digitised. But even if the companies have sensorized workstations that collect vast volumes of data, they are not connected to each other. This results in standalone islands, that are not contributing much to the improvement of the overall manufacturing process as the data is not distributed to a manufacturing execution system (MES). In a MES the data has a tremendous operational value. The factory simulation is established to show undigitized companies the advantages of a connected factory as well as giving them input for actual implementation.
This covers one of the pillars of industry 4.0, but there is still more to do for local companies to get most of the data. This is where artificial intelligence comes into fruition. New ways of data processing and analysis, as well as pattern recognition are made possible by neuronal networks, which leads to further improvement and efficiency gains. But the current workforce lacks these skills and has insufficient knowledge on the topic. Furthermore, a traditional mentality towards methodologies and operations, has translated into slow-moving transitions to new technologies. To get rid of these problems they need to be trained with courses that provide an easy hand-on to the field AI.
The learning factory is a convenient way to demonstrate that the path to manufacturing excellence is easy to adopt. It is common that managers are lacking the right mindset to start the process of making their factories ready for industry 4.0

and therefore the learning factory can be used to show that it achievable for everyone.

1.3 Real World Example

To improve the understanding of the simulated production process, a real-world factory is taken as example. The example is about a rim manufacturer for cars. In their factory they have a warehouse, an oven, a milling machine and after production workpieces get stored depending on colour. The rims can be ordered in colours red, blue and white. In the beginning an order picker collects raw material, puts it into an oven to temper the material. After the hardening is finished, the workpiece needs to be carried to the milling machine, where it is engraved. After that, a person needs sort it by colour into the right chute for the final storage.

The factory simulation shows how to automate all these work steps. For implementation at first robots are introduced. All the work steps that were executed by a human can also be handled by a robot. That means that one robot can fetch the raw material, a further robot can deliver the workpieces to the factoring sites. Conveyor belts can transport the workpieces between the factoring steps and finally to the chutes. Utilizing a colour detecting sensor, the sorting of workpieces can be executed automatically. After implementing the robots just one worker who overviews all production steps is necessary. As a reduction in total number of laborers is possible, a result is lower production costs.

To make it easier for the worker overviewing the production site production will be digitalised. This includes that data is gathered, analysed and visualized. In a final step, the pillar of artificial intelligence is deployed. An intelligent software is then in charge of supervising the production. The AI can also work with the provided production site data, by applying advanced analysis methods. If all is working perfectly together, there is just a production site manager of the rim producing company necessary to overview the progress of the factory, as execution, monitoring and controlling is done by software tools. Looking at more advantages, the aim is to improve factories process performance. The primary goal of the fourth industrial revolution for local companies should be to reduce failures during production and with that saving money. But with knowing how to establish a more digitalized

production system, companies can also use the other benefits that are coming along. As examples one can take the improved traceability of the workpieces and information chain for the customers.

1.4 Objectives

This chapter provides the specification of the project. Definitions will be given on what is needed to be done to complete the projects. It will be described what the MES and Lernfabrik objectives are.

For the MES project the final goal is to build a platform that enables the UTHM to transfer knowledge. Local companies should be able to understand the concept and the benefits of a Manufacturing Execution System. In detail this means to implement steps from data collection until the creation of a dashboard that gives an overview over the current production steps for the factory simulation. These implementations can then be presented and explained closer with an user guideline.

For the AI-Campus Lernfabrik project the objective is to transfer knowledge for the question: How to setup a factory for the usage of artificial intelligence? To answer the question, documents for a workshop should be created which explain all the steps from data preparation to applying a neuronal network. The workshop will be performed by the University for local companies to help them understand the implementation and the benefits. The workshop must be based on the AI Lernfabrik and includes general explanations as well as follow-along tasks and their solution. Furthermore, an analysis of the simulation factory’s hardware should be performed to understand if and how it is possible to use it.

1.4.1 General Description of the Product

The following parts will explain more detail what the finished products should look like. Therefore, a summary of the functions, a perspectives analyzation and a user characteristics analyzation will be made. These chapters are necessary to determine how the results looks like. They are also a guideline during development.

1.4.1.1 Product Functions

For the first part of the project a MES software is used. It is used to digitalize manufacturing processes. The process data should be put together in a dashboard. This dashboard should be usable for different persons. The first group of persons are the factory workers. The second group of persons are those that oversee the production. Both need to know which production step is running and which products are completed.
For the AI-Campus learning factory a set of explanations will be created. This will include a user manual and a workshop in which the functions will be discussed. Furthermore, it will be evaluated what steps need to be done to have the learning factory ready to implement to the software of the Lernfabrik.

1.4.1.2 Product Perspective

The factory is shown on an exhibition table. The results of the implementation of the MES Software can be accessed via a PC and a screen. The dashboard can be accessed with any PC that is connected to the UTHM network. It is also required to have Arcstone credentials to open the dashboard. For show purposes it is advised to locate the screen right next to the factory, that the prospects are able to compare what they can see happening at the factory and how the dashboard is visualizing it.
The AI-Campus factory learning material will be given digitally. It includes workshop materials. These should consist of two documents. A deeper explanation of every step and software that is used as well as a step-by-step instruction to solve example tasks.

1.4.1.3 User Characteristics

The expected users divide into two groups: Workers and Managers.
The workers have expertise for all the steps of the factory. He also expertise will be to use the PLC software in a way that he knows how to start the execution of the factory. He probably has no higher education and won't be able to implement any changes to the PLC software, MES Software nor the AI code. It will also be hard for him to read complex graphs. This means that data shown on the dashboard must be simple.
The manager will have higher education. He has experiences with complex software. He needs to overview that the MES or AI software is running correctly.

1.5 Scope of Study

The following chapter is describing the exact tasks that need to be performed. The tasks are collected in tables and are split for each subject. The following list shows abbreviations used specifically for the tables of this chapter:

ID - Is the identification number of a specification.

Pr. - Is the priority of a specification. It specifies whether a goal has to be achieved or not.
A: Has to be fulfilled (Verification needed).
B: Should be fulfilled
C: Can be fulfilled

1.5.1 Functional Requirements

This chapter gives the functional requirements in table format.

1.5.1.1 MES Software

Table 1.1: MES Software

ID	Name	Description	Verification	Pr.
A.01	Data collection	With arc.quire data of the factory must be demanded	Software verifies connection	A
A.02	Database	A database table needs to be created, that is capable to store all important data.	Load data into database	A
A.03	Database	Data from the factory must be stored into the database.	Request list of database	A
A.04	Dashboard Pre-processing	Data has to be filtered, sorted and allocated to a meaning	Understandable data	A
A.05	Dashboard	Data needs to be put into a dashboard. Important information about the manufactory needs to be shown.	Reviewing dashboard on screen	A
A.06	Dashboard	Pictures of the workstations can be added		C

1.5.1.2 MES External Interfaces

Table 1.2: Interfaces

ID	Name	Description	Verification	Pr.
B.01	Screen	To access the MES software a display is required.	Plug in display	A
B.02	Connection	All systems need to connect to the UTHM network	Confirmation of established connection	A
B.03	Connection	The factory data needs to be transmitted to the UTHM server	Access data	A
B.04	Connection	The arc.quire software needs to connect with the UTHM server.	Data can be loaded	A

1.5.1.3 KI Learning Factory

Table 1.3: LI Learning Factory

ID	Name	Description	Verification	Pr.
C.01	Information gathering	Course of AI learning factory should be attended		B
C.02	Doing tasks	Task during the course should be done, within the possibilities of equipment.	-	B
C.03	Manual	Creating a manual for the use of the AI factory software.	Report finished	A
C.04	Workshop	Prepare documents in which interactively the information is provided	Finished workshop	A
C.05	Connection	Understand if a connection to the factory hardware is possible	-	C
C.06	Data	AI software working principle should be understood and put into words	-	B

1.5.2 Non-Functional Requirements

Table 1.4: Non-Functional Requirements

ID	Name	Description	Verification	Pr.
D.01	Software	Usage of Arcstone arc.ops	All PCs need to be controlled to always use the latest version of the software	A
D.02	Software	Usage of Arcstone arc.quire		A
D.03	Software	Usage of Microsoft SQL database manager		A
D.04	Software	Anaconda for python for the AI factory		A
D.05	Server	UTHM server is required to be used	Connection to the server established	A

CHAPTER 2

LITERATURE REVIEW

Keeping pace with digitalization and ever-shorter innovation cycles is a major challenge for companies. Therefore, ongoing networking, knowledge and technology transfer, as well as constant competence development, are becoming increasingly important to remain competitive, especially for small and medium-sized enterprises (SMEs) with limited resources. As the objective of this project is transferring knowledge, a deeper look in the methods and benefits of the state of Baden-Württemberg [2] is taken firstly.

It is pointed out, that the exchange of knowledge and technologies between research institutions, companies, industries and sectors, as well as public administration, represents a great added value for innovative capacity. In particular, where experts from different disciplines meet, disruptive ideas can emerge and horizons can be enlarged.

Technology and knowledge transfer can be used and promoted in a variety of ways. Thanks to the digital transformation, new and more efficient ways of exchanging ideas are also emerging. Overall, a large number of different virtual and physical transfer offerings have developed that share knowledge between a wide variety of players: from tutorials and trainings, continuing education, cooperation exchanges, to science-industry working groups, crowdsourcing offerings, regional cluster initiatives, and topic- and technology-related hubs and labs, to events and trade fairs.

The state of Baden-Württemberg considers technology and knowledge transfer to be a key instrument that SMEs can use to optimize processes both within the company and beyond, and to strengthen their competitiveness through new products, services and business models. In this context, the state also provides a wide range of transfer

offers, in which up-to-date knowledge on important technologies, such as artificial intelligence, is imparted for diverse sectors, from industrial production to trade and commerce. In principle, transfer will continue to become increasingly digitalized in the future, so that virtual communication channels, platforms and formats will come more into focus. The digitization of transfer in particular will help to develop new transfer offerings beyond what has been known to date and facilitate access to important expert knowledge in a wide range of industry and technology fields, including for new user groups.

To start with knowledge transfer the theory of the project's topics needs to be explained. This will be covered by the following chapter. The first part will be about Manufacturing Execution Systems. The second part gives the basics for the Artificial Intelligence Lernfabrik.

2.1 MES

MES technology has come a long way since its inception in the 1980s and is now a vital component in the success of manufacturing companies around the world. The following chapter will cover the definition of a MES, its functionalities and how it is embedded into an enterprise. After that the role of MES in the context of Industry 4.0 will be evaluated. Lastly, the used MES for the project is reviewed.

2.1.1 Functionality of a MES

MES is designed to improve operational efficiency and increase overall productivity by giving manufacturers insight into their production processes and providing them with tools to help optimize workflows. This is achieved by monitoring all aspects of the production process in real time. That is why the production process has to be fully digitized before a MES can be applied. In the best case all sensor data and actuator states are available permanently, otherwise additional data has to be provided manually.

Kletti [3] describes the functionality of a nowadays MES by the combination of three functional scopes. Those areas have to work hand in hand to create a high-performance MES and enable a well-timed and effective manufacturing control.

First there is the function group production, where data acquisition plays the most important role. For production, data order and person related times and quantities are recorded, but also machine data is collected to manage machines and other operational resources. Moreover, operational data such as order and personnel timing and amounts are acquired. Other possible applications of a MES in the connection with production are tool and resource management, where tools and other auxiliary materials are managed, material and production logistics, where information about currently circulating material is provided, energy management and many more.

When it comes to human resources a MES should be able to handle time recording, time management, personnel resource planning and wage calculation. Furthermore, short-term manpower planning and escalation management can also be use-cases for a MES.

For the third field quality assurance different mechanisms like production inspection, complaints management, testing and measurement data acquisition can be implemented into a MES.

Lots of associations have developed standards to describe the requirements of a MES. As defined by the VDI (Verein Deutsche Ingenieure, *engl*. Association of German Engineers) in the guideline VDI 5600 [4] the tasks of a MES are the following:

- Detailed planning and detailed scheduling control
- Operating resources management
- Material management
- Personnel management
- Data acquisition and processing
- Interface management
- Performance analysis
- Quality management
- Information management
- Order management
- Energy management

Furthermore, the context of the MES in the complete enterprise is described in these standards, where the attempts in the VDI 5600 and the ISA 95 [5] are similar with three levels stacked on each other.

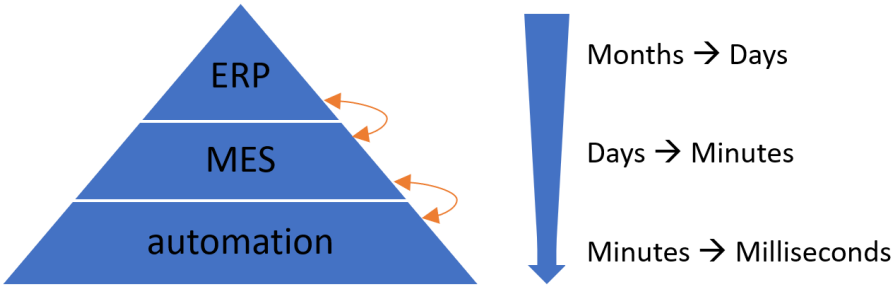


Figure 2.1: Architecture of an enterprise with MES

As shown in figure 2.1 the MES is located in between the enterprise resource planning (ERP) and the automation on shopfloor level. Furthermore, the time horizons of the different stages get significantly shorter from top to bottom. As in the ERP long- or mid-term decisions are made, the time period of interest varies from months to days, whereas for the MES single days to minutes matter. For the shopfloor level, where the actual production happens, it can even depend on milliseconds.

Since the MES has interfaces with lots of other systems, it acts like a hub for the data. The ERP provides the base data such as orders, quality requirements and capacity planning that needs to be saved and processed in the MES. But the MES also has to receive data from the production on shopfloor level including sensor values, process data, machine status, counter ticks or measurements that get processed to business relevant units. It is also possible that the MES provides specific data for the production, for instance process value specifications, target values or recipes. When it is not possible to access these data automatically, workers have to insert them manually. When the MES is seen as a hub, it is important that it is integrated horizontally as well as vertically, which means that the information is distributed on the MES level and also throughout the different levels of the pyramid.

When using a MES, a database, either an external one or the MES itself, with high requirements is needed. Data has to be consistent, plausible and complete, which is also relevant for data acquisition. Furthermore, the database has to meet various security aspects so that the production is not endangered in case of power outage and other problems.

2.1.2 MES in the Context of Industry 4.0

The term Industry 4.0 has been circulating in the manufacturing industry for a while, but there are different approaches to revolutionizing industrial production. It is agreed that the use of information technology and increased networking can significantly improve the efficiency of industrial production, potentially increasing productivity by more than 30%. However, achieving productivity gains requires the use of applications that take advantage of these new technologies and changes in organization such as MES.

Many visionaries dream of the ultimate networking of all resources and systems in manufacturing, where each machine is aware of its capabilities and each material knows which product it will become and for which customer it is destined. While this vision has evolved, a more practical approach seeks for each resource to be able to communicate with every other resource, allowing for early error detection and an increase in the flexibility and intelligence of the machines. Decentralized decision-making will become more common, but human experience will always be needed for final decision-making. This vision is known as a Smart Factory.

The concept of a Smart Factory presents new requirements for manufacturing software such as MES. IT systems need to become more flexible in order to adapt to short-term changes in manufacturing processes easily. The increasing interconnectivity generates huge amounts of data that require semantic structuring to become valuable. This "Big Data" must be processed in real-time to derive intelligent and useful insights. Industry 4.0 has to include human involvement and concrete use cases for optimal success.

The basic principles of a MES as defined in VDI guideline 5600 provide a good starting point for the industry 4.0 era. In particular horizontal integration and comprehensive penetration of the company are the most important issues here. The collection of data in production, their aggregation and also the processing and displaying in real time are essential for the success of a smart factory. However, there is much more to it than the processing of data and measured values into meaningful information. In the future modern companies will depend much more on the integration of all functions and modules across all elements involved in production. In addition, the importance of cross-system data exchange will increase, which is therefore another task a MES must fulfill.

In summary, it can be stated that MES systems would be ideally suited as a central information and data hub in a smart factory. But in order to do so perfectly today's systems still need to be further developed. Still the basic approach of the MES idea is already heading in the right direction. The goal must be to synchronize all systems and functions involved in production. Transparent data exchange is the basis for a functioning future scenario. This does not just mean the pure distribution of data, but also application-related preprocessing or aggregation. For example, ERP systems only need final summary values and not comprehensive detailed data from the individual status messages at the machine. An MES system, as the central hub, has the task of providing each user and each system with exactly the data that is expected or required.

The developments and enhancements of the MES 4.0 concept aim to satisfy market and customer demands, such as mass customization and resource efficiency, to ensure manufacturing companies remain competitive in an increasingly globalized market. MES 4.0 provides flexible software tools to support the individualization of mass-produced products and improve efficiency, which is crucial for maintaining and expanding market leadership in a high-wage, highly automated region. The concepts of Industry 4.0 and MES 4.0 will promote sustainable efficiency in manufacturing in the long term.

2.1.3 Arcstone’s MES solutions

Arcstone enables all enterprises to unlock the value of data [6] by making it visible and traceable. To do so they provide the MES arc.ops that comes with 20 modules. It features a specific tool for connecting to machines and pull the data named arc.quire, which works with industries standard protocols, including OPCUA and is also capable of manual data input. As a hardware integrator it writes the data into a SQL database inside the MES, from where arc.flow can distribute the data further to an ERP or other top level planning systems. A complete overview of Arcstone’s MES solutions is displayed in figure 2.2. On the right side the gateway to the different stakeholders inside an enterprise is shown.

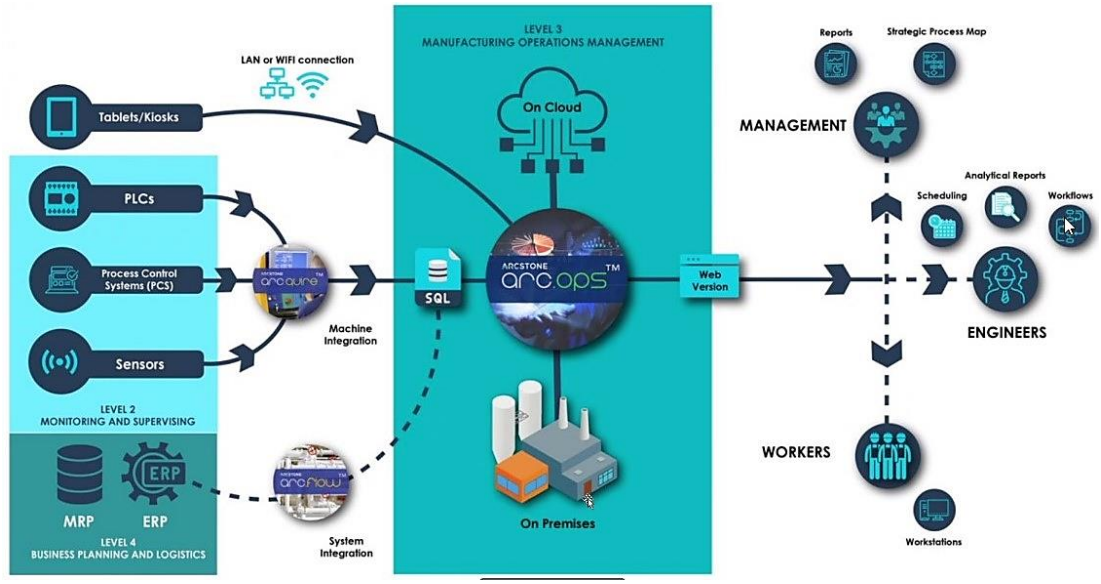


Figure 2.2: Arcstone's MES solutions [6]

When looked at Arcstone's MES solutions in the context of the ISA 95 framework, arc.quire connects the automation level to the MES, arc.ops is the MES itself and arc.flow is the link between ERP and MES. This is shown in figure 2.3.

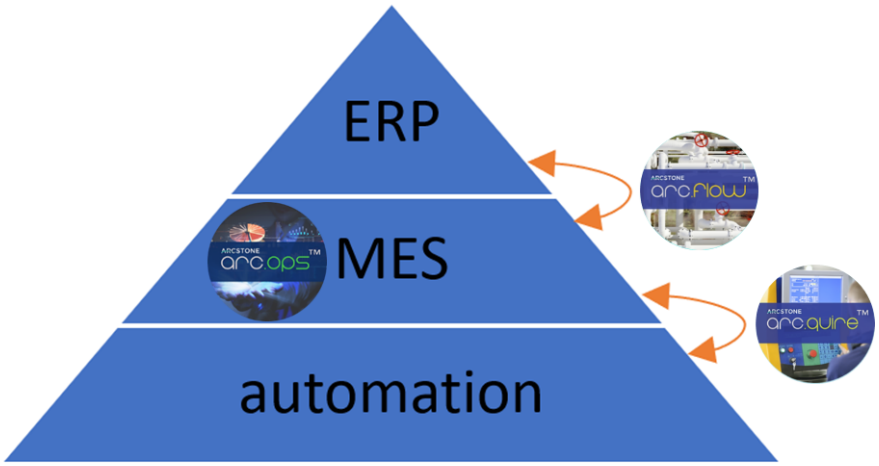


Figure 2.3: Arcstone in the context of the ISA 95 framework

2.2 Artificial Intelligence

So far, the factory is controlled by a PLC software. This software allows the developer to set clear instructions what the factory and each machine does. Instructions are worked through precisely. It furthermore needs to be controlled by workmen. The production process is started, viewed and stopped by humans. Furthermore, the quality of products needs to be screened. For a company this always means to pay wages. But how avoid these costs?

With “[the] branch of computer science that is concerned with the automation of intelligent behavior” [7], which is called artificial intelligence (AI). This could help to take over the task of the mentioned steps of work.

The factory is supposed to be monitored and controlled by artificial intelligence. While AI comprehend many ways of implementations, this project is supposed to use neuronal networks. This chapter will review neuronal networks.

2.2.1 Neuronal Network in General

When talking about neuronal network it is meant to have an artificial neuronal network. The idea is to create software that works similar like a human brain, a natural neuronal network. This network gives us our intelligence. It helps us to train our abilities, physically and more important mentally. We can adopt to situations and changes.

But how to build such a system? A computer has an architecture not matching with the brain. It is basically a calculator that can execute software which is based on mathematics. The attempt is to analyse human brain and to create models that reflect the brain.

The network consists of many small entities, that are similar like the neurons. They are working like single information storages, powered by electric voltage. As messages reach a neuron, also electrical energy is sent. The voltage level is rising. At some point there is too much energy in a neuron and it will reach out and sends its information. But where to?

“What fires together, wires together” is a quote by Hebb [8] describing that neurons often reach out to each other will connect. The more often they communicate the

stronger their connection will be. For an artificial neuronal network it should work similar. When raw data is put in a new system, there is no connection between the entities. The computer then need to find connections. Hebb is describing the idea of how it's done: Data often colliding, will have some kind of link. This is how the computer can derive clusters. Once these clusters are established it can be used to solve tasks. The more tasks it solves, the more information the system can collect. This information can also lead to an evolvement of the clusters.

Ertel [10] gets deeper into architecture of a neuronal network and provides schematics for illustration on this which are reused here.

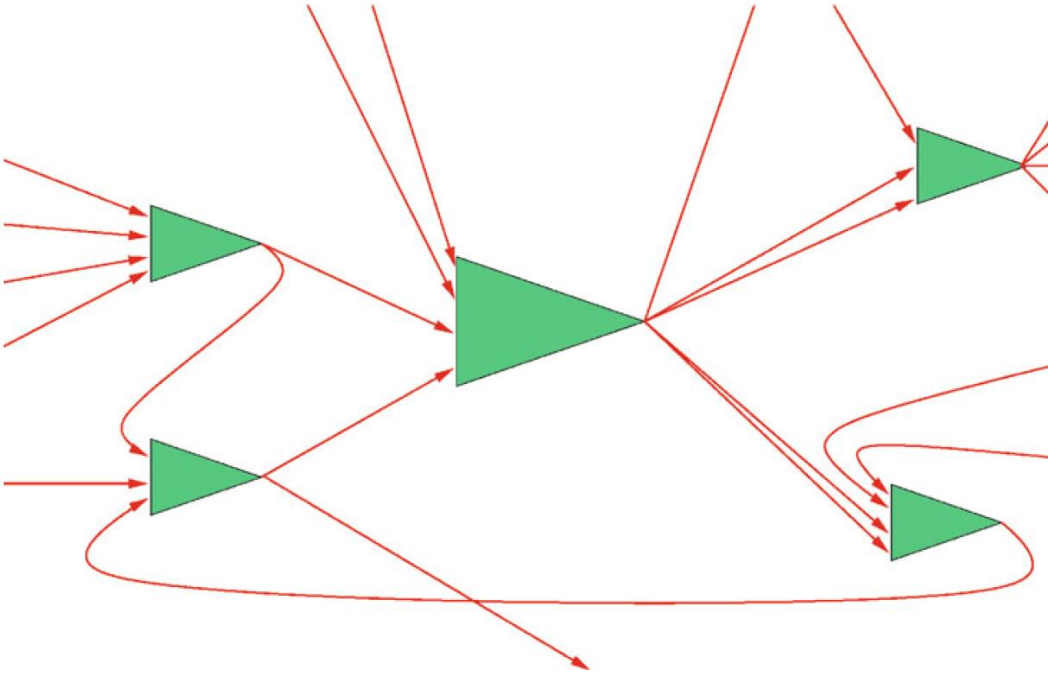


Figure 2.4: Model of network [10]

Figure 2.4 represents the network. The green triangles are the neurons connected by wires. The following shows how this is implemented.

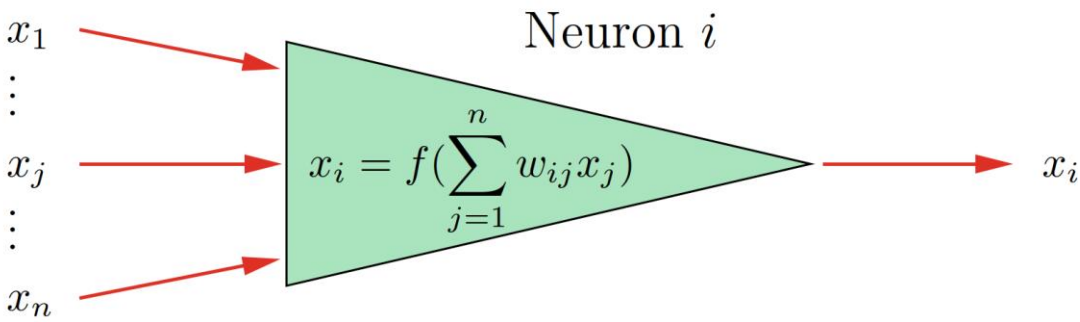


Figure 2.5: Mathematical Neuron [10]

The input voltage of each wire is represented by variables named x_j in the picture. The higher the voltage value of the human brain is, the higher the value of x_j is in the mathematical model. The factor ω_{ij} is fixed to each x_j and represents how strong the connection to the respective previous neuron is. The better the connection, the more electrical power will go through. Despite this being a factor, its value can change, if the connection is changing as result of an adapting process. The product of each input and its factor will then be summed up. This is the output value x_i.

Despite its called learning network in most networks the input data is processed through fixed calculations. In most cases it is a kind of matrix multiplication or applying filters and masks. It is also possible to perform several steps of calculation. That's why an expert is required, who knows what the network is used for. The expert can then create the processing formulas.

What is it the network is learning then? It is the adjustment of the factors x_j. When saying the network is being trained, it means that the artificial network does the calculation of the factors based on the received input data. With that the network is then able to detect patterns.

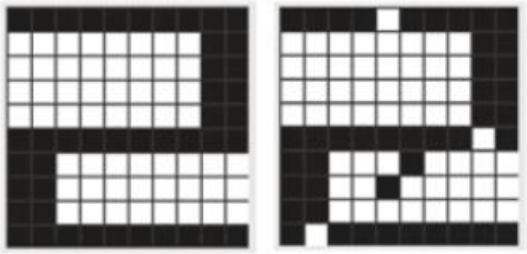


Figure 2.6: Pattern of number 2 [10]

In the above figure Ertel gives an example of pattern image recognition. The network is trained to detect numbers the number 2. The ideal picture of the number is shown in the left side of the figure. The network should then be able to recognize the 2 in other pictures. What makes a neuronal network special, that it is not just able to recognize the same image of the number, but also an image with changes. That is necessary in the real world as there are for example disturbances. This is shown on the right side of the figure. Some pixels of the 2 are not in place. If there was a standard recognition, it would fail to find the 2, but we want to recognize it as a 2. That is where the neuronal network comes in to play. What the network does is applying filters and masks to the picture. This will be done until the pattern of the 2 is recognized.

Another example is face recognition. If you take a picture of a person. The next day you want to recognize the face, it will be hardly possible. Because every day a person looks a bit different. And over the years a person's face can change immensely. The network still needs to be able to recognize the pattern of a human face. For that a series of pictures is required to enable the network to learn. It then can find similarities and use them to recognize a changed face.

The examples just considered image processing. But neuronal networks can be used in different areas. In the human brain, different parts are responsible for different tasks. So, every part is structured differently to match its requirements. For artificial networks it is similar. Different networks suit different tasks. That's why there are many different models that try to attempt to rebuild a part of the brain. Each model has its benefits an application its suits. For this project it will be the convolutional neuronal network and the recurrent neuronal network.

2.2.2 Convolutional Neuronal Network

The issue of most networks is that they have a vast amount of data as input. Processing all of it in just one step would take very long. This is addressed by having several steps of processing. In most networks the first step is to extract attributes. These are often referred as preprocessing layers. They often consist of filters, similar like described above and capable of learning. With every layer different things can be extracted. If you take image processing as an example, there is one layer to recognize edges of the picture and a layer to extract a face. In most cases the more layers you have the more attributes you can extract. In a second step attributes will be interpreted.

In a convolutional neuronal network (CNN) most of the filters are based on the mathematical convolution. The advantage of the convolution is that it reduces the number of the factors. This helps fasten up the calculation operation.

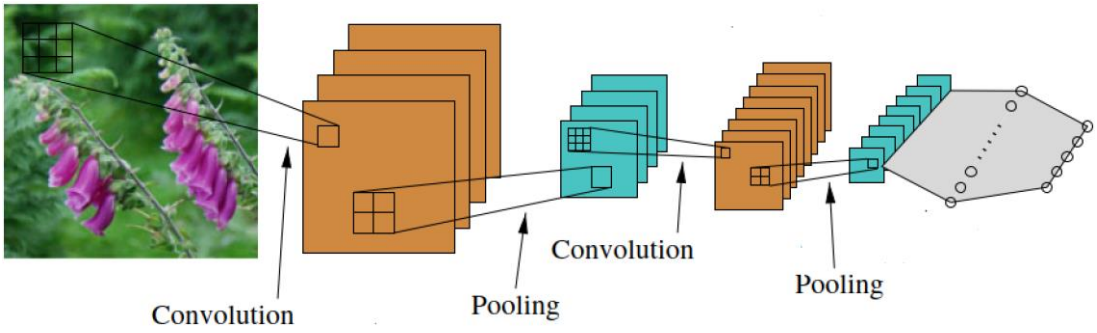


Figure 2.7: CNN working principle [10]

The figure 2.7, Ertel visualizes the working principle of the filters for image processing. The input is a picture of flowers. The picture then will be filtered by using the convolution. The result is a smaller picture with highlighted attributes. The next step is pooling. Information that are similar get bundled here. This data reduction helps to safe storage space and ultimately processing time. Repetitions of the two steps will be executed. The goal is to have a smaller set of extracted data.

The whole process is complex. Several mathematical functions need to be used the right way. Additionally, as there are less factors, they need to be more precise. This is leading to long training times. It also requires more validation, to check if the learned factors are precise enough. Otherwise, a to big error in factor value can lead to misinterpretation.

Nowadays, scientist and engineers need to have a huge knowledge about every aspect of the network to make it work correctly. It often requires a lot of experience to know which filters to use and how to order them. Also, on the mathematical side expertise is required. The filters can consist of many different mathematical operations. This can also lead to long development times.

The results of correct implemented convolutional neuronal networks are satisfactory though. For example, face recognition has not been possible with classical computation. Despite it is easier to imaging this kind of network is used for image processing it is also possible to use it in other branches. It just gets more abstract, as the constructor of the network needs to be able to create filters that work for data sets that come as example form a factory.

2.2.3 Recurrent Neuronal Network

The second network that this chapter covers is a recurrent neuronal network. Compared to the human brain this is working like the short-term memory. It is capable of remembering data input for a while and even take it in consideration for new learning processes.

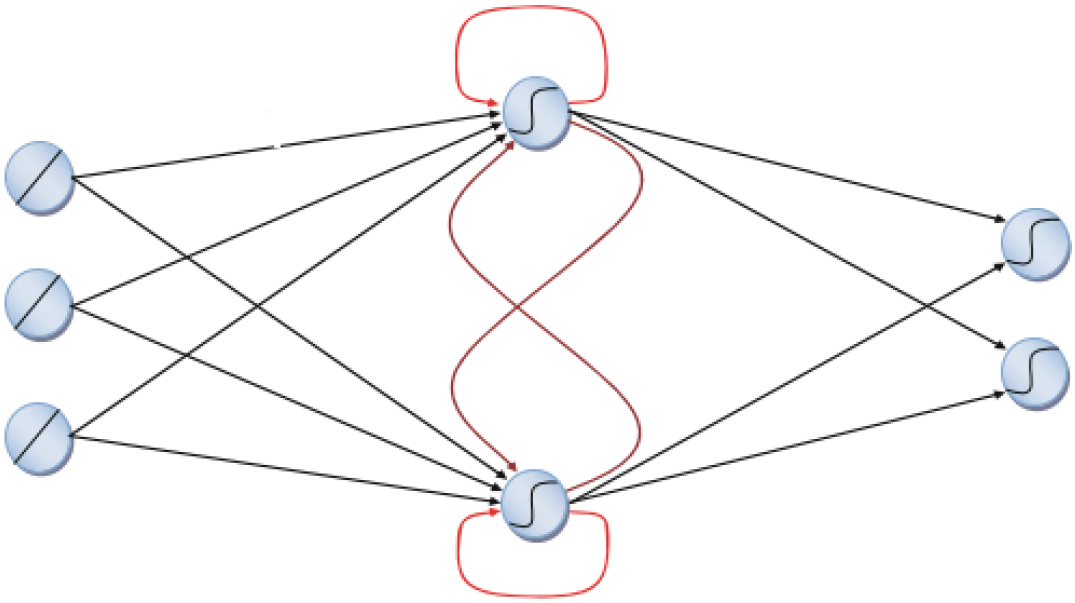


Figure 2.8: Recurrent Neuronal Network [11]

Figure 2.8 from Grum [11] is a simplified visualization of the working principle. The blue points represent the neurons with connections between them. The processing order is from left to right. This kind of network is special, because of the red connections. The dark red connections are between the same layer level, the brighter red connections are to a neuron itself. This can lead to processing of a specific information for several times. It can even be transferred between different neurons of a layer for several times, which makes an information available a longer time. This construction of a network is the counter part of the human's short-term memory.

2.2.4 Summary Neuronal Networks

For humans it is often easy to acquire a series of data. The challenge of artificial neuronal networks is to provide the computer with the information. It is also required to process as much raw data as possible, without giving many rule bases. The result

should be software that is capable of learning from data, extract patterns and recognize them on new data.

Neuronal networks still have a lot of potential to be unlocked in future. Scientists researching on the matter for a while and still will. The main issues are that the inspiring model, the human brain, has a huge size with approximately 10^{11} neurons and that its hardly possible to access the human brain. But with ongoing research artificial neuronal networks will grow in power.

CHAPTER 3

METHODOLOGY

3.1 Flow Chart

In this chapter two flow charts will explain the execution of the project, divided into the topics MES (Figure 3.1) and Lernfabrik (Figure 3.2).

After the start of the MES project the objectives and requirements were defined. They serve as a guideline for the following parts of the project. To have basic knowledge about the subject, a literature review was performed, followed by the implementation, with the steps to collect the data from PLC to UPCUA server, fetching the information from the server and writing them into the UTHM database and finally to create a dashboard. After these steps are completed, the fulfilment of the requirements needs to be checked. If a requirement is not fulfilled, the above-named implementation steps will be repeated. As soon as all requirements are met, the report can be created and submitted.

The project AI-Learning-Factory begins with a definition of the project's objectives and requirements, which serve as a guide throughout the project. They also need to be checked at the end of the project as the penultimate step. If the requirements are not fulfilled, all previous steps need to be checked until the requirements are met. Then the report can be written. After the introduction, a literature review about neuronal networks were conducted. This is followed by the implementation of the project. Steps that are performed are named in the following text. The first step here is to understand the learning factory by doing the AI-Campus course. After that several task branches open. The left one is about reviewing and translate the AI-Lernfabrik course. The middle branch includes understanding the Lernfabrik tools,

perform the course tasks and write a documentation for the solutions. The first two branches then combine again as the analysation about the learned is required. The branch on the right side includes the analysis of the simulation hardware and to check if an implementation using the AI-Lernfabrik software is possible. In the final project execution step, the information of all branches is collected and the requirements are checked. As soon as the report is finished, it will be submitted.

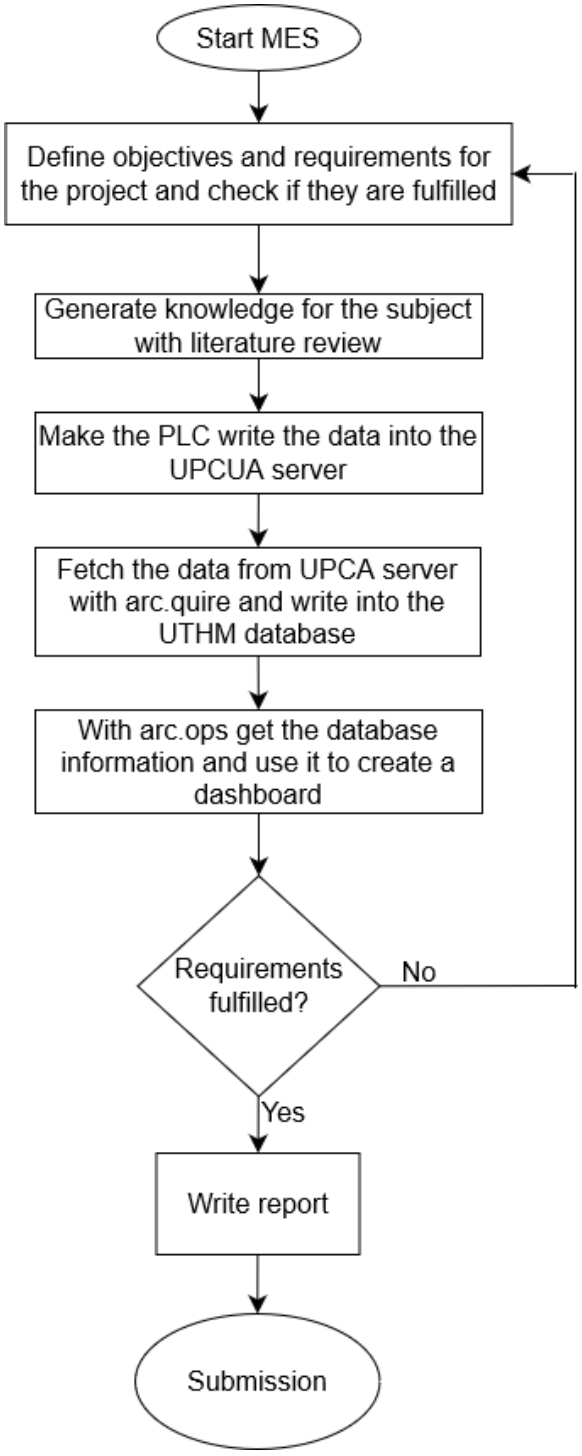


Figure 3.1: MES Flowchart

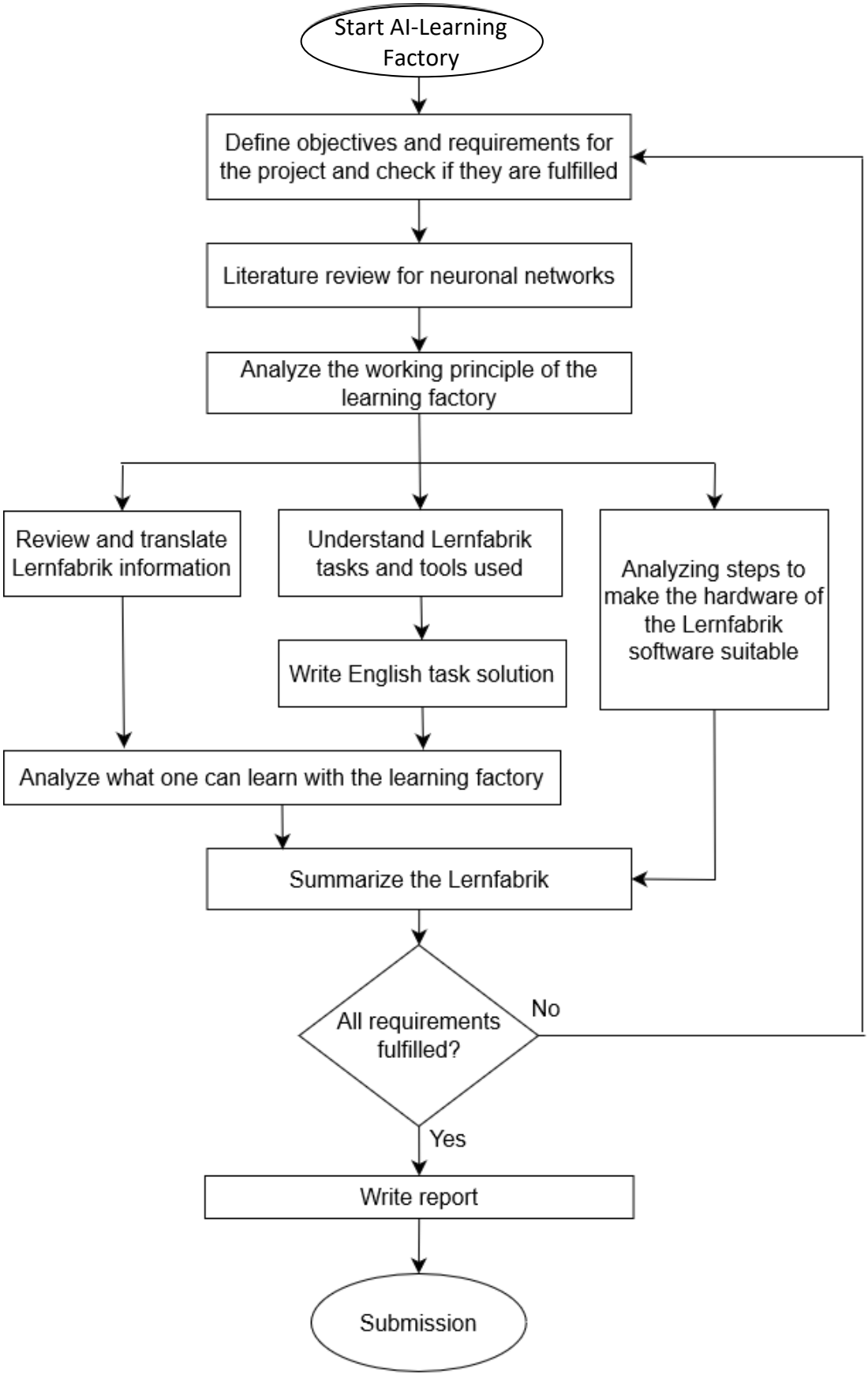


Figure 3.2: AI-Learning Factory Flowchart

3.2 Previous State of the Learning Factory

In this chapter the setup of the learning factory in the laboratory is described and it is shown from what basis the project has started. This is done to clarify the existing problems in an already digitised factory.

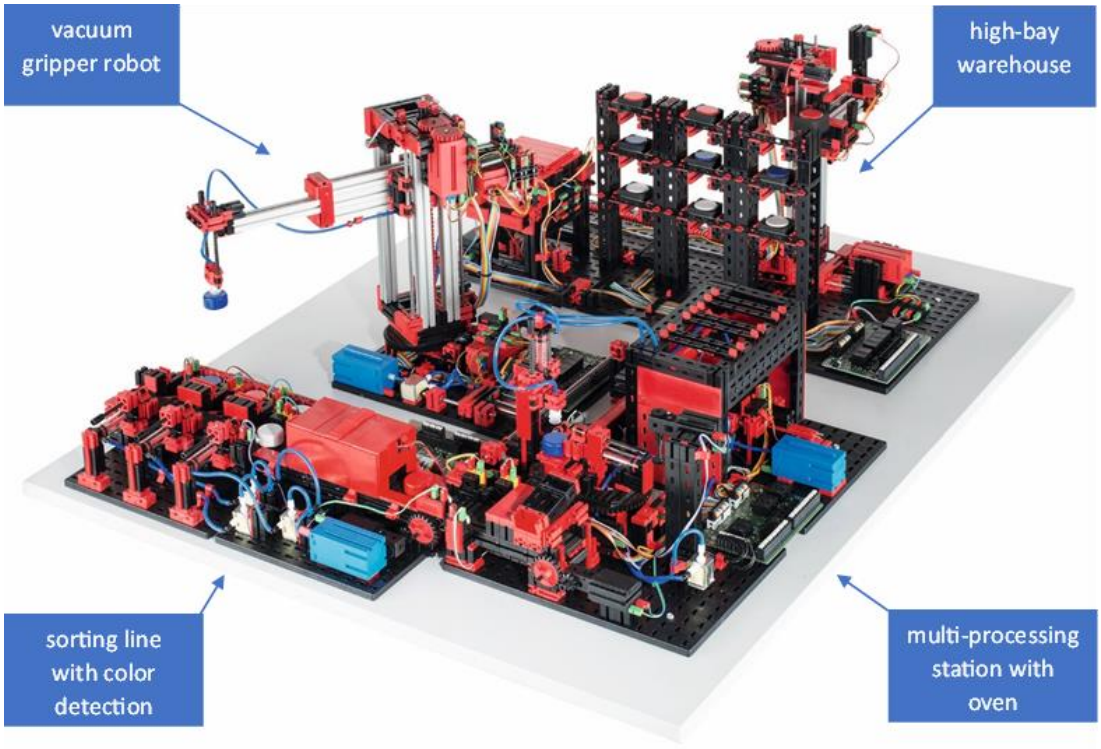


Figure 3.3: Fischertechnik Smart Factory [13]

The Fischertechnik Smart Factory [13] is used in the project as depicted in figure 3.3. It is a learning factory that comes with many features and in the following a typical process is described.

When the process is started, the transport arm of the high-bay warehouse (HBW) moves to the storage system, picks up the wanted workpiece and places it in the output station. There it is conveyed to the pick-up position of the vacuum gripper robot.

The vacuum gripper robot (VGR) picks up the workpiece from the output station of the high-bay warehouse (HBW) and places it on the slide of the oven of the multi-processing station with oven (MPO). There the workpiece is carried in, fired and moved out again. The transport carriage with vacuum suction pad then transports the workpiece to the milling machine. There, the workpiece is placed on the rotary table and is milled.

After the milling process, the workpiece is pneumatically pushed onto the conveyor belt of the sorting line with colour detection (SLD). On the conveyor belt, the workpiece passes through a colour recognition system. Depending on the colour detected, the workpiece is pneumatically pushed onto the corresponding chute. Figure 3.4 illustrates the schematic plan of the learning factory.

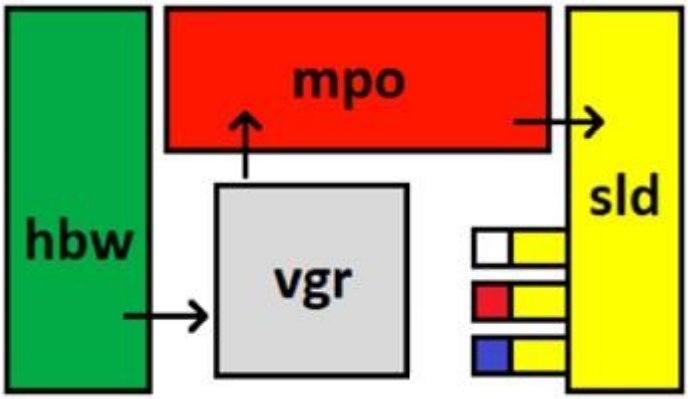


Figure 3.4: Schematic Plan of the Learning Factory [15]

As for the start of the project the learning factory is connected to a programmable logic controller (PLC) from Beckhoff and controlled with their software TwinCAT 3. This does not fulfil all the requirements of a factory in the context of Industry 4.0 as described in the previous chapters. Therefore, the projects deal with making the learning factory ready for the future.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Dataflow from Automation Level to MES

In this chapter the data flow from the automation at shopfloor level to the MES software is described. For that purpose all machines need to be digitized and sensorized in order for a PLC to control the production steps and read the input and output data. With this done the data can be written into a server, which is then also accessed by the MES, where it can be processed, analysed and displayed. Figure 4.1 illustrates the dataflow from the automation software TwinCAT to the MES arc.ops and where the data is stored during this process.

Figure 4.1: Dataflow from the automation software to the MES

In the following subchapters each connection step is evaluated in more detail. First, the connection of the PLC with the server is described. Then the linking of the server data to the MES using arc.quire is discussed and lastly, the access of the data inside the MES is elaborated.

4.1.1 Architecture of the Connection between Automation Software and Server

One of industries standard protocol that is widely used in automation is OPCUA, featuring flexible data transport across platforms and making machine data readable for other applications. Furthermore, it allows the user to easily write data into an OPCUA server.

When it comes to connecting the PLC to the OPCUA server there are three components that must be arranged and connected correctly. The TwinCAT environment controls the PLC and the input and output data of the PLC is written into the OPCUA server. The OPCUA expert can be utilized to test the connection and view the data. On the device that hosts the OPCUA server, TwinCAT XAR (runtime only) or TwinCAT XAE (runtime and engineering) has to be installed as well as the function TF6100 OPCUA [14]. On each PC where a TwinCAT software product is used, the license TF6100 has to be activated. When setting up an OPCUA server authentication can be implemented by defining an username and password.

Figure 4.2 illustrates possible architectures of this system. Firstly, it is possible to run all components on one PC, which is obviously the easiest method. If it is wished to distribute the components on more than one PC, there are several ways to do so. For example, the TwinCAT environment with the PLC and the OPCUA server can be run on one PC, while the server data is accessed from another PC. Basically, every combination of the elements is possible, but it is notable, that a functional TwinCAT environment supporting the operation of the PLC is always on one device.

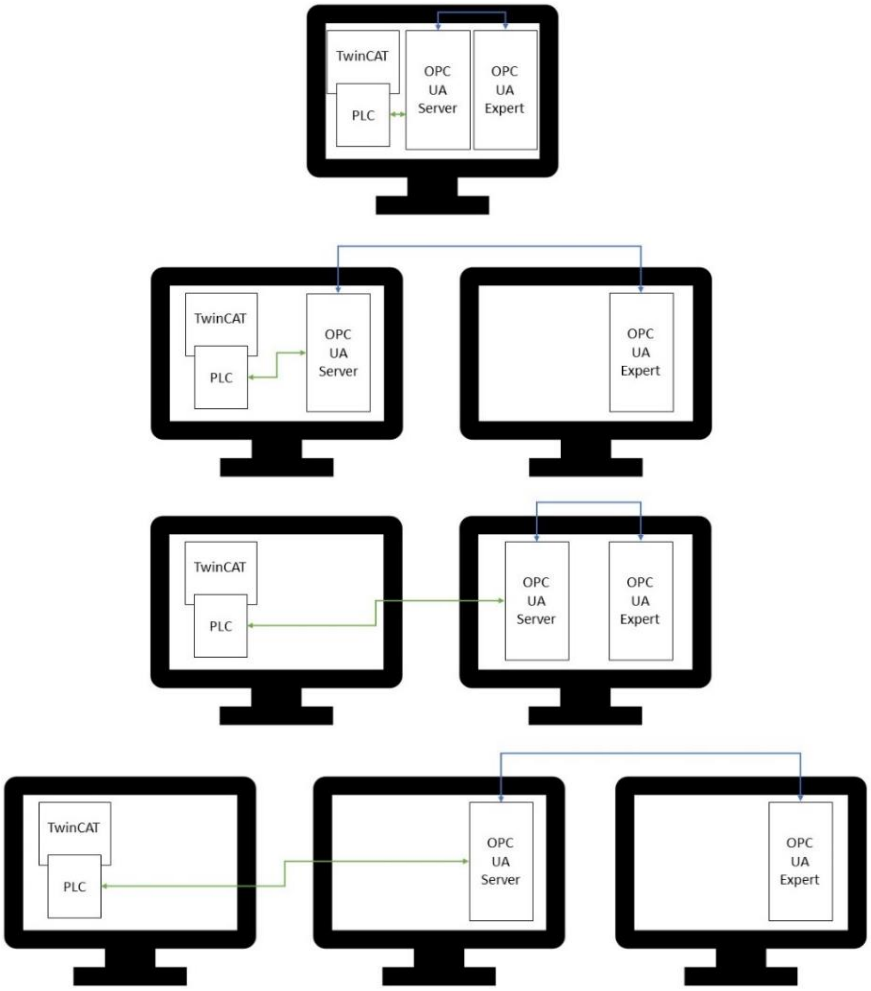


Figure 4.2: Architecture examples of the connection

Another way of connecting the components, when they are distributed on two PCs is shown in figure 4.3. The left PC is the engineering computer, where the PLC program for the server is developed and the right PC is an industrial computer from Beckhoff. Here TwinCAT also has to be installed, but functions only as a runtime for the PLC. Furthermore, the OPCUA server is hosted on the industrial PC. The access to the server data happens from the engineering computer with the OPCUA expert. In this architecture it is important that a valid OPCUA license is activated in each TwinCAT software product.

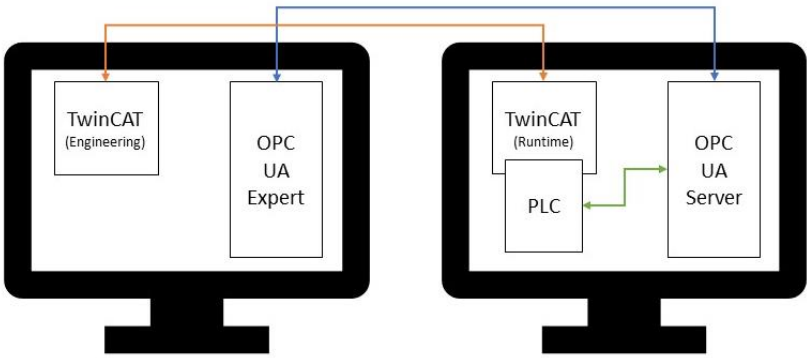


Figure 4.3: Used architecture for the learning factory

4.1.2 Pull Data from the Server to the MES Database

Once the OPCUA server is set up, the data can be utilized in the MES. A new arc.quire profile needs to be set up in order to pull the data to the MES. A meaningful name is chosen and a description of the profile can be added in the notes. Next, the data provider and export types have to be selected, whereby OPCUA is the provider and Database is the export for the learning factory. Refer to Figure 4.4 for the example settings.

In the following step the data source has to be defined as illustrated in Figure 4.5. The OPCUA server URL has to be inserted as well as the publish interval and the authentication credentials if required. Afterwards, add the data that is wanted for the MES by giving each a meaningful name and inserting the corresponding node ID. If a test connection is running, the current data value is displayed in the right column.

Before the profile is ready to run, the data destination has to be specified, which is shown in Figure 4.6. Most of the fields here are predefined and there is no need to change them. But the connection string has to be modified to match the username and password of the Arcstone server.

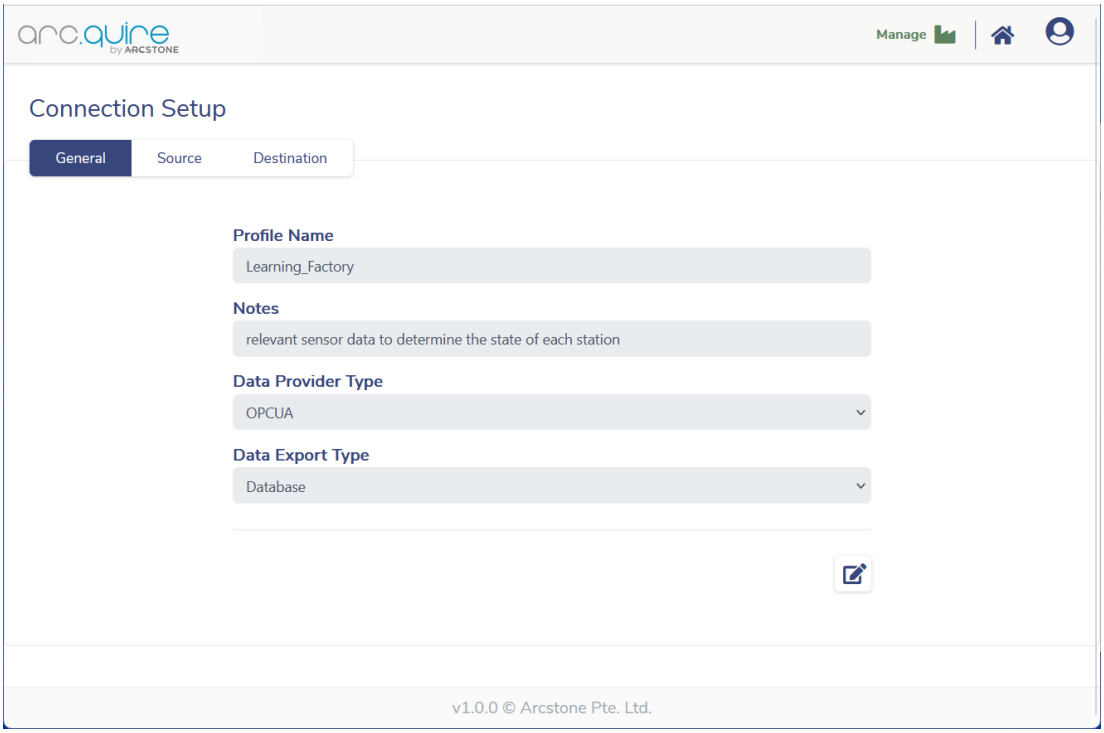


Figure 4.4: General connection setup of arc.quire profile

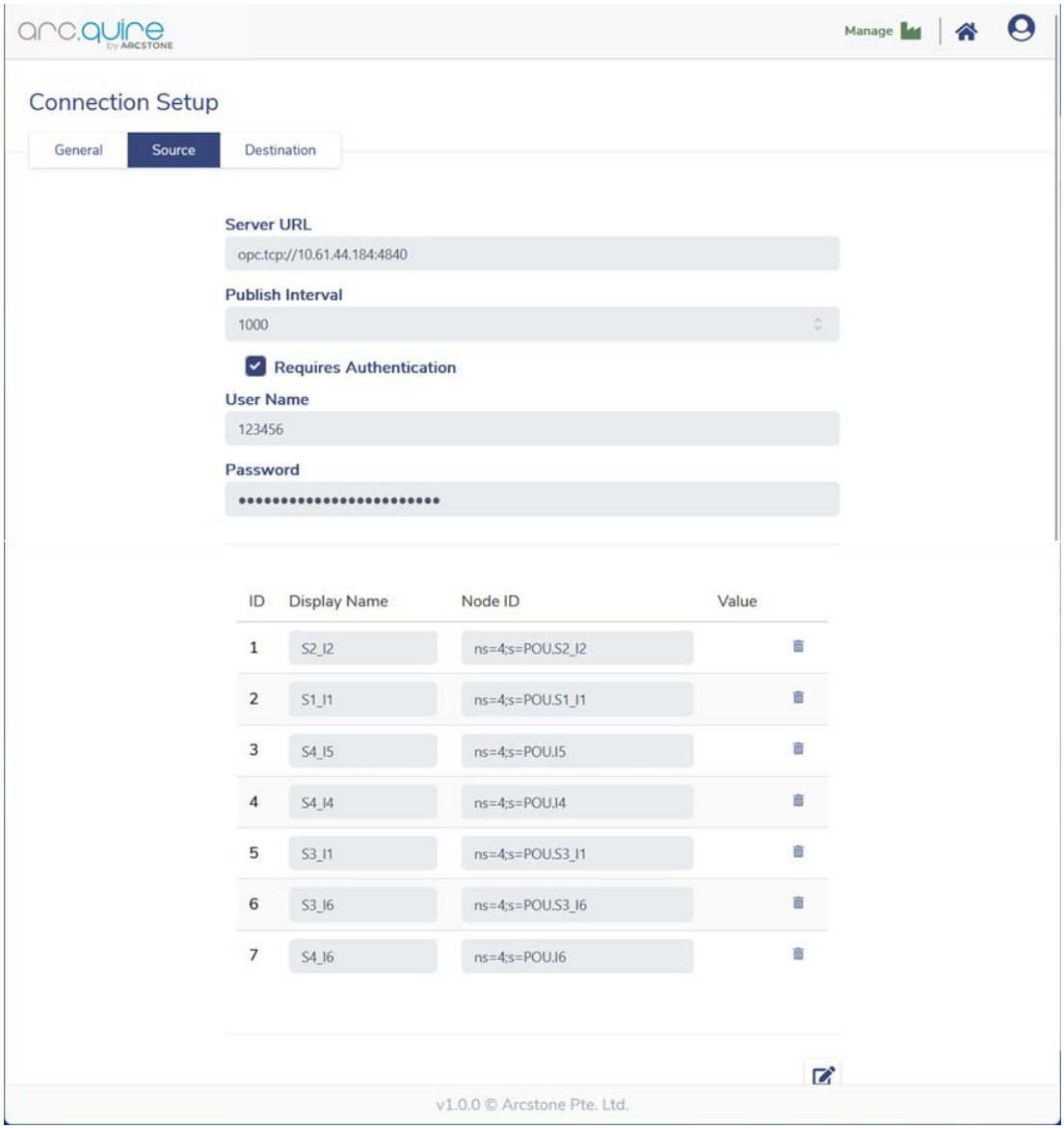


Figure 4.5: Source setup of arc.quire profile

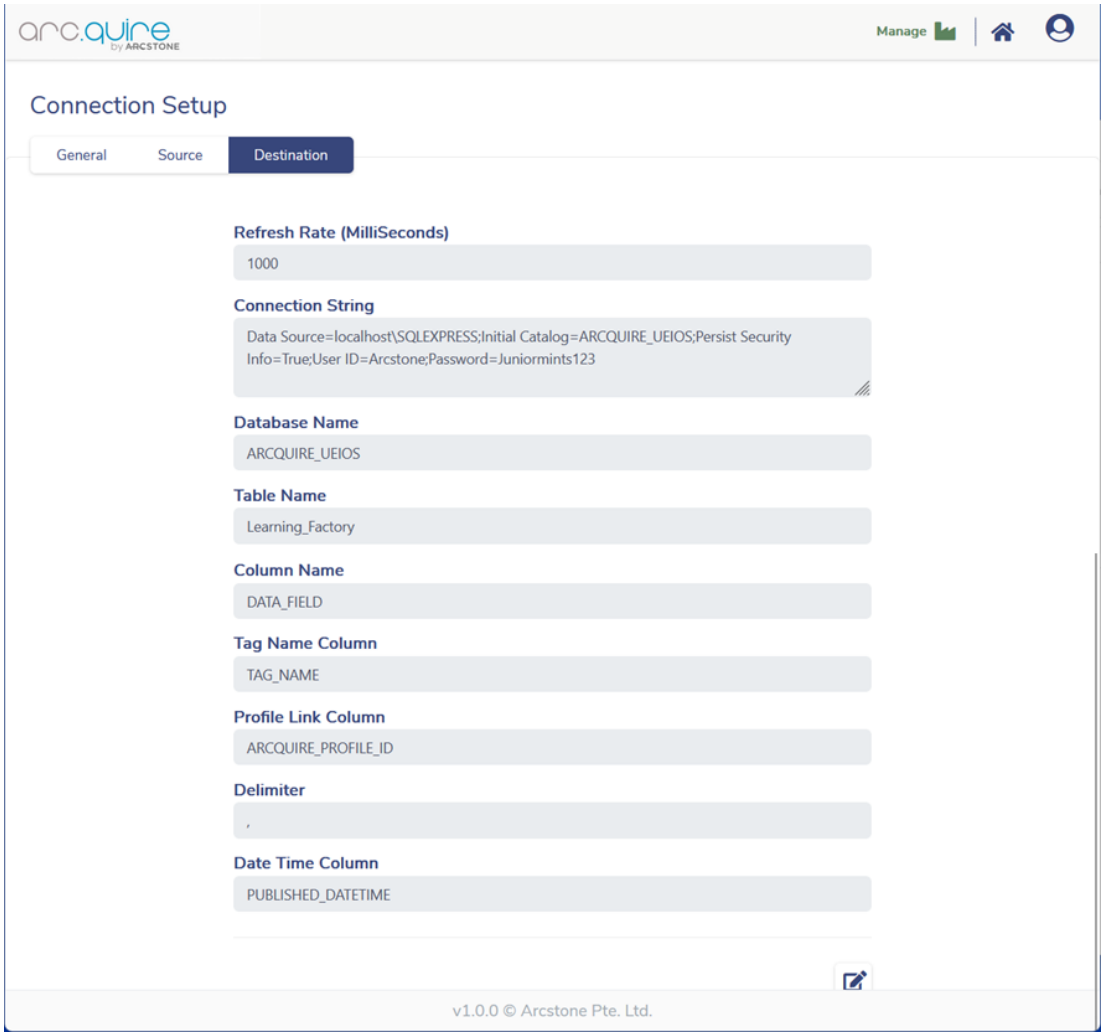


Figure 4.6: Destination setup of arc.quire profile

With that done, the profile is ready to be started by clicking on the play button and once it is running the selected data is automatically read from the OPCUA server and written into the SQL database. To view the database in the Arcstone server of the UTHM Microsoft SQL Server Management Studio can be used. Figure 4.7 contains a screenshot of the Microsoft SQL Server Management Studio where the first 1000 elements of the database are viewed.

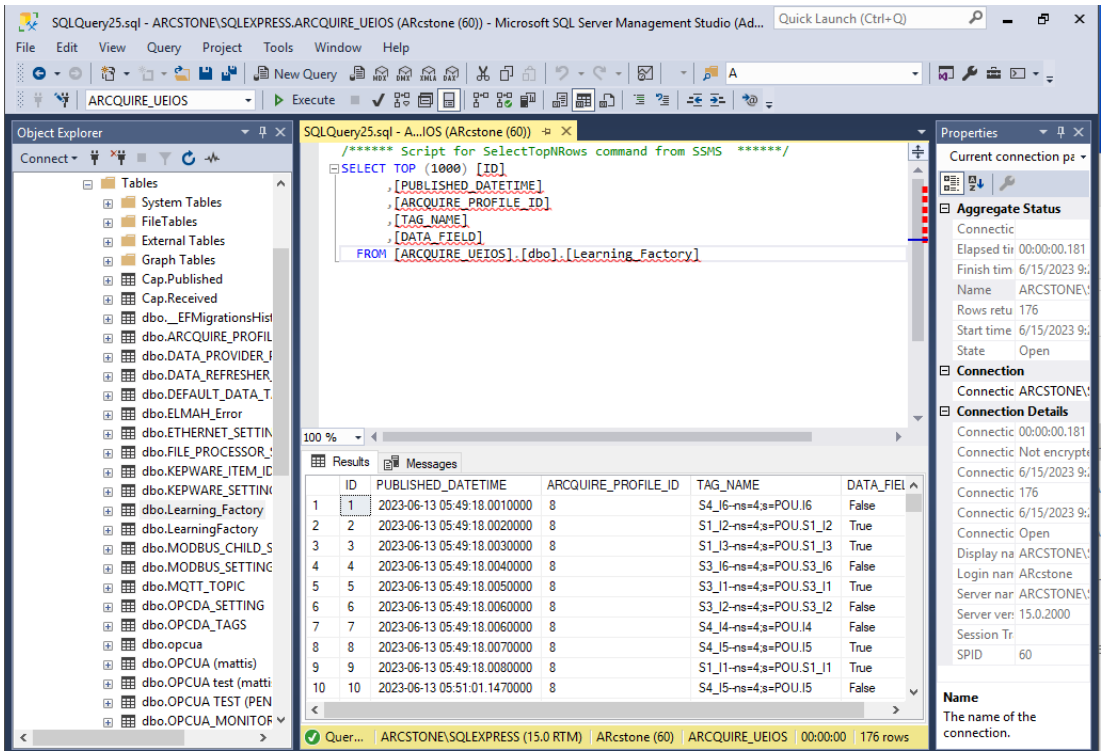


Figure 4.7: Microsoft SQL Server Management Studio

4.1.3 Access Data Inside the MES

As the data is now available for Arcstone's MES arc.quire a dashboard for visualization is created. When a new dashboard is made, the first thing to do is adding a data source as shown in figure 4.8. Insert a name, a short description and the connection string from the destination of the arc.quire profile that was created earlier.

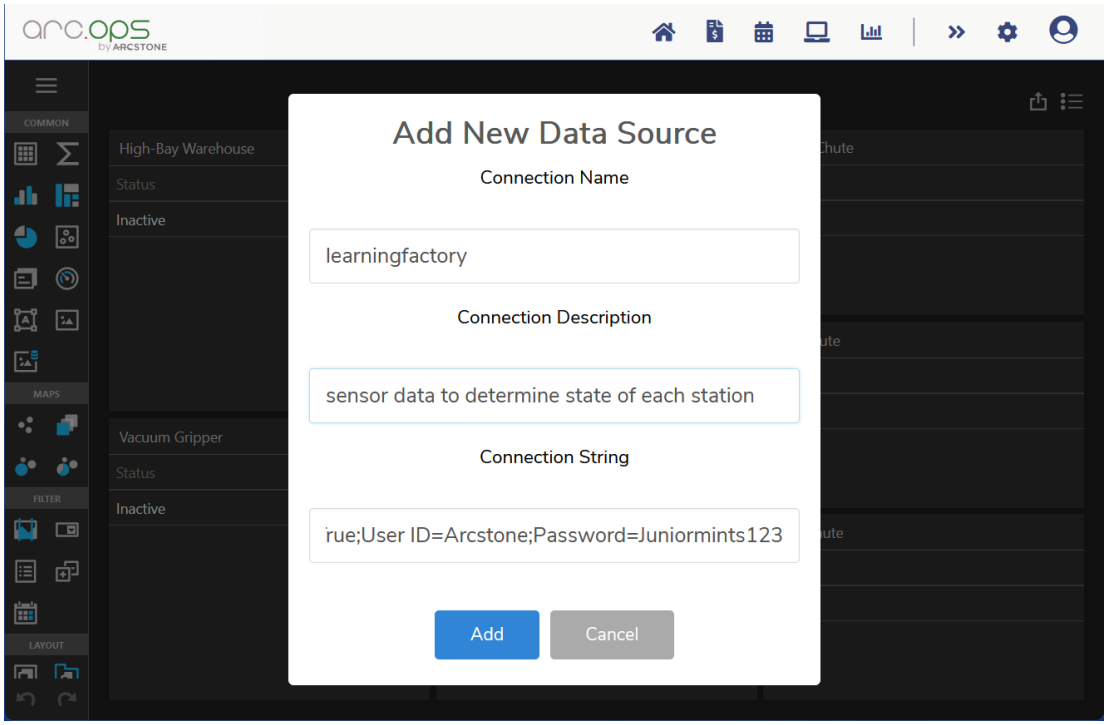


Figure 4.8: Add new data source to a dashboard

Navigate to data sources, which opens the dashboard data source wizard and create a new source. The data is stored in a database, so this source type is selected. Choose the previously added data source as connection. Figure 4.9 shows the window where a query has to be created. The query builder can be utilized to simplify the process.

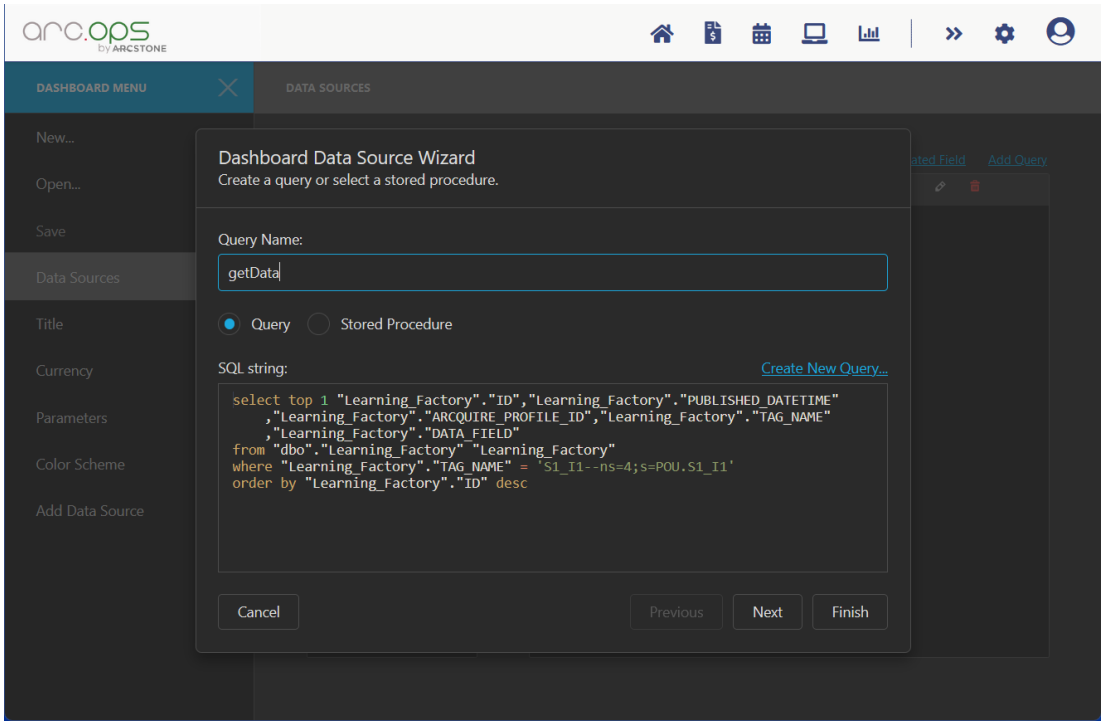


Figure 4.9: Dashboard data source wizard

By using drag and drop the required table can be selected as well as the columns. The SQL string can then be modified to fulfil the users demands. In the shown example only the newest value of S1_I1 is selected by filtering for the TAG_NAME and putting the ID in a descending order. Other methods of an SQL query can also be used here. Now the data is ready to be displayed on the dashboard.

For that purpose, Arcstone offers a variety of options. The most common and easiest way of displaying data is the grid, which is also used for the Learning Factory, but there are also elements like the bar graph or the pie chart. The grid is basically a table, where data from the data sources can be added in the columns. Figure 4.10 shows the settings for a grid. The data either be directly displayed or processed further by adding a calculated field. This is useful if the output needs to be normalized.

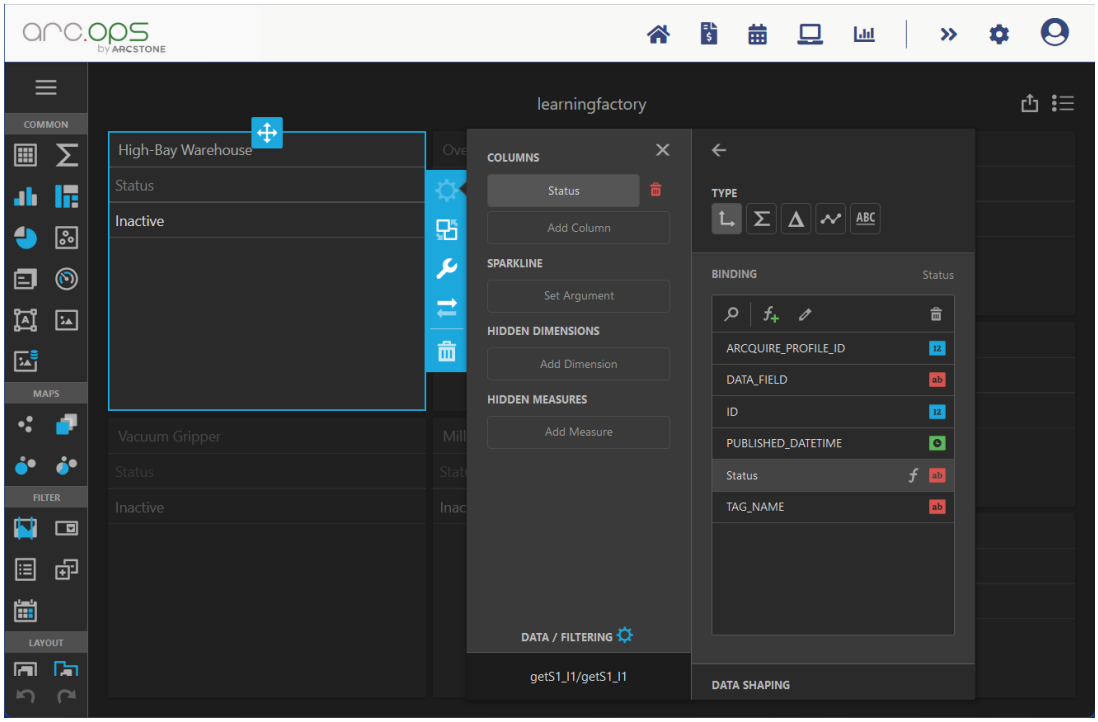


Figure 4.10: Bind data to grid

4.2 Creating the Dashboard

The dashboard should clearly visualize which station is currently running. To achieve this, appropriate sensors that indicate the status of each station have to be selected.

The first station is the high-bay warehouse. Here the end-switch S1_I1 of the transportation arm is utilized to determine whether the station is running or inactive. If the switch is pressed the sensor value equals true and the station is inactive. As soon as the transportation arm starts to move to the storage system the switch gets released and the sensor value turns to false. When a workpiece has been taken out of the storage system and is put on the output station the process is finished and the end-switch is pressed again.

The vacuum gripper robot then picks up the workpiece from the output station. To reach the pick-up position the robot needs to extent its arm and the end-switch S2_I2 is released. After putting the workpiece on the slide of the oven the arm returns to its initial position and the end-switch is triggered again. Just like for the first station a sensor value that equals false means that the robot is running.

The multi-processing station is divided in two separate stations for the visualization. Starting with the oven, which is running as long as the end-switch of the slide S3_I6 is pressed. In contrast to the previous stations the logic is inverted here, meaning that a true sensor value stands for a running oven. When the workpiece leaves the oven, it is transported to the milling machine. The end-switch of the rotary table S3_I1 indicates the state of this station. If the milling process is running the switch is released, thus its sensor value is false during this time.

The state detection of the sorting line with colour detection differs from the previous stations as it is not determined if it is running or inactive. Instead, the state of each chute is derived from the sensor values and the dashboard should show if it is empty or full. The light barriers S4_I4, S4_I5 and S4_I6 return true if the chute is empty and true if the workpiece is sorted.

A summary of the sensor selection and their meaning for the states of the stations and the chutes can be found in the following table 4.1.

Table 4.1: State of each station and the chutes

Station	Sensor	State	Sensor value
High-bay warehouse	S1_I1	Running	false
		Inactive	true
Vacuum gripper robot	S2_I2	Running	false
		Inactive	true
Oven	S3_I6	Running	true
		Inactive	false
Milling machine	S3_I1	Running	false
		Inactive	true
White chute	S4_I4	Full	false
		Empty	true
Red chute	S4_I5	Full	false
		Empty	true
Blue chute	S4_I6	Full	false
		Empty	true

As not all sensors have the same meaning the sensor value needs to be modified, which is achieved by adding a calculated field with a simple if-query as shown in figure 4.11. The first input in the brackets of the if-query is the statement that is checked. If the statement is true, the second input is returned and else the third input is returned. The example shows the calculation for S1_I1.

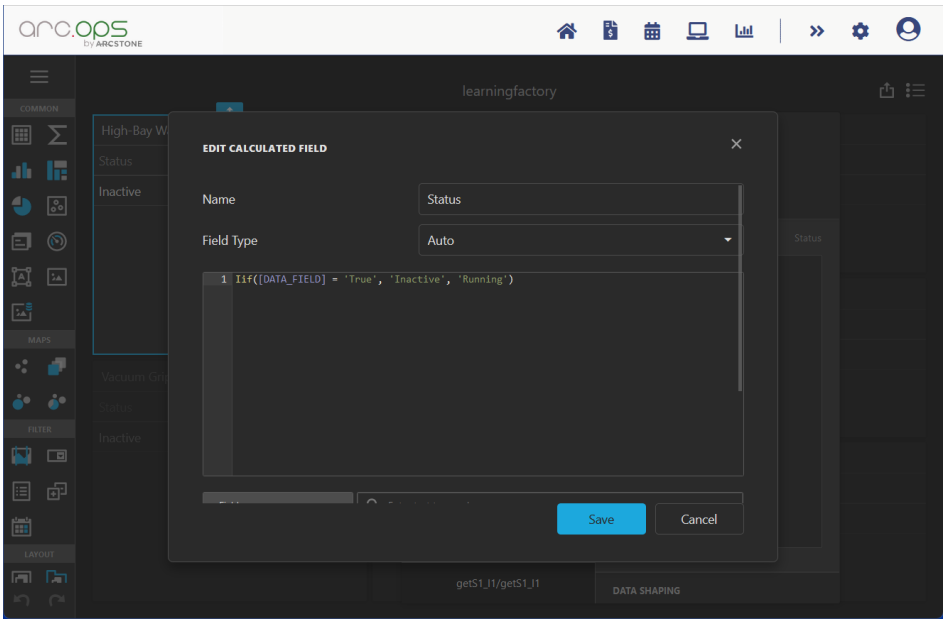


Figure 4.11: Calculated field

For each of the seven sensors a grid is added into the dashboard and to make the dashboard easier to understand an image of each station is added below the grid. The final dashboard is shown in Figure 4.12 below. In the top left corner is the high-bay warehouse, below that the vacuum gripper is shown. In the middle column the oven and milling machine are shown and on the right side are the chutes of the sorting line. In the screenshot the oven is running and only the white chute is full.

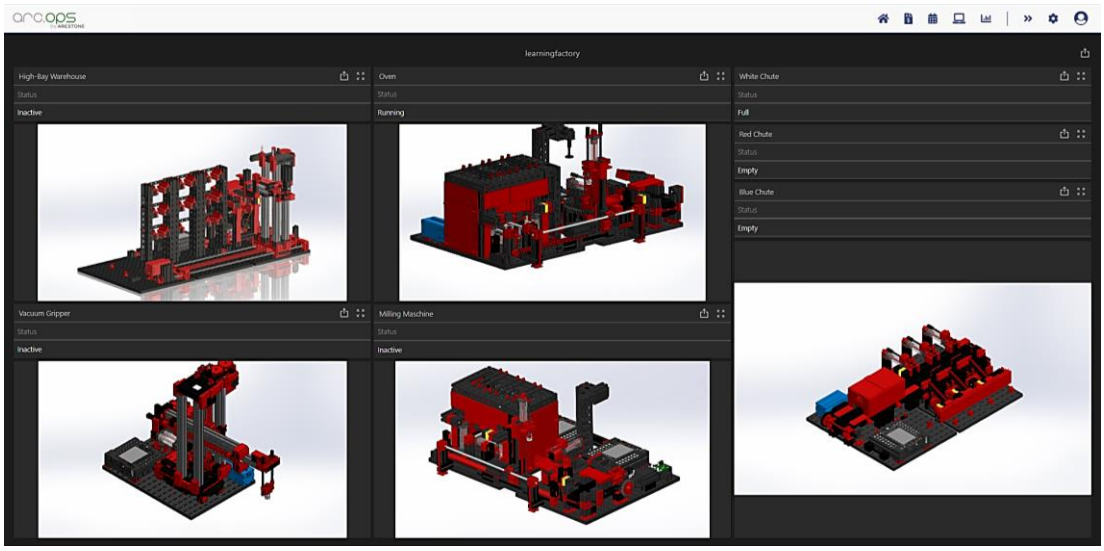


Figure 4.12: Dashboard

4.3 How the AI-Lernfabrik Works

This chapter will evaluate what one can learn from university Albstadt-Sigmaringen's AI-Campus course [15] of the learning factory. Notice that the Lernfabrik differs from the learning factory that is used in the laboratory, which makes it hard to apply on the learning factory. Therefore, this should only provide information on what may be possible with the model in the laboratory, rather than actually make it work. More about this can be found in chapter 5.

The focus will only be on the software tools and their application. The overall goal is to set up a system that is capable of monitoring and controlling the factory. The course is showing all the steps which are necessary for implementation. In the following paragraphs each step will be summarized and analysed. The full course and the task solutions can be reviewed in Appendix A and B.

To monitor the factory, the artificial intelligence system needs to fetch the available data and store them into a cloud. Most of the available data are sensor signals. The

data communication is done by MQTT. The first thing that is taught is a way to read the incoming data and write it into a database. For that purpose, the software tool Apache Nifi is used. The programming is done by a kind of block programming. The course is providing a sample code. As user the task is to control and change the source URI. The sample code enables the user to understand the modus operandi of the tool.

For the next step a python script is introduced. Its purpose is to request current states and to place orders, that get executed by the factory. The script will be accessed via Jupyter Hub. Basic understanding about python scripts and the python programming language are of advantage. The user should also be familiar with command language as the execution is performed via the Python Hub.

As a next step, sensor data is put into graphs. This step is not essential for the factory control, but interesting for factory analysation. Therefore, the data is transfered into the Amazon could. This enables the user to use the AWS Redshift tool. The course is accessing to the Redshift tool via another python script.

A second possibility of communication is introduced in the next step. The tool to establish the communication is Node-RED. The communication standard is UPCUA. Node-RED is a tool that is using block coding. The course is providing sample code. The tool is used to read form the factory, put the data into the Microsoft SQL database and control the learning factory. With that the course introduced the two most common used communication standards and useful tools to work with.

In the penultimate step the preparation for the artificial intelligence is done. Therefore, data editing is performed to have it neuronal network ready. The course is teaching how to use Node-RED and Python for that purpose. In the final step the neuronal network is fitted to the purpose of the factory. The course is also using a Python script to accomplish this. The next subchapter describes the application of the two used neuronal networks in detail.

4.3.1 Applying the Neuronal Networks

The university Albstadt-Sigmaringen is using two neuronal networks to detect the current state of the Lernfabrik during a typical ordering process. To make this possible it is necessary to preprocess all data in a way that a neuronal network is

applicable on it. How to preprocess the data in that manner and how to train and test the neuronal networks is described in the following.

4.3.1.1 Data Preprocessing

To begin with the data preprocessing, an ordering process is launched in order for the Lernfabrik to provide real data for the neuronal networks. First of all, the data is acquired by receiving messages from each topic, which are the different stations of the learning factory. Each message comes with several information, such as the time stamp, the station, an acknowledge code, an active or inactive state and an optional description and target. To get a set of data to work with, the ordering process is conducted once and the raw data is written into a JSON file and saved. Therefore, this data can be used in all of the following steps and it is not necessary to repeat an ordering process more often. A sample message from the high-bay warehouse is shown in the following figure 4.13.


```
{"topic": "f/i/state/hbw",
"message": {"ts": "2022-04-06T18:43:06.453Z", "station": "hbw", "code": 1, "description": "",
"active": false, "target": ""}}
```


Figure 4.13: Sample message [15]

The raw data is not sorted correctly corresponding to the time steps, so the first step for data preparation is to run a simple sorting algorithm. The reason for the data to be in the wrong order is that the distance between some recorded time stamps is less than a millisecond and during transmission they can get mixed up. Data is read from the JSON file and written into a list, then the time stamps is extracted from the messages and a list with two columns is created, where the time stamps are in the first column and the message in the second. Now the list is sorted by the time stamps, with the newest being on top, and the messages can be written in a new JSON file. An example is given in the figure 4.14 below.

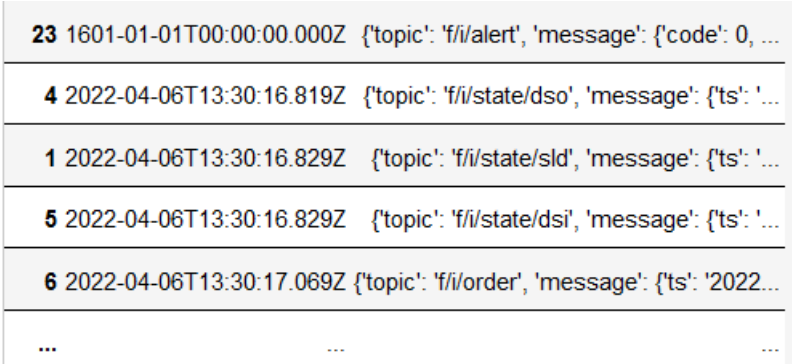


Figure 4.14: Sorted messages [15]

Now the raw data needs to be converted into a form that allows the models of the neuronal networks to work with it. The relevant data are the code, the active state and the target, which must be values from the same data type and range in order for the model to work efficient. Since the codes are either 0, 1 or 2 and the active state is either 0 or 1 it makes sense to also convert the target to small integer values. In this case 0 to 3 are used to represent the different targets. With that done there are only 13 inputs with small integer values left. The starting value of each input is set to minus 1 and updated whenever a new message is received, then the integer values are written into a new file. The initial state is reached as soon as there is a message from each station.

Table 4.2: Possible active state, code and target of each station

station	active	code	target
vacuum gripper robot	0, 1	1, 2	0, 1, 2, 3
sorting line with colour detection	0, 1	1, 2	
multi-processing station with oven	0, 1	1, 2	
high-bay warehouse	0, 1	1, 2	
input station	0, 1	0, 1	
output station	0, 1	0, 1	

The data is provided as in the following table 4.3. It can be observed that every value is -1 at the beginning and one station after the other gets updated to the initial state.

Table 4.3: Data

hbwstate	hbwcode	vgrstate	vgrcode	vgrtarget	mpostate	mpocode	sldstate	sldcode	dsistate	dsicode	dsistate	dsocode
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	1
-1	-1	-1	-1	-1	-1	-1	0	1	-1	-1	0	1
-1	-1	-1	-1	-1	-1	-1	0	1	0	1	0	1
-1	-1	-1	-1	-1	0	1	0	1	0	1	0	1
-1	-1	0	1	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	1	0	1	0	1	0	1
…	…	…	…	…	…	…	…	…	…	…	…	…

Next, every state that is possible during an ordering process has to be determined. This is achieved by sorting out all identical states, but only the consecutive ones, because one state can appear twice during the process. For example, the initial state (0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1) is found twice in that list because it occurs at the beginning as well as at the end of an ordering process. Next, all states that have a -1 in it are removed as they only occur before the stations are initialized. Then the remaining states have to be labelled manually, because otherwise the model would not know what they mean. This leads to the following data, where the state is added at the end of the values.

0,1,0,1,0,0,1,0,1,0,1,0,1,Resting
0,1,0,2,0,0,1,0,1,0,1,0,1,StorageOut
0,2,0,2,0,0,1,0,1,0,1,0,1,StorageOut
1,2,0,2,0,0,1,0,1,0,1,0,1,StorageOut
1,1,0,2,0,0,1,0,1,0,1,0,1,StorageOut
1,1,0,2,0,0,2,0,1,0,1,0,1,Processing
0,2,0,2,0,0,2,0,1,0,1,0,1,Processing
0,2,1,2,1,0,2,0,1,0,1,0,1,TransportToMPO
0,2,1,2,1,1,2,0,1,0,1,0,1,TransportToMPO
0,2,0,2,0,1,2,0,1,0,1,0,1,Processing
0,1,0,2,0,1,2,0,1,0,1,0,1,Processing
0,1,0,1,0,1,2,0,1,0,1,0,1,Processing

0,1,0,1,0,0,2,0,1,0,1,0,1,Processing
0,1,0,1,0,0,2,1,2,0,1,0,1,Sorting
0,1,0,1,0,0,1,1,2,0,1,0,1,Sorting
0,1,0,2,0,0,1,1,2,0,1,0,1,Sorting
0,1,0,2,0,0,1,1,1,0,1,0,1,Transport
0,1,0,2,0,0,1,0,1,0,1,0,1,Transport
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,0,1,0,0,OutputStation
0,1,1,2,3,0,1,0,1,0,1,1,0,OutputStation
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,1,0,0,1,TransportToDSO
0,1,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,1,2,2,0,1,0,1,1,0,0,1,TransportToHBW
0,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
0,1,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,0,2,0,0,1,0,1,0,1,0,1,StorageIn
1,2,0,1,0,0,1,0,1,0,1,0,1,StorageIn
0,2,0,1,0,0,1,0,1,0,1,0,1,StorageIn
1,2,0,1,0,0,1,0,1,0,1,0,1,StorageIn
0,2,0,1,0,0,1,0,1,0,1,0,1,StorageIn
0,1,0,1,0,0,1,0,1,0,1,0,1,Resting

If two or more states are represented by the same data, they have to be summarized under one name. This happened twice in the previous labelling, so it has to be changed to a meaningful common name. Now there are 10 states left, which are easy to process in the model. The states are StorageOut, TransportToMPO, Processing, Sorting, OutputStation, Transport, TransportToDSO, TransportToHBW, StorageIn and Resting

4.3.1.2 Create and Test the Models

When all the data preparation and preprocessing is done, they can finally be loaded into the model of the neuronal networks. At first the convolutional neuronal network (CNN) is used and then a recurrent neuronal network (RNN) is applied.
For the CNN the data has to be separated into input and output data. As input the 13 values of the stations are used. The 10 different output states, have to be encoded into an output array. First the names need to be encoded into integer values. The simple list then needs to be transferred into an array, where the number determines which position is a 1 and the other values are 0. For example, ‘Resting’ gets assigned with the number 9, which will be (0, 0, 0, 0, 0, 0, 0, 0, 0, 1). Figure 4.15 shows the final input and output array.

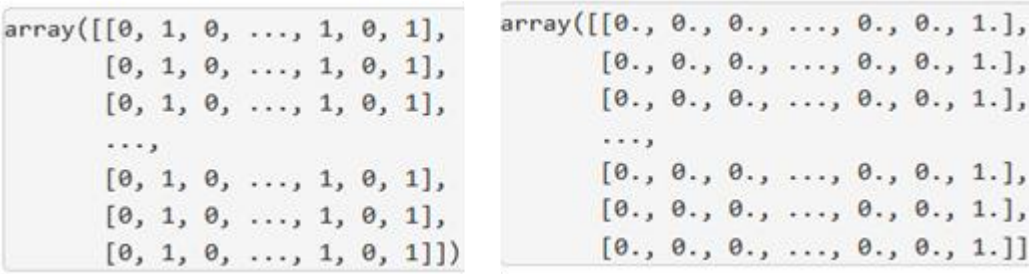


Figure 4.15: Input array (left) and output array (right) [15]

The following step is to set the definitions for the sequential model of the convolutional neuronal network. The model consists of 13 input neurons and 1 dias neuron that fires random values to ensure the model will not specify too much. A hidden layer with 10 neurons is added to the model as well as the 10 output neurons and the activation functions are defined. When the model is fully defined it has to be configured using a compile function, where the standard optimizer is chosen.
When training the model with the input and output data, the results shows that it gets better and better for each epoch. After 100 epochs the accuracy is >99% and the mean absolute error is <2%. After 200 epochs the accuracy reaches 100% and the mean absolute error is near 0% and from then on, the improvements are neglectable. When the model is now tested with sample input data is it expected that it always gives the correct result.

For the RNN the previous states are also put into consideration, leading to a faster and more precise result. The output data is the same as for the CNN, but the input

data has to be processed further as a 1D-array is not enough for the RNN. A multidimensional array with as many previous states as wanted is created. In this case only one previous state is used.

```
[[[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]
```

Figure 4.16: Multidimensional input array [15]

For model definition a different input layer called long short-term memory is required, which contains the 13 neurons, the dias neuron plus the previous states.

The model training follows the same procedure as for the CNN, but the results are worse this time, because after 500 epochs the accuracy is about 98% and the mean absolute error 2%. It seems like the RNN does not suit the application as well as the CNN does. When tested with the same sample data the results are also correct which indicates that the RNN is still good, but there are some issues with transitions between the output states.

CHAPTER 5

CONCUSSION AND RECOMMENDATIONS

5.1 MES

As for the end of this project the MES has been successfully connected to the learning factory and a dashboard that visualises the current state of the stations of the model has also been created. This is a significant improvement of the initial state of the learning factory, as data is now available to the MES and displayed on a dashboard, but it does not cover all the functionalities and capabilities of the MES. This chapter will discuss what can be done to enhance the usage of the MES in the future in order to get the most out of the learning factory.

When taking the guideline VDI 5600 into account, it can be seen that some of the points, like data acquisition and processing or information management are met. But still there is a lot more that a MES is capable of. At the moment only a dashboard that keeps track of the production process is used. Other features of Arcstone's MES arc.ops are not implemented yet. For example, orders can be tracked, jobs can be scheduled and viewed, workflows can be created or the personnel can be managed within the arc.ops software. As the dataflow from the automation level is established, these steps should be achievable in future projects with the learning factory.

The MES can also be seen as a data hub that connects all systems within an enterprise. By the end of this project the link to the MES is running by using arc.quire. The next step would be to connect the MES to an ERP, which can be done with arc.flow.

Moreover, there are improvements that can be done to the dashboard. Right now, it is designed to be easily understandable and to fulfil the purpose of showing the current

state in the most convenient way. Only the data of the sensors that are crucial to determine the state is used, but other sensor data may also be shown on a dashboard. With that it would be possible to show the production process in more detail or provide more information on the dashboard like the runtime of the stations.
The focus of this project was to show how to integrate a MES into a factory and how an enterprise can benefit from it. The dashboard is a good solution to demonstrate these points. Now it is up to others to take the usage of the MES to the next level.

5.2 AI Lernfabrik

This chapter concludes the project AI-Learning factory and gives recommendations. To analyse if the project was conducted successfully all requirements must be fulfilled. After the literature review the AI-Lernfabrik course was attended and the task were performed. Based on that the workshop a manual and task solutions were created. In appendix A the complete course can be found and the step-by-step solutions for the exercises are given in appendix B. Then the working principle was summarized to understand what one can learn from the course.
The remainder of this chapter will analyse what needs to be done in future to use the learning factory in full extent. It will be shown which hardware needs to be acquired, and what software is missing.

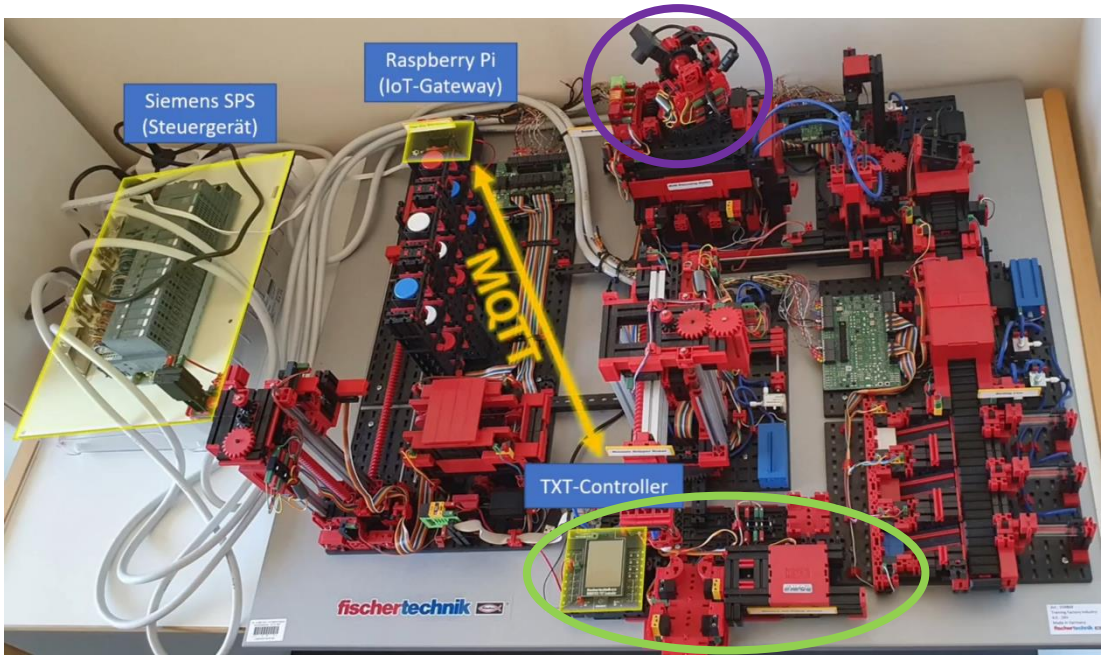


Figure 5.1: Controller overview [16]

The figure above shows the prototype of the factory. All the parts not highlighted can be found in the learning factory of the Innovation-Lab. The other elements are discussed in the following.

The purple highlighted parts are a combined sensor station. It is located on top of the oven, which is already part of the laboratory's factory. The biggest sensor of the station is a camera. It is capable of 360° filming. This is used for a web based remote control. The camera is not integrated in the software that controls the factory. It is meant for enabling persons far away from the factory to take a look. Environment sensors make up the other sensors of the station. The built in sensors are for measuring:

- Brightness
- Temperature
- Humidity
- Air quality

These values of the sensors get collected and analysed. They help to monitor the factory. With them, one can make sure that the production process is always within limits.

Highlighted by the green circle is the delivery station. The whole station is not present in the Innovationslab. It is used for incoming as well as outgoing goods. It consists of several elements. At the end of the production the gripper puts the workpiece into the delivery space. A light barrier is activated hereby. This information is collected by the controlling script. The information is then processed with all the other station data through the preparation steps to analyse it with the neuronal network. The missing information will be seen as an interruption to the production and a danger of a hang-up of the software.

In the following step the gripper puts the workpiece onto a colour sensor. It detects the colour of the product and this information is also stored. As soon as the workpiece is back into the storage, the software knows which colour it has. During ordering a customer can select the wished colour. The software is able to get a matching product.

In the final step of the delivery station a NFC reader and writer tags the workpiece with the production information. This is important that the workpiece "knows" what it is. Thus, this step is used as a simulation to real world factories, the NFC chip of

the workpiece is not read again. Meaning it is a nice to have feature for the factory simulation, but not necessary for the neuronal network.

Attached to the delivery station is the Fischertechnik txt controller. Its purpose is to collect all sensor information. After the data collection it is send to the IoT-Gateway via MQTT standard. There is no specification which chip has been used. There is also no information about the used software. This makes it hardly possible to rebuild this element.

The IoT-Gateway receives the data from the txt controller. The hardware used is a Raspberry PI, which has two purposes. The first purpose is to control the ordering process. The utilised software is the python script that was introduced in the course. The second purpose is to transmit the factory data to the cloud. This is handled with the OPCUA protocol. In the cloud the source code for the neuronal network is running. So, this connection is crucial to hand the production information to the neuronal network.

As important hardware is missing, the course of university Albstadt-Sigmaringen can not be applied to the learning factory in the Innovationslab. But the major takeaways are pointed out and the accumulated knowledge of the course can be used to create own neuronal networks that supervises the factory’s state.

REFERENCES

1. Ministry for Economy, Labour and Tourism (2023). *Technology and Knowledge Transfer*. Retrieved on June 30, 2023, from https://www.wirtschaft-digital-bw.de
2. Mercedes Benz Group (2023). *Industry 4.0 and the networked factory*. Retrieved on May 20, 2023, from https://group.mercedes-benz.com
3. Kletti, J.. *MES – Manufacturing Execution System*. 2nd Ed. Mosbach. Springer Vieweg. 2015
4. Verein Deutscher Ingenieure. *Manufacturing execution systems (MES)*.Düsseldorf. VDI 5600. 2016
5. International Society of Automation. *Enterprise-Control System Integration*. Research Triangle Park. ISA95. 2014
6. Arcstone (2022). *Unlocking the Value of Data*. Retrieved on April 30, 2023, from https://www.arcstone.co
7. Luger G. F. & Stubblefield W. A.. *Artificial intelligence*. 2nd Ed. Albuquerque. Benjamin/Cummings Pub. Co.. 1993
8. Hebb, D. O.. *The Organization of Behavior*. 1st Ed. New York. Psychology Press. 2002
9. Picton, P.. *Introduction to Neural Networks*. 1st Ed. London. Red Globe Press London. 1994
10. Ertel, W.. *Grundkurs Künstliche Inelligenz*. 5th Ed. Weingarten. Springer. 2021
11. Grum, M..*Construction of a Concept of Neuronal Modeling*. 1st Ed. Berlin. Springer. 2022
12. Sonnet, D.. *Neuronale Netzwerke kompakt*. 1st Ed. Hamburg. Springer Vieweg. 2022
13. Fischertechnik (2023). *Lernfabrik 4.0*. Retrieved on May 5, 2023, from https://www.fischertechnik.de
14. Beckhoff (2023). *TF6100 | TwinCAT 3 OPC UA*. Retrieved on May 20, 2023, from https://infosys.beckhoff.com

15. KI Campus (2022). *Lernfabrik 4.0 – Steuerung, Monitoring und NN-Modell*. Retrieved on June 25, 2023, from https://learn.ki-campus.org

16. Klein, M. (2022). *Osteraktion 12 Weiterbildung.* Retrieved on May 26, 2023, from https://www.hs-albsig.de

APPENDIX A

AI-CAMPUS COURSE: LERNFABRIK
LEARNING FACTORY 4.0 – CONTROL, MONITORING AND NN-MODEL

MORITZ HOHNEL

and

MATTIS TOM RITTER

Faculty of Mechanical and Manufacturing Engineering
Universiti Tun Hussein Onn Malaysia

JULY 2023

APPENDIX A

CONTENTS

CHAPTER 0		Introduction	60
CHAPTER 1		Module 1	61
	1.1	Communication structure of the learning factory	61
	1.2	University communication structure	62
	1.3	Exercise	63
CHAPTER 2		Module 2	66
	2.1	Flow Chart	66
	2.2	ConsumeMQTT	67
	2.3	UpdateAttribute / ConvertJSONToSQL	68
	2.4	PutSQL	69
	2.5	PutHDFS	69
	2.6	Exercises	70
CHAPTER 3		Module 3	72
	3.1	Ordering process	72
	3.2	storage process	73
	3.3	python script to control the factory	74
		3.3.1 Source code	75
	3.4	3.3 Request the states of the respective stations	76
		3.4.1 Source code	76
	3.5	State of high-bay warehouse	77
		3.5.1 Source code	77
	3.6	Ordering	79

APPENDIX A

3.6.1 Source code 79
3.7 Tyni-Client source code 80
3.8 Exercises 87
CHAPTER 4 Module 4 88
4.1 AWS Redshift Connector 88
4.2 SQL statement ldr 89
4.3 Brightness sensor value 89
4.4 Brightness in % 90
4.5 Brightness sensor source code 91
4.6 SQL-Statement bme680 92
4.7 4.7. Temperature 93
4.8 Humidity 94
4.9 Air quality 94
4.10 Air pressure 95
4.11 Environmental sensor source code 95
4.12 Exercises 96
CHAPTER 5 Module 5 97
5.1 Introduction 97
5.1.1 Flow editor 97
5.2 Reading from the learning factory 100
5.2.1 inject 101
5.2.2 OpcUa item 101
5.2.3 OpcUa-Client 103
5.2.4 join 104
5.2.5 function 105
5.2.6 MSSQL 108
5.2.7 debug 108

APPENDIX A

5.3 Controlling the learning factory 109
5.3.1 inject 109
5.3.2 function 110
5.3.3 OpcUa-Item 111
5.3.4 opcUa client 111
5.3.5 debug 112
5.4 Exercises 113
CHAPTER 6 Module 6 114
6.1 Nifi 114
6.2 6.2 Node-Red 116
6.3 OPC-Router 119
CHAPTER 7 Module 7 123
7.1 Neural Network Model as CNN and RNN with Tensorflow and Keras 123
7.2 System requirement 123
7.3 Data collection 124
7.4 Data preparation first part 125
7.5 Data analysis 127
7.6 Data preparation second part 128
7.7 Data preparation third part 131
7.8 Data Preparation fourth part 134
7.9 Data preparation the fifth part 136
7.10 CNN - Introduction 136
7.11 CNN - Load and provide data 137
7.12 CNN - Model Definition 139
7.13 CNN - Model Training 140
7.14 CNN - Model tests 141

APPENDIX A

7.15 RNN - Introduction 143

7.16 RNN - Data preparation 143

7.17 RNN - Model Definition 145

7.18 RNN - Model Training 146

7.19 RNN – Model tests 147

7.20 Model application in live mode 149

7.21 Exercise 151

APPENDIX A

CHAPTER 0 Introduction

Welcome to the course **Learning Factory 4.0 - Control, Monitoring and NN-Model**.

The fischertechnik learning factory is a learning factory that consists of the fischertechnik elements and is used to playfully simulate a digitized factory. The simulation maps the processes of ordering, production, delivery and storage in digitized process steps.

In the course, you will learn how to handle Internet of Things systems using the MQTT and OPC/UA protocols. You will learn how to handle data ingestion (ETL) routes with Nifi using exercise tasks to extract data from the production line, convert its form and write it to a database. Furthermore, you will perform data analysis using Python. At the end, we will take a look at a neural network for learning factory status detection.

APPENDIX A

CHAPTER 1 Module 1

Welcome to module 1.

In module 1 we describe the communication using MQTT and OPC/UA between the individual components of the learning factory and the university cloud. Your task is to connect to the individual systems via the university cloud.

Learning objectives

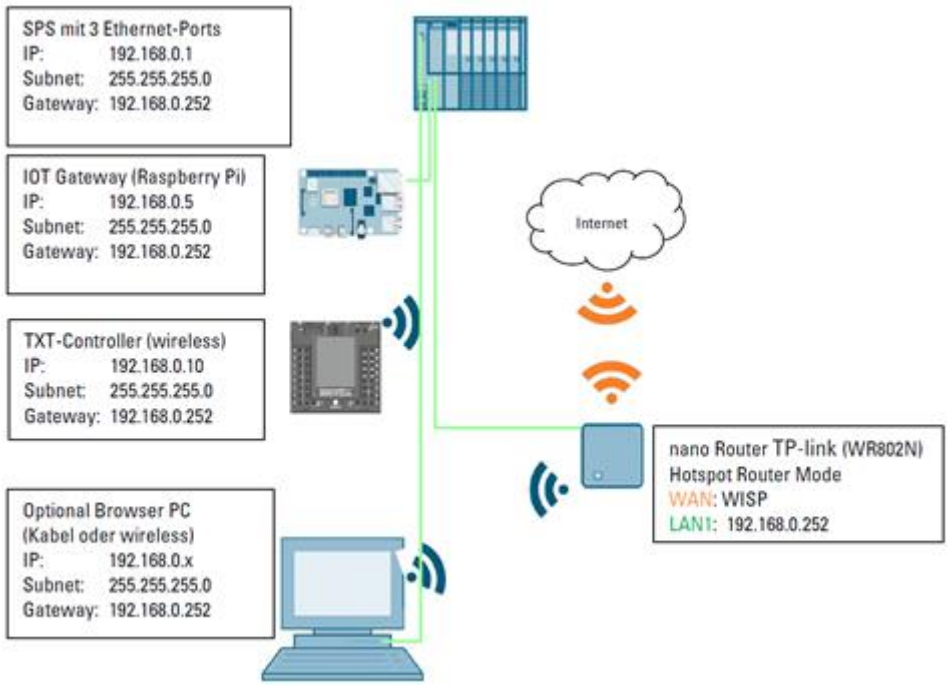
- Getting to know the structure of the learning factory.
- Create a communication to the individual systems of the learning factory.

MQTT is a system that can be used to transmit data to a server, which is called a broker here. A publisher writes the data and a subscriber reads the data for further processing.

OPC/UA is a protocol for monitoring PLC controllers.

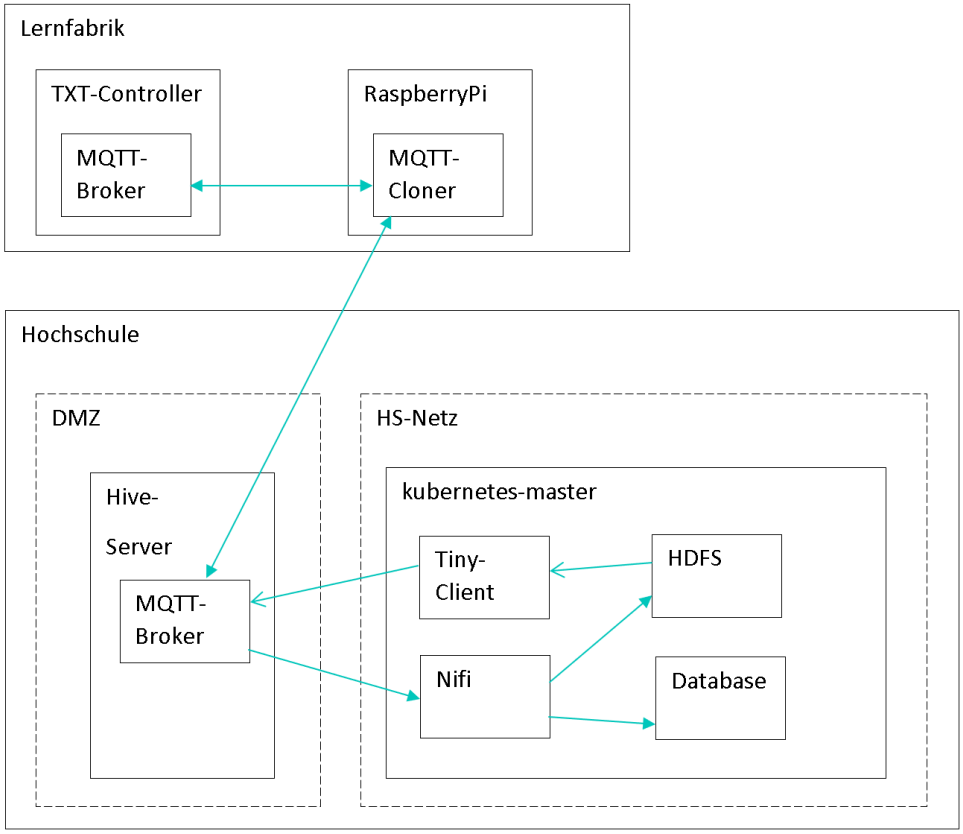
1.1 Communication structure of the learning factory

The figure describes the communication between the individual components of the learning factory. The components used are a PLC, a Raspberry PI and a controller. The Raspberry PI serves as a gateway to the university cloud. The PI runs a Python script to execute orders and send monitoring data to the university cloud. An MQTT broker running on the controller is used for data monitoring. The MQTT broker runs on the controller.



1.2 University communication structure

The figure shows the individual components in the HochschulCloud. A Hive and an MQTT broker are available in the demilitarized zone (DMZ) within the university cloud. The MQTT messages are sent from the learning factory to the MQTT server in the DMZ, where they are further processed using Apache Nifi as an ETL system and written to a relational database or Apache Hadoop. The MQTT cloner copies the MQTT communication that takes place on the controller to the university's MQTT broker in the DMZ. The cloner is a Python script that connects to both brokers and copies the messages



Apache Hadoop is a distributed file system.

Using Hive, files can be accessed and analyses can be created using SQL. SQL is a standard query language for database systems.

Apache Nifi is a data ingestion system, often called an ETL system.

ETL stands for Extract, Transfer, Load. In the Extract phase, data is loaded from a system, Transfer is used to modify the data, and Load is used to store the data back into a target system. In this system the data is read from the learning factory, transformed into the format of the target database and written into the database.

1.3 Exercise

Connect to the following systems to learn the system structure described in modules 1.1. and 1.2. You will perform exercises with these systems in Modules 2 through 5.

- Nifi
- Redshift via Python
- University Cloud
- Node-RED (OPC communication)

APPENDIX A

Note: If you have connection problems, please use a different browser.

System access

- Nifi

  You can access Nifi via the address hivemqserver.feste-ip.net:9443/nifi/.

  The access data are (user: lernfabrik / password: L5nf1br|k)

- Redshift via Python

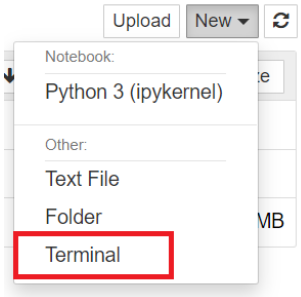
  You can run the Python scripts via university Jupyter notebooks in our Jupyter hub. To do this, access the address hivemqserver.feste-ip.net:9001.

  Redshift is a relational database in the AWS Cloud. In a relational database, data is stored in the form of tables.

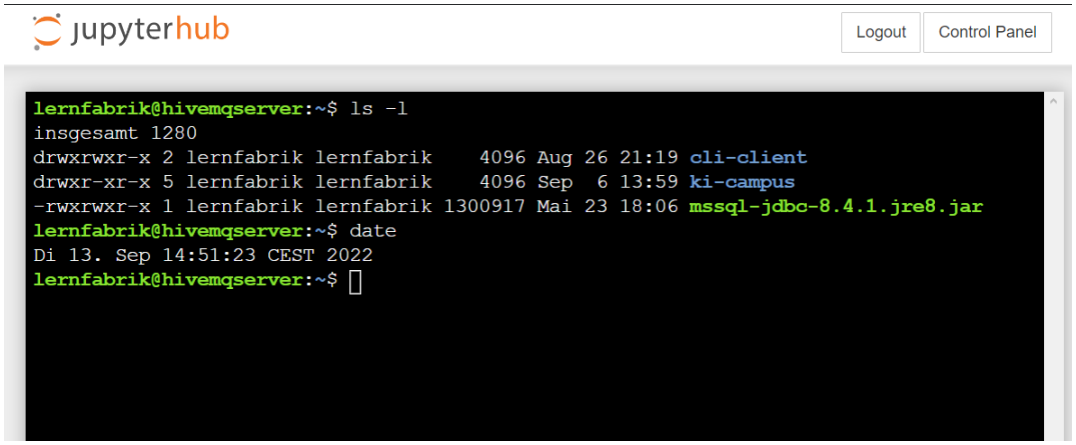
  The access data are (user: lernfabrik / password: L5nf1br|k)

- University cloud

  In order to use the access to the university server, log in to the JupyterHub server of the HS and start a terminal window. To do this, click on New in the upper right corner after logging in. And then click on Terminal.



A fully functional terminal window will open with which you can operate the CLI client, for example.



APPENDIX A

The scripts for the exercises can be found in the directory /home/lernfabrik/ki-campus. You do not have to execute the scripts yet.

- Node-Red (OPC communication)

  Node-RED is used to realize the OPC communication to the learning factory. Node-Red can be reached at the following address hivemqserver.feste-ip.net:1880.

  The access data are (user: lernfabrik / password: L5nf1br|k)

APPENDIX A

CHAPTER 2 Module 2

Welcome to module 2.

In this module we will learn about the Apache Nifi ETL system to read data from an MQTT broker, convert its form to SQL and write it to a database.

Learning Objectives

- To get to know Apache Nifi in order to read data from an MQTT broker, convert its form to SQL and write to a database.

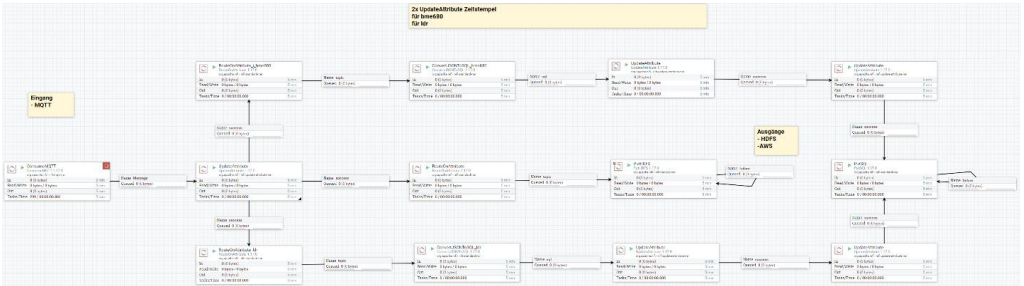
2.1 Flow Chart

Nifi is used to read messages from the learning factory and write them to a relational database.

Apache Nifi is a data ingestion system, often called an ETL system. ETL stands for Extract, Transfer, Load. In the Extract phase, data is loaded from a system, Transfer is used to modify the data, and Load is used to store the data back into a target system. In this system, the data is read from the learning factory, transformed into the format of the target database, and written to the database. The Nifi system is available in the university cloud at the URL hivemqserver.feste-ip.net:9443/nifi/. To the right of the Nifi logo are processors that can be used for a variety of data transformations.

Nifi is used to read the cloned messages and split them accordingly. Most of the messages are stored on HDFS. The messages that contain sensor values are written to a database. HDFS is a distributed file system (Hadoop Distributed Filesystem). XML- JSON or CSV files can be stored in HDFS.

The entire flow in Nifi is shown in the figure below:



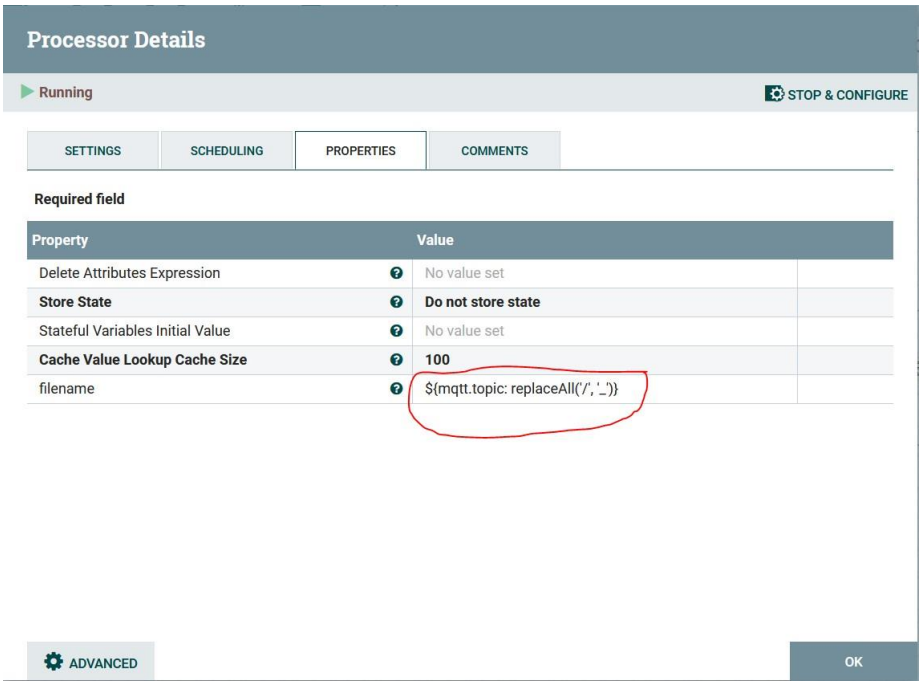
2.2 ConsumeMQTT

The ConsumeMQTT processor is used to read data from an MQTT server. Message Queue Telemetry Transport (MQTT) is an open message protocol for machine-to-machine (M2M) communication. It enables the transmission of telemetry data in the form of messages between devices, despite high delays or limited networks. Corresponding devices range from sensors and actuators, cell phones, embedded systems in vehicles or laptops to fully developed computers. The protocol was developed by Andy Stanford-Clark of IBM and Arlen Nipper of Cirrus Link Solutions. In Nifi, the processor ConsumeMQTT is used as an MQTT subscriber that connects to an MQTT broker (server) under a specific topic.

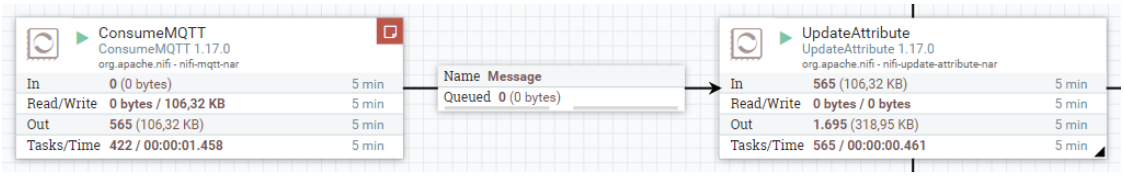
In Nifi there is the processor ConsumeMQTT where the following parameters have to be set:

Property	**Value**
Broker URI	tcp://141.87.109.227:1883
Client ID	nifi_lernfabrik_tls_client
Topic Filter	#

The data is changed using the UpdateAttribute process to swap the "/" in Topics for "_". The background is that otherwise the data is stored in HDFS directories. For this the character / is a hindrance. Thus e.g. from "i/ldr" -> "i_ldr" becomes. For this the following is entered in the property "filename":



View Chart.



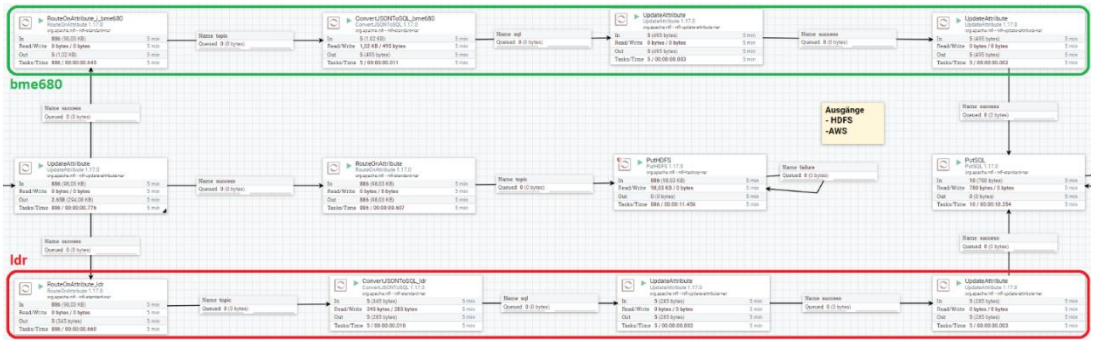
2.3 UpdateAttribute / ConvertJSONToSQL

After that the branch splits. The division is based on the data values. A distinction is made between environmental sensor values (type bme680), brightness sensor values (type ldr) and production data. Production data is written to HDFS, while the other values are written to a relational database, here AWS Redshift. AWS Redshift is a relational database in the Amazon Cloud (AWS).

Additionally, the relational data ldr and bme680 are split again to better distinguish the topics in the RouteOnAttribute process.

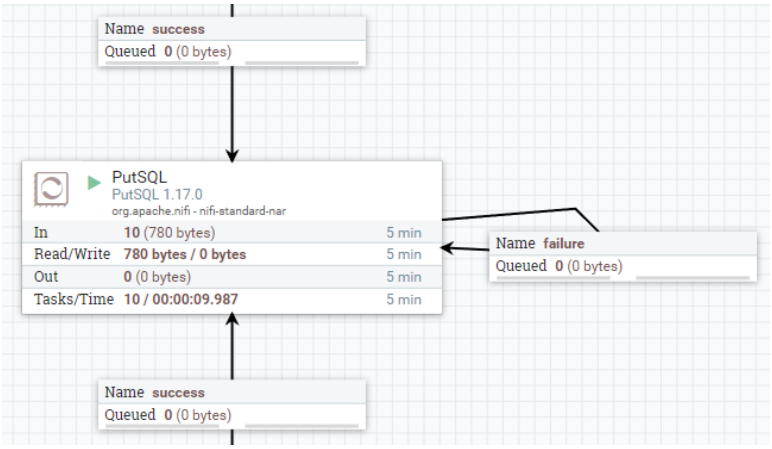
Therefore, the data is split by "i/bme680" and "i/ldr". The two branches are almost identical with the exception of the data processed in them. After converting the data to SQL using the ConvertJSONtoSQL process, two UpdateAttribute processes are still applied to it to adjust the timestamps so that AWS can store them. In the timestamp, the letters "Z" and "T" are removed using:

```
${sql.args.9.value: replace('Z', '')}
${sql.args.9.value: replace('T', '')}
```



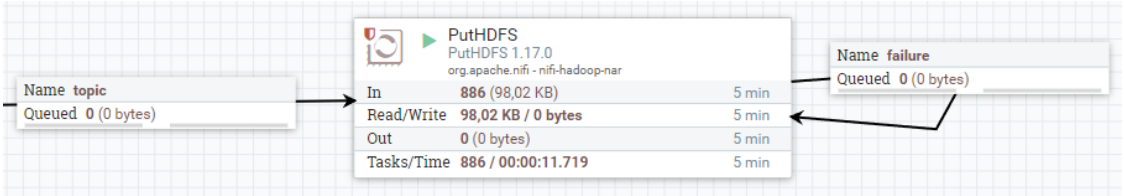
2.4 PutSQL

Finally, the data is written to the database using the PutSQL process.



2.5 PutHDFS

All other topics are written to HDFS using PutHDFS.



2.6 Exercises

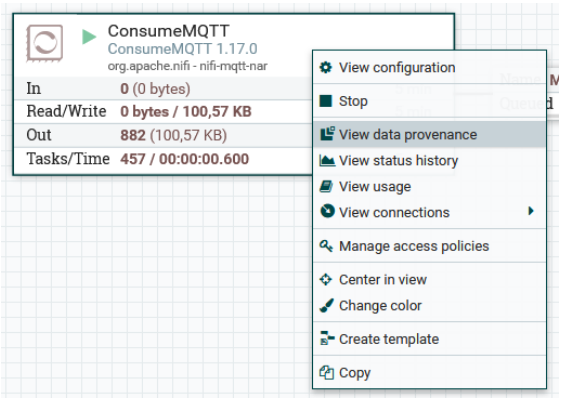
Connect to Apache Nifi at the following address https://hivemqserver.feste-ip.net:9443/nifi/. (user: lernfabrik / password: L5nf1br|k) and solve the following tasks:

a) Determine the address of the MQTT server in the university cloud (section 2.2).

b) In the MQTT processor, see how data is fetched cyclically from the facility in JSON format. Track the data by right-clicking on the "PutHDFS" or "PutSQL" processors and using "View data provenance" to view the contents.

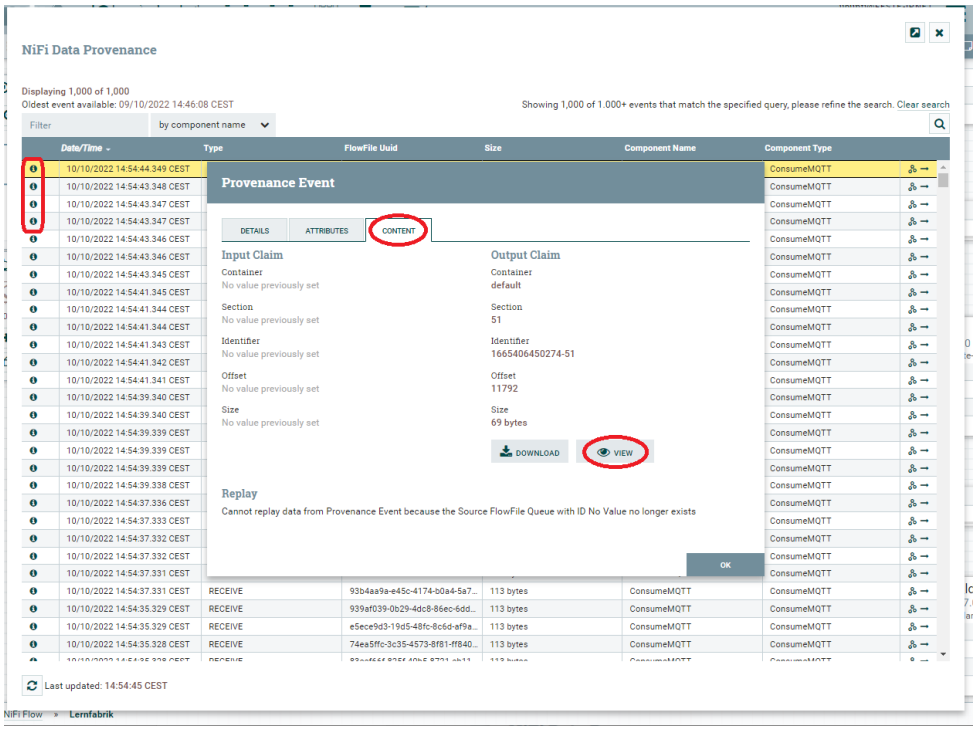
Use the following example as a guide:

Right-click on the ConsumeMQTT process. And then click on View data provenance.

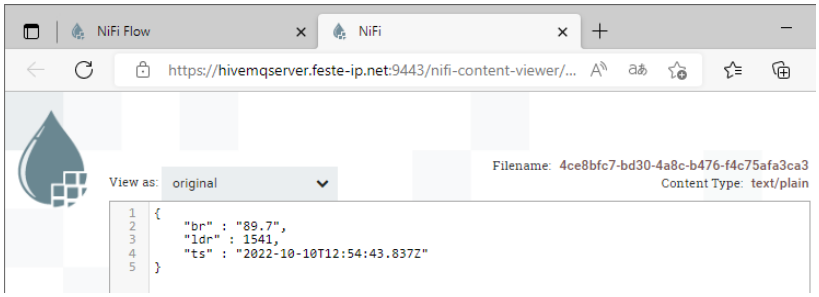


A list with all received MQTT messages will open. Click on one of the info icons on the far left of the table, Again a screen with three tabs opens, go to Content. And finally on View

APPENDIX A



The full MQTT message will we displayed.



APPENDIX A

CHAPTER 3 Module 3

Welcome to module 3.

In Module 3, we will learn a Python script that we can use to control the system such as placing orders, requesting the status of stations, and displaying camera images.

Learning Objectives

- Control a production plant using Python with MQTT.

Materials

- Python script to control the production line

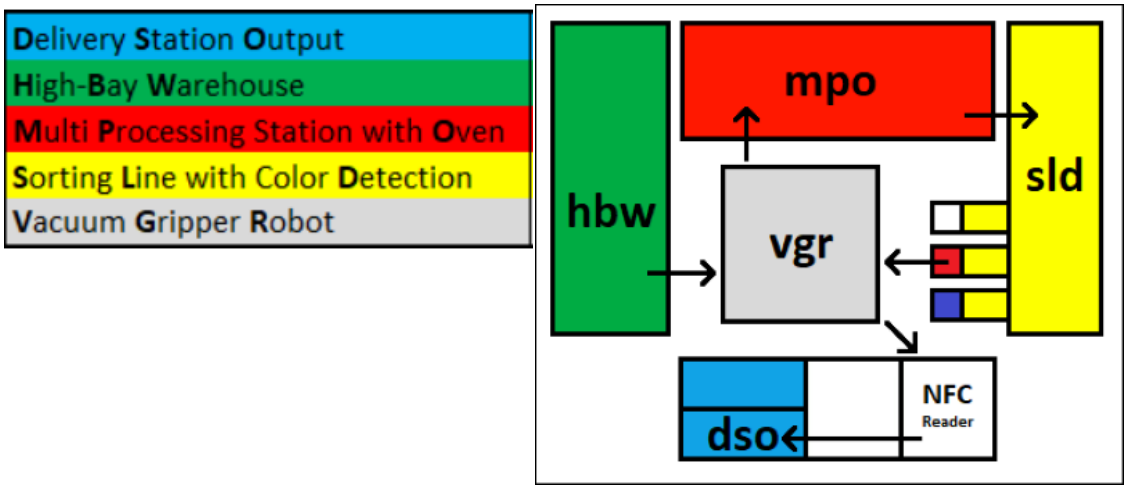
3.1 Ordering process

When an ordering process is triggered, the transport arm of the high-bay warehouse (HBW) moves to the storage system, picks up the ordered workpiece and places it in the input/output station. There it is conveyed to the pick-up position of the vacuum suction pad.

The vacuum gripper robot (VGR) picks up the workpiece from the input/output station of the high-bay warehouse (HBW) and places it on the slide of the furnace (MPO). There, the workpiece is pushed in, fired and moved out again. The transport carriage with vacuum suction pad then transports the workpiece to the "Mill" processing machine. There, the workpiece is placed on the rotary table. After the milling process, the workpiece is pneumatically pushed onto the conveyor belt (SLD).

On the conveyor belt, the workpiece passes through a colour recognition system. Depending on the colour detected, the workpiece is pneumatically pushed onto the corresponding material chute, where it is located in a pick-up area of the vacuum suction pad. From here, the workpiece is picked up by the vacuum gripper robot (VGR) and held for final labelling via the NFC reader, where the part is written with workpiece-relevant data such as order date, manufacturing or delivery data.

Finally, the workpiece is deposited in the output tray (DSO) of the input/output station.



3.2 storage process

Both the output tray (DSO) and the input tray (DSI) are mounted at an angle. After completion of the ordering process, the workpiece is placed in the output tray and slides directly into the input tray. Two light barriers are interrupted in the process. The first light barrier (DSO) signals the completion of the order. When the second light barrier (DSI) is interrupted, the infeed process is started automatically.

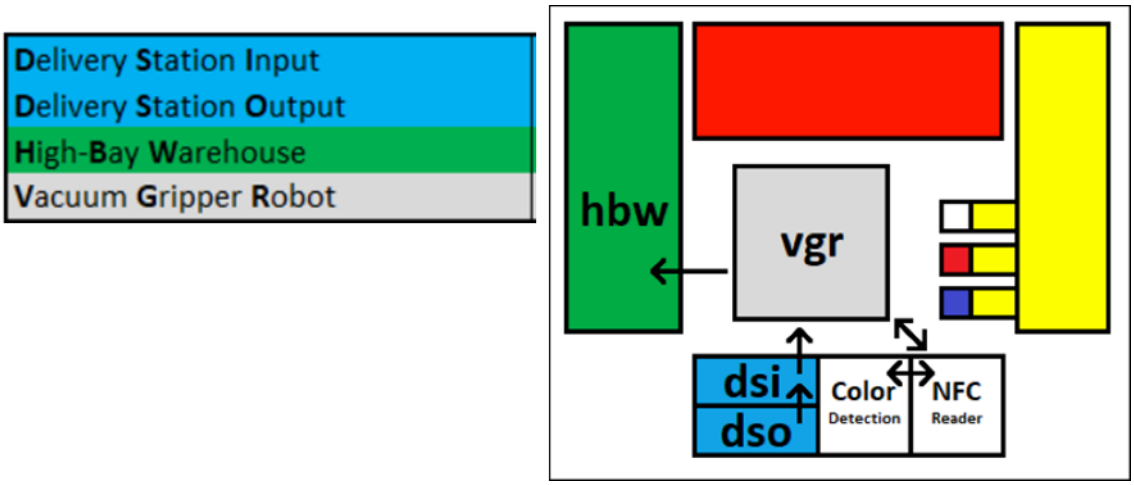
Here, the workpiece is picked up by the vacuum gripper robot and moved to the NFC reader, which deletes old data on the workpiece's NFC tag.

The workpiece is then moved to the colour sensor for colour detection.

Before storage, the determined colour, as well as other information such as delivery data, is written to the workpiece's NFC tag with the NFC reader.

Next, the transport arm of the high-bay warehouse picks up an empty container from the high-bay warehouse and transports it to the pickup position of the vacuum gripper robot.

Finally, the vacuum gripper robot places the workpiece in the waiting container, which is transported back to the high-bay warehouse by the transport arm of the high-bay warehouse and stored.



3.3 python script to control the factory

You can execute the script as described in module 1 with the command python3 lernfabrik.py -h. The individual options of the script are described in the following sections.

python3 lernfabrik.py -h

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -h
usage: lernfabrik.py [-h] [-o order] [-s state] [-t stock] [-c cam]

Benutzung python3 lernfabrik.py -XX z.B. lernfabrik.py -o RED z.B.
lernfabrik.py -s hbw z.B. lernfabrik.py -t 1

optional arguments:
  -h, --help            show this help message and exit
  -o {RED,BLUE,WHITE}, --order {RED,BLUE,WHITE}
                        Bestellung aufgeben.
  -s {hbw,vgr,mpo,sld,dsi,dso}, --state {hbw,vgr,mpo,sld,dsi,dso}
                        Stations Status abfragen.
  -t {0,1}, --stock {0,1}
                        Status des Hochregals abfragen
  -f, --follow          Sequentielles Abfragen aller Stationen.
  -c {0,1}, --cam {0,1}
                        Aktuelles Kamerabild anzeigen.
  -m movecam movecam, --movecam movecam movecam
                        Kamerasteuerung. Richtung: ['up', 'down',
'right',
                        'left']; Winkel: [2, 5, 10, 20]
```

3.3.1 Source code

The function parse_args() is a helper function to evaluate the passed arguments.

```
# Argument verwaltung
def parse_args():
    parser = argparse.ArgumentParser(
        description='Benutzung python3 lernfabrik.py -XX \n' \
                    'z.B. lernfabrik.py -o RED \n' \
                    'z.B. lernfabrik.py -s hbw \n' \
                    'z.B. lernfabrik.py -t 1'
    )
    parser.add_argument(
        '-o', '--order',
        choices=colors,
        help='Bestellung aufgeben.',
    )
    parser.add_argument(
        '-s', '--state',
        choices=stations,
        help='Stations Status abfragen.',
    )
    parser.add_argument(
        '-t', '--stock',
        choices=modi,
        type=int,
        help='Status des Hochregals abfragen.',
    )
    parser.add_argument(
        '-f', '--follow',
        action='store_true',
        help='Sequentielles Abfragen aller Stati.',
    )
    parser.add_argument(
        '-c', '--cam',
        choices=modi,
        type=int,
        help='Aktuelles Kamerabild abfragen.',
    )
    parser.add_argument(
        '-m', '--movecam',
        metavar='movecam',
        nargs=2,
        help='Kamerasteuerung. Richtung: ' + str(directions) + ';
Winkel: ' + str(angle),
    )
# end function parse_args
```

3.4 3.3 Request the states of the respective stations

The command python3 lernfabrik.py -s vgr displays the current state of the vacuum gripper (Vacuum Gripper Robot). There are the two states active (true) and inactive (false).
The command python3 lernfabrik.py -s hbw displays the current state of the High-Bay Warehouse. There are the two states active (true) and inactive (false).
The python3 lernfabrik.py -s mpo command displays the current state of the Multi Processing Station. There are the two states active (true) and inactive (false).
The command python3 lernfabrik.py -s sld displays the current state of the Sorting Line. There are the two states active (true) and inactive (false).
A detailed explanation of the codes can be found in module 6.2.

python3 lernfabrik.py -s vgr

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -s vgr

Status Station:         vgr
Letzter Zeitstempel:    2021-09-28T08:00:56.911Z
Letzter Status:         False
Code:                   1
```

3.4.1 Source code

The function createState() shows the current state of a specific station

```
# Returns state of a station
# {"ts":"2021-09-
22T09:32:45.436Z","station":"sld","code":1,"description":"","active
":false,"target":""}
def createState(station):
   value = getJSONFromHDFSByKey('f_i_state_' + station)
   state_dict = json.loads(value)

   print(' ')
   print('Status Station: \t' + str(state_dict['station']))
   print('Letzter Zeitstempel: \t' + str(state_dict['ts']))
   print('Letzter Status: \t' + str(state_dict['active']))
   print('Code: \t\t\t' + str(state_dict['code']))
   print('Description: \t\t\t' + str(state_dict['description']))
   print(' ')
# end function createState
```

3.5 State of high-bay warehouse

The command python3 lernfabrik.py -t 1 displays the fill levels of the high bay warehouse. The high bay warehouse is filled with 9 bins of the colours RED, BLUE or WHITE.

python3 lernfabrik.py -t 1

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -t 1

Status Hochregal
Letzter Zeitstempel:    2021-09-28T08:00:58.940Z

                |       Num 1           |       Num 2           |       Num 3
-------------------------------------------------------------------------------
        | id:   | 045e55a2186580        | 046c57a2186580        | 045159a2186580
 ROW A  | state:| RAW                   | RAW                   | RAW
        | type: | RED                   | WHITE                 | BLUE
-------------------------------------------------------------------------------
        | id:   | 046c57a2186580        | 045159a2186580        | 045b56a2186580
 ROW B  | state:| RAW                   | RAW                   | RAW
        | type: | WHITE                 | BLUE                  | RED
-------------------------------------------------------------------------------
        | id:   | 045159a2186580        | 045b56a2186580        | 044057a2186580
 ROW C  | state:| RAW                   | RAW                   | RAW
        | type: | BLUE                  | RED                   | WHITE
-------------------------------------------------------------------------------
```

3.5.1 Source code

The function createStock() shows the current state of the high-bay warehouse.

```
# Gibt den aktuellen Status des Hochregals zurueck
def createStock(stock):
   value = getJSONFromHDFSByKey( stock_topic )
   stock_dict = json.loads(value)

   print(' ')
   print(' Status Hochregal')
   print(' Letzter Zeitstempel: \t' + str(stock_dict['ts']))
   print(' ')

   if(stock == 1): # manuelle Ausgabe
      print('\t\t| \tNum 1 \t\t| \tNum 2 \t\t| \tNum 3')
      print('-------------------------------------------------------
---------------------------------')

      for i in [0, 3, 6]:
         # find empty storage
         for j in [0, 1, 2]:
```

APPENDIX A

```
            try:
               temp =
stock_dict['stockItems'][j+i]['workpiece']['id']
            except:
               stock_dict['stockItems'][j+i]['workpiece']['id'] =
"0"
               stock_dict['stockItems'][j+i]['workpiece']['state']
= " "
               stock_dict['stockItems'][j+i]['workpiece']['type'] =
" "


         print(' \t| id: \t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['id']) + \
                                  ('\t\t\t| ' if
str(stock_dict['stockItems'][i+0]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+1]['workpiece']['id']) + \
                                     ('\t\t\t| ' if
str(stock_dict['stockItems'][i+1]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+2]['workpiece']['id']) )
         print(' ROW ' + ('A' if i == 0 else ('B' if i == 1 else
'C')) +' \t| state:| ' + \

str(stock_dict['stockItems'][i+0]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['state']) )
         print(' \t| type:\t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['type']))
         print('-----------------------------------------------------
-----------------------------------')
   else: # Standard ausgabe JSON
      print(json.dumps(stock_dict, indent=3, sort_keys=True))
   print(' ')
# end function createStock
```

3.6 Ordering

The command python3 lernfabrik.py -o can be used to order a container with the colours WHITE, RED or BLUE. The container is provided in the output tray and then automatically stored again so that it can be ordered again.

python3 lernfabrik.py -o RED

```
hadoop@kubernetes-master:~/lernfabrik$ python3 lernfabrik.py -o RED

Es wird ein Werkstück mit der Farbe "RED" bestellt.
Connected to broker via TCP
Bestellvorgang war erfolgreich!
```

3.6.1 Source code

The function createOrder() is used to trigger an order process.

```
# Gibt einen Bestellvorgang auf
# {"ts":"YYYY-MM-DDThh:mm:ss.fffZ", "type":"BLUE"}
def createOrder(color):
   print(' ')
   print('Es wird ein Werkstück mit der Farbe \"' + color +'\"
bestellt.')
   print(' ')

   # get flag
   if(getSimulateFlag(1,False)):
      print(' Bestellung kann nicht ausgefuehrt werden da im Moment
ein Bauteil bestellt wird.')
      print(' ')
      return 1

   # MQTT Verbindung aufbauen
   client = paho.Client(client_id)
   client.on_publish = on_publish
   client.on_connect = on_connect
   client.connect(broker, port=port)
   client.loop_start()

   while connected != True:
      time.sleep(0.1)

   # aktueller Zeitstempel
   the_time = datetime.datetime.now()
   time_stamp = the_time.strftime(format = "%Y-%m-%dT%H:%M:%S.%f")
```

```
   order = '{\"ts\":\"' + time_stamp[:-3] + 'Z\", \"type\":\"' +
color + '\"}'
   client.publish(order_topic, order, qos=1)
   time.sleep(1)
   print(' ')
# end function createOrder
```

After placing an order, you can use the "-f" argument to track the status of each station.

python3 learningfactory.py -f

3.7 Tyni-Client source code

lernfabrik.py

```
import argparse
import sys
import os
import json
import paho.mqtt.client as paho
import datetime
import time
import http.server
import socketserver

hdfsMainDir = '/user/lernfabrik'
colors = ['RED', 'BLUE', 'WHITE']
stations = ['hbw', 'vgr', 'mpo', 'sld', 'dsi']
modi = [0, 1]
image_json = ''

# MQTT-Broker
connected = False
broker = '141.87.109.227'
port = 1883
http_port = 8000
client_id = 'hs_client_order_lernfabrik'
order_topic = 'f/o/order'<
```

The MyHttpRequestHandler() class has the do_GET() function. This writes the HTML page to the HTTPRequestHandler. This in turn is used by the web server to send the web page on request.

```
class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
   def do_GET(self):
      # Sending an '200 OK' response
      self.send_response(200)
```

```
      # Setting the header
      self.send_header("Content-type", "text/html")
      # Whenever using 'send_header', you also have to call
'end_headers'
      self.end_headers()

      global image_json

      # Some custom HTML code, possibly generated by another
function
      html = '<html><head></head><body><h1>current camera
image</h1><div>time stamp: ' + str(image_json['ts']) + \
             '</div><div><img src=\"' + str(image_json['data']) +
'\" alt=\"camera image\"
style=\"width:640px;height:480px;\"/></div></body></html>'

      # Writing the HTML contents with UTF-8
      self.wfile.write(bytes(html, "utf8"))

      return
# end Class MyHttpRequestHandler
```

The function startWebServer() starts the web server.

```
# Startet HTTP Server
def startWebServer():
   try:
      #handler = http.server.SimpleHTTPRequestHandler
      handler = MyHttpRequestHandler
      with socketserver.TCPServer(("", http_port), handler) as
httpd:
         #print("Server started at localhost:" + str(http_port))
         httpd.serve_forever()
   except:
      print("Fehler Bei HTTP Server: ", sys.exc_info()[0])
      raise
# end function startWebServer
```

The function getJSONFromHDFSByKey() is a helper function, which reads the data to a specific topic from HDFS.

```
# Holt einen Wert in HDFS by Key
def getJSONFromHDFSByKey(key):
   try:
      loc = hdfsMainDir + '/' + key
      value = os.popen("hdfs dfs -cat " + loc).readlines()

      value2 = ''
      for val in value:
         value2 += val
```

```
      return value2
   except:
      print("Fehler Bei HDFS Abfrage: ", sys.exc_info()[0])
      raise
# end function getJSONFromHDFSByKey
```

The function craeteState() shows the current state of a specific station.

```
# Gibt Status zur einer Station zurueck
# {"ts":"2021-09-
22T09:32:45.436Z","station":"sld","code":1,"description":"","active
":false,"target":""}
def createState(station):
   value = getJSONFromHDFSByKey('f_i_state_' + station)
   state_dict = json.loads(value)

   print(' ')
   print('Status Station: \t' + str(state_dict['station']))
   print('Letzter Zeitstempel: \t' + str(state_dict['ts']))
   print('Letzter Status: \t' + str(state_dict['active']))
   print('Code: \t\t\t' + str(state_dict['code']))
   print('Description: \t\t\t' + str(state_dict['description']))
   print(' ')
# end function createState
```

The function createSTock() shows the current filling level of the high-bay warehouse.

```
# Gibt den aktuellen Status des Hochregals zurueck
def createStock(stock):
   value = getJSONFromHDFSByKey( stock_topic )
   stock_dict = json.loads(value)

   print(' ')
   print(' Status Hochregal')
   print(' Letzter Zeitstempel: \t' + str(stock_dict['ts']))
   print(' ')

   if(stock == 1): # manuelle Ausgabe
      print('\t\t| \tNum 1 \t\t| \tNum 2 \t\t| \tNum 3')
      print('-----------------------------------------------------
---------------------------------')

      for i in [0, 3, 6]:
         # Leeres Regalfach abfangen
         for j in [0, 1, 2]:
            try:
               temp =
stock_dict['stockItems'][j+i]['workpiece']['id']
            except:
               stock_dict['stockItems'][j+i]['workpiece']['id'] =
"0"
               stock_dict['stockItems'][j+i]['workpiece']['state']
```

```
= " "
                stock_dict['stockItems'][j+i]['workpiece']['type'] =
" "


         print(' \t| id: \t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['id']) + \
                                    ('\t\t\t| ' if
str(stock_dict['stockItems'][i+0]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+1]['workpiece']['id']) + \
                                      ('\t\t\t| ' if
str(stock_dict['stockItems'][i+1]['workpiece']['id']) == "0" else
'\t| ') + \

str(stock_dict['stockItems'][i+2]['workpiece']['id']) )
         print(' ROW ' + ('A' if i == 0 else ('B' if i == 1 else
'C')) +' \t| state:| ' + \

str(stock_dict['stockItems'][i+0]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['state']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['state']) )
         print(' \t| type:\t| ' +
str(stock_dict['stockItems'][i+0]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+1]['workpiece']['type']) + '\t\t\t|
' + \

str(stock_dict['stockItems'][i+2]['workpiece']['type']))
         print('-----------------------------------------------------
-----------------------------------')
   else: # Standard ausgabe JSON
      print(json.dumps(stock_dict, indent=3, sort_keys=True))
   print(' ')
# end function createStock
```

The function on_connect() is a callback, which gets called as soon as the MQTT-Broker is connecting.

```
# Callback Funktion beim connecten
def on_connect(client, userdata, flags, rc):
   if rc == 0:
      print("Connected to broker via TCP")
      global connected            # Use global variable
      connected = True            # Signal connection
   else:
```

```
      print("Connection failed via TCP")
# end function on_connect
```

The function on_publish() gets called, when a message is published.

```
# Callbak Funktion beim publishen
def on_publish(client, userdata, result):
   print('Bestellvorgang war erfolgreich!')
# end fcuntion on_publish
```

The function createOrder() triggers a ordering process.

```
# Gibt einen Bestellvorgang auf
# {"ts":"YYYY-MM-DDThh:mm:ss.fffZ", "type":"BLUE"}
def createOrder(color):
   print(' ')
   print('Es wird ein Werkstück mit der Farbe \"' + color +'\"
bestellt.')
   print(' ')

   # Blockierungs Flag holen
   if(getSimulateFlag(1,False)):
      print('Bestellung kann nicht ausgefuehrt werden da im Moment
ein Bauteil bestellt wird.')
      print(' ')
      return 1

   # MQTT Verbindung aufbauen
   client = paho.Client(client_id)
   client.on_publish = on_publish
   client.on_connect = on_connect
   client.connect(broker, port=port)
   client.loop_start()

   while connected != True:
      time.sleep(0.1)

   # aktueller Zeitstempel
   the_time = datetime.datetime.now()
   time_stamp = the_time.strftime(format = "%Y-%m-%dT%H:%M:%S.%f")
   order = '{\"ts\":\"' + time_stamp[:-3] + 'Z\", \"type\":\"' +
color + '\"}'
   client.publish(order_topic, order, qos=1)
   time.sleep(1)
   print(' ')
# end function createOrder
```

The function createCam() shows the current camera image.

```
# Gibt das aktuelle Kamerabild zurueck
def createCam(cam):
   try:
```

```
      value = getJSONFromHDFSByKey('i_cam')
      global image_json
      image_json = json.loads(value)
      print(' ')

      if(cam == 1):
         print('Um sich das Kamerabild anzeigen zu lassen öffnen
sie folgende URL im Browser:')
         print(' ')
         print('\thttp://kubernetes-master:' + str(http_port))
         print(' ')
         print('Anzeige muss manuell mit str+c abgebrochen
werden!')
         print(' ')
         startWebServer()
      else:
         print('Letzter Zeitstempel: \t' + str(image_json['ts']))
         print('Daten: \t\t\t' + str(image_json['data'][:60]))
      print(' ')
   except:
      print("Fehler Bei HDFS Abfrage: ", sys.exc_info()[0])
      raise
# end function createCam
```

The function parse_arg() is a helper function to evaluate the passed arguments.

```
# Argumente verwaltung
def parse_args():
    parser = argparse.ArgumentParser(
        description='Benutzung python3 lernfabrik.py -XX \n' \
                    'z.B. lernfabrik.py -o RED \n' \
                    'z.B. lernfabrik.py -s hbw \n' \
                    'z.B. lernfabrik.py -t 1'
    )
    parser.add_argument(
        '-o', '--order',
        choices=colors,
        help='Bestellung aufgeben.',
    )
    parser.add_argument(
        '-s', '--state',
        choices=stations,
        help='Stations Status abfragen.',
    )
    parser.add_argument(
        '-t', '--stock',
        choices=modi,
        type=int,
        help='Status des Hochregals abfragen.',
    )
    parser.add_argument(
        '-f', '--follow',
```

```
        action='store_true',
        help='Sequentielles Abfragen aller Stati.',
    )
    parser.add_argument(
        '-c', '--cam',
        choices=modi,
        type=int,
        help='Aktuelles Kamerabild anzeigen.',
    )
    parser.add_argument(
        '-m', '--movecam',
        metavar='movecam',
        nargs=2,
        help='Kamerasteuerung. Richtung: ' + str(directions) + ';
Winkel: ' + str(angle),
    )
# end function parse_args
```

The main() function.

```
def main():
    try:
        args = parse_args()

        # Bestellung pruefen
        if(args.order): #!= None):
           createOrder(args.order)

        # Status einer Station pruefen
        if(args.state): #!= None):
           createState(args.state)

        # Status des Hochregals pruefen
        if(args.stock != None):
           createStock(args.stock)

        # Sequentielles Abfragen der Stati
        if(args.follow): #!= None):
           createFollow()

        # Letztes Kamerabild anzeigen
        if(args.cam != None):
           createCam(args.cam)

        # Kamera bewegen
        if(args.movecam != None):
           if(args.movecam[0] in directions and
int(args.movecam[1]) in angle):
              moveCam(args.movecam[0], args.movecam[1])
           else:
              print('Bitte die Parameter beachten: Richtung: ' +
str(directions) + '; Winkel: ' + str(angle))
```

```
    except KeyboardInterrupt:
        print(" Interrupted")
        exit(0)

if __name__ == "__main__":
    main()
```

3.8 Exercises

Connect to the university cloud as described in Module 1 and solve the following tasks:

(a) Trigger an ordering process using the Tiny client. (Section 3.1.1 / 3.5)

b) Then, cyclically determine the status of each station and the high-bay warehouse to see the passage of the workpiece. (Section 3.4)

APPENDIX A

CHAPTER 4 Module 4

Welcome to module 4.

In module 4 the data analysis with Python is explained.

Monitoring data is read from the learning factory via Apache Nifi and stored in a relational database. AWS Redshift is used for this purpose. The Redshift database runs in the Amazon Cloud (AWS). Data analysis is performed using a Python script. The script is located in a Jupyter notebook in the university cloud at the following URL: https://hivemqserver.feste-ip.net:9001 The credentials are: User: lernfabrik Password: L5nf1br|k

Learning Objectives

- Data analysis using Python for IoT data.

Materials

- Python script source code brightness sensor visualization
- Python script source code environmental sensor visualization

4.1 AWS Redshift Connector

To access the AWS Redshift database, the RedShift Connector is required. The connector is already installed. The connector can be installed with pip install redshift_connector in the Jupyter notebook.

Installation:

```
python3.9 -m pip install redshift_connector
```

With the method redshift_connector_connect a connection to the RedShift database in the AWS Cloud is established. Connection data:

```
# Verbindungsdaten
conn = redshift_connector.connect(
    host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
    port=5439,
    database='dev',
    user='lernfabrik',
    password=L5nf1br|k
)
```

4.2 SQL statement ldr

In the table lernfabrik_ldr the brightness values within the plant are stored.

The column Brightness indicates the brightness in %. The sensor value column specifies the value measured by the sensor.

The time period for which the sensor data is to be created is specified, as well as the access data of the Connector.

```
cursor: redshift_connector.Cursor = conn.cursor()
cursor.execute("select * from lernfabrik.lernfabrik_ldr where ts >
'" + \
         zeitstempel_von + "' and ts < '" + zeitstempel_bis + "'
order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()
conn.close()
```

Output of the sensor values

```
df.columns = ["Zeitstempel", "Helligkeit", "Sensorwert"]
df
```

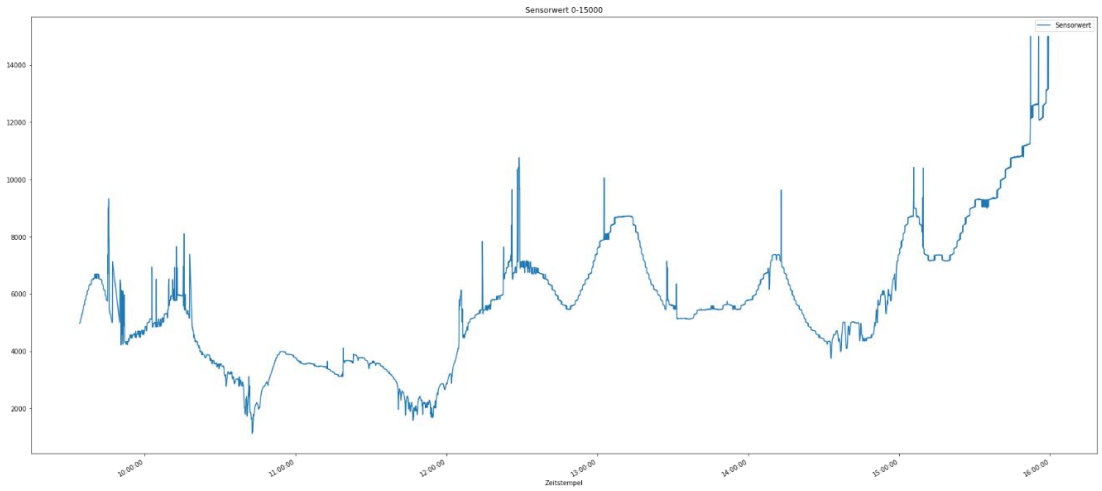
	Time stamp	Brightness	Sensor value
0	2022-02-06 15:43:42.470	0.0	15000
1	2022-02-06 15:43:39.454	12.3	13156
2	2022-02-06 15:43:36.450	12.3	13161
3	2022-02-06 15:43:27.470	0.0	15000
4	2022-02-06 15:43:24.424	12.3	13151
...	...	...	...

4.3 Brightness sensor value

The sensor value is the raw value of the brightness sensor. This is a photoresistor and has the unit Ohm. The value is between 0 - 15.000. The higher the resistor value the darker.

The select statement is executed and the data sets are returned as DataFrame. Thus the data can be visualized directly. Furthermore, more meaningful names are assigned to the column names.

```
# Sensorwert Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Sensorwert'], figsize=(30,14),
title='Sensorwert 0-15000')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



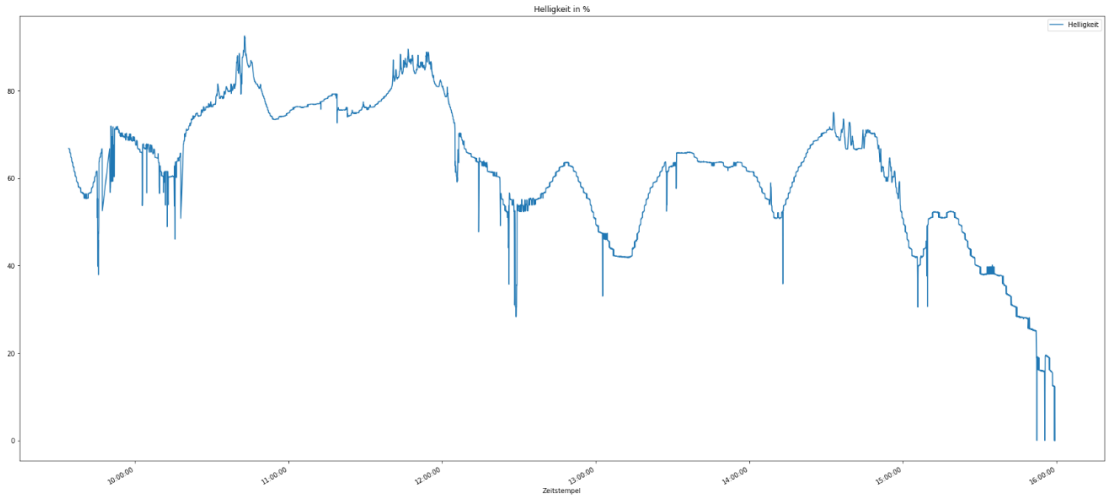
4.4 Brightness in %

Conversely, the percentage display is mirror-inverted, since for us 100% is bright and 0% is dark. Brightness in percent is a percentage representation of the raw value of the sensor.

Brightness in % = 1 – (Sensor value / 15000)

Therefore the graphs are identical, but mirrored.

```
# Helligkeit in %
ax0 = df.plot( x='Zeitstempel', y=['Helligkeit'], figsize=(30,14),
title='Helligkeit in %')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



4.5 Brightness sensor source code

lernfabrik_helligkeitssensor.py

```
import redshift_connector
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from matplotlib.dates import DateFormatter

# Zeitstempel fuer die Grafiken
zeitstempel_von = "2021-10-07 00:00:00.000"
zeitstempel_bis = "2021-10-08 00:00:00.000"

# Verbindung
conn = redshift_connector.connect(
   host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
   port=5439,
   database='dev',
   user='lernfabrik',
   password='L5nf1br|k'
)

cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_ldr where ts >
'" + zeitstempel_von + "' and ts \
   < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()
```

```
df.columns = ["Zeitstempel", "Helligkeit", "Sensorwert"]
df

%matplotlib inline

# Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Helligkeit'],
                      figsize=(30,14), title='Helligkeit in %')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)

# Sensorwert Helligkeit
ax0 = df.plot( x='Zeitstempel', y=['Sensorwert'], figsize=(30,14),
title='Sensorwert 0-15000')
ax0.xaxis.set_major_formatter(date_form)
```

4.6 SQL-Statement bme680

In the table lernfabrik_bme680 air data is saved.

CREATE TABLE
  lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
	Zeitstempel
	Sensorwert (gas resistance [Ohm])
	Luftqualität Genauigkeit [0-3]
	Luftqualität [index air quality 0-500]
	Luftdruck [hPa]
	Luftfeuchtigkeit [raw-value]
	Luftfeuchtigkeit [%]
	Temperatur [raw-value]
	Temperatur [°C]

The connector data is identical to those from the brightness sensor.

The select statement is executed. The records are returned as DataFrame. Here, the column names are also assigned more meaningful names.

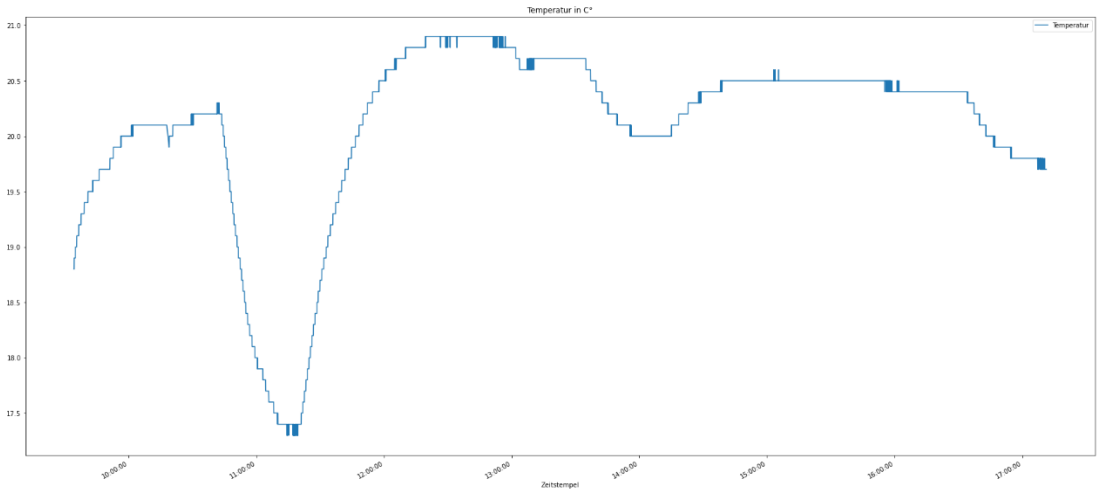
```
cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_bme680 where ts
> '" + zeitstempel_von + \
       "' and ts < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()
```

```
df.columns = ["Zeitstempel", "aq", "gr", "Luftfeuchtigkeit",
"Luftqualitaet", "Luftdruck", "rh", "rt", "Temperatur"]
df
```

	Time stamp	aq	gr	Humidity	Quality	Pressure	rh	rt	Temperature
0	2022-02-06 15:43:42.470	1	819622.0	42.7	30.0	935.3	34.77	16.20	13.0
1	2022-02-06 15:43:39.454	1	817629.0	42.7	31.0	935.3	34.77	16.20	13.0
...	...	...	...	...	...	...	...	...	...

4.7 4.7. Temperature

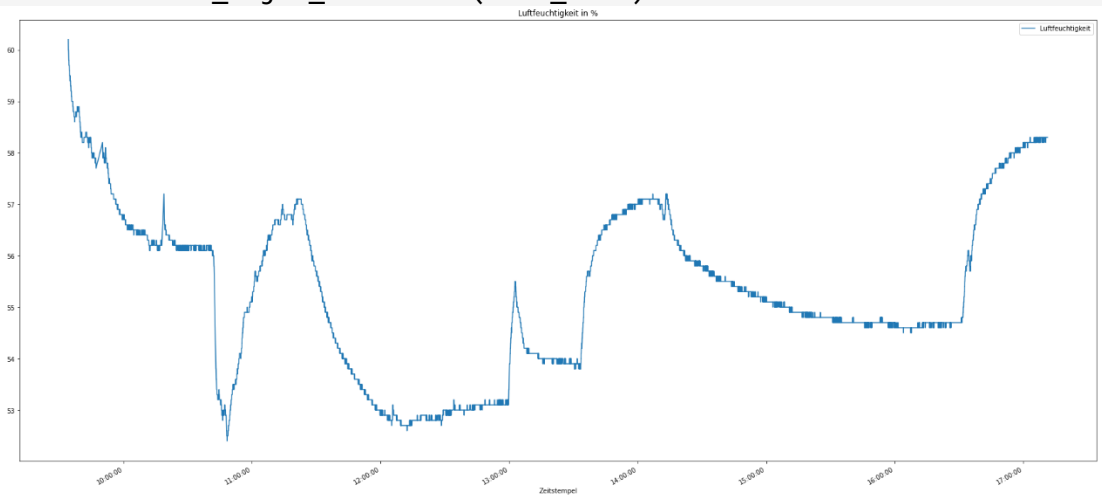
```
The following python script visualizes the temperature changes.
%matplotlib inline

# Temperatur
ax0 = df.plot( x='Zeitstempel', y=['Temperatur'],
                      figsize=(30,14), title='Temperatur in C°')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)
```



4.8 Humidity

The following python script visualizes the humidity development.

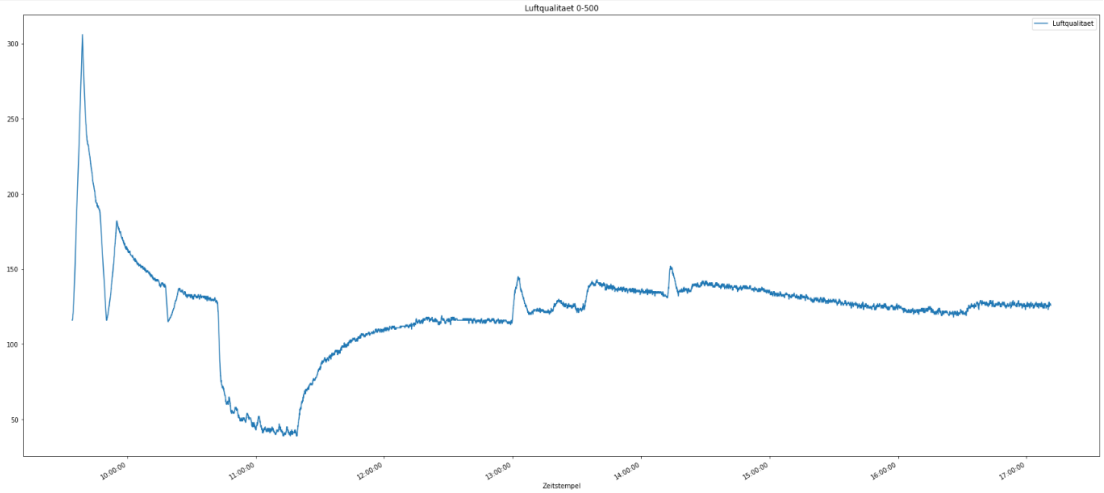
```
# Luftfeuchtigkeit
ax0 = df.plot( x='Zeitstempel', y=['Luftfeuchtigkeit'],
                        figsize=(30,14), title='Luftfeuchtigkeit in
%')
ax0.xaxis.set_major_formatter(date_form)
```



4.9 Air quality

The following python visualizes the air quality development.

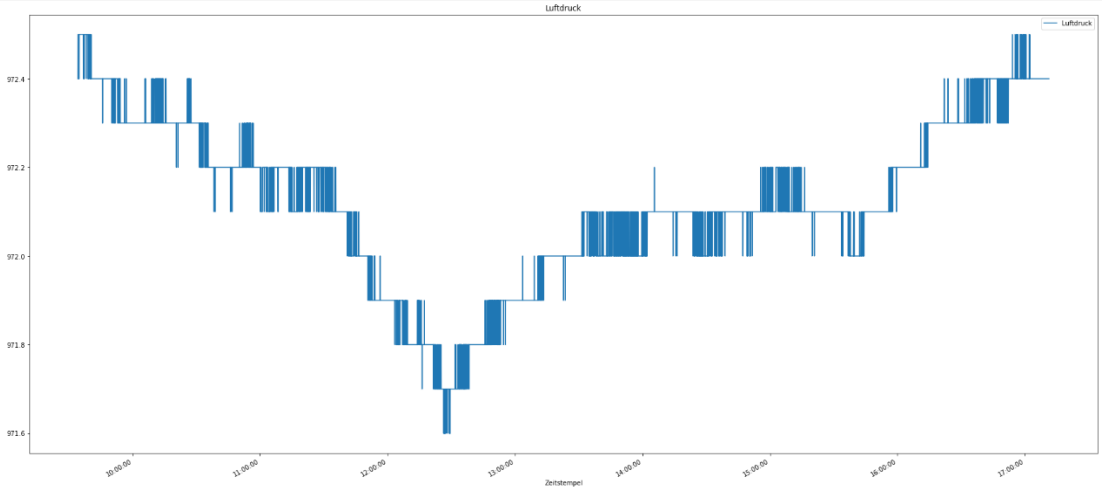
```
# Luftqualitaet
ax0 = df.plot( x='Zeitstempel', y=['Luftqualitaet'],
                        figsize=(30,14), title='Luftqualitaet 0-500')
ax0.xaxis.set_major_formatter(date_form)
```



4.10 Air pressure

The following python script visualizes the air pressure development.

```
# Luftdruck
ax0 = df.plot( x='Zeitstempel', y=['Luftdruck'],
                        figsize=(30,14), title='Luftdruck')
ax0.xaxis.set_major_formatter(date_form)
```



4.11 Environmental sensor source code

lernfabrik_umweltsensor.py

```
import redshift_connector
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from matplotlib.dates import DateFormatter

# Zeitstempel fuer die Grafiken
zeitstempel_von = "2021-10-07 00:00:00.000"
zeitstempel_bis = "2021-10-08 00:00:00.000"

# Verbindung
conn = redshift_connector.connect(
   host='redshift-cluster-1.c2yh8v1apkos.us-east-
2.redshift.amazonaws.com',
   port=5439,
   database='dev',
   user='lernfabrik',
   password='L5nf1br|k'
)
```

```
cursor: redshift_connector.Cursor = conn.cursor()

cursor.execute("select * from lernfabrik.lernfabrik_bme680 where ts
> '" + zeitstempel_von + \
       "' and ts < '" + zeitstempel_bis + "' order by ts asc")
df: pd.DataFrame = cursor.fetch_dataframe()

conn.close()

df.columns = ["Zeitstempel", "aq", "gr", "Luftfeuchtigkeit",
"Luftqualitaet", "Luftdruck", "rh", "rt", "Temperatur"]
df

%matplotlib inline

# Temperatur
ax0 = df.plot( x='Zeitstempel', y=['Temperatur'],
                      figsize=(30,14), title='Temperatur in C°')

# Datumsformat in der X-Achse anpassen
date_form = DateFormatter("%H:%M:%S")
ax0.xaxis.set_major_formatter(date_form)

# Luftfeuchtigkeit
ax0 = df.plot( x='Zeitstempel', y=['Luftfeuchtigkeit'],
                      figsize=(30,14), title='Luftfeuchtigkeit in
%')
ax0.xaxis.set_major_formatter(date_form)

# Luftqualitaet
ax0 = df.plot( x='Zeitstempel', y=['Luftqualitaet'],
                      figsize=(30,14), title='Luftqualitaet 0-500')
ax0.xaxis.set_major_formatter(date_form)

# Luftdruck
ax0 = df.plot( x='Zeitstempel', y=['Luftdruck'],
                      figsize=(30,14), title='Luftdruck')
ax0.xaxis.set_major_formatter(date_form)
```

4.12 Exercises

Display the sensor values for the period of the ordering process graphically. Use the scripts of this module as an orientation.

APPENDIX A

CHAPTER 5 Module 5

Welcome to module 5.

In module 5 we will get to know the OPC/UA communication between Node-RED and the production plant. The production plant is controlled by a Siemens PLC using OPC-UA.

Learning objectives

- Introduction to OPC/UA communication between Node-RED and a PLC.
- Read of the learning factory with Node-RED.

Materials

- Node-RED

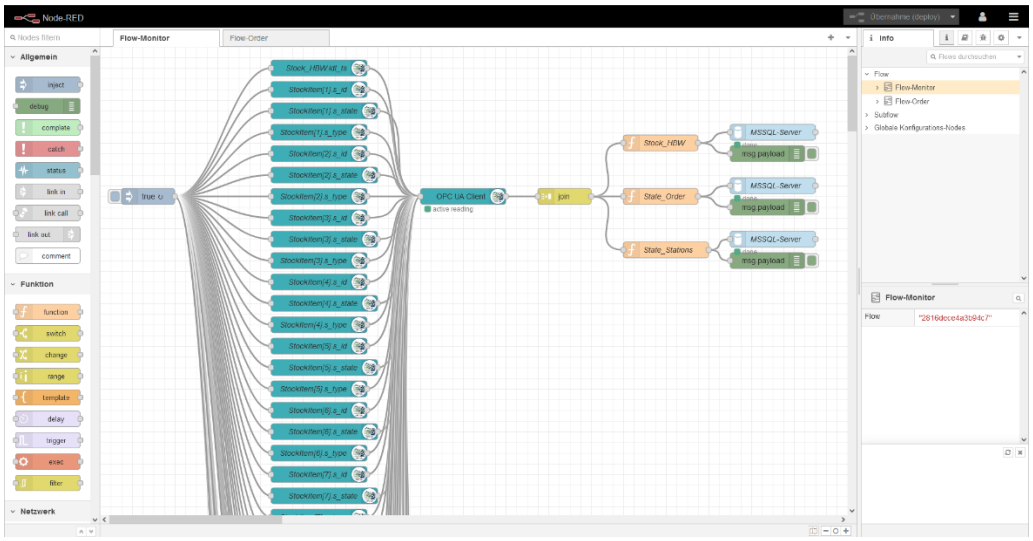
5.1 Introduction

Node-RED is a graphical open-source development tool from IBM, which makes it possible to implement various use cases in the field of IoT. For this purpose, a flow editor is available in the web browser, with which JavaScript functions can be created. In addition, numerous modules for the most common services and technologies are provided, which are connected with each other according to a simple modular principle. Each module symbolizes a node and has different tasks as input, output or processing node. The runtime environment is based on Node.js and the created flows are stored in JSON, which can be easily imported/exported or shared with others.

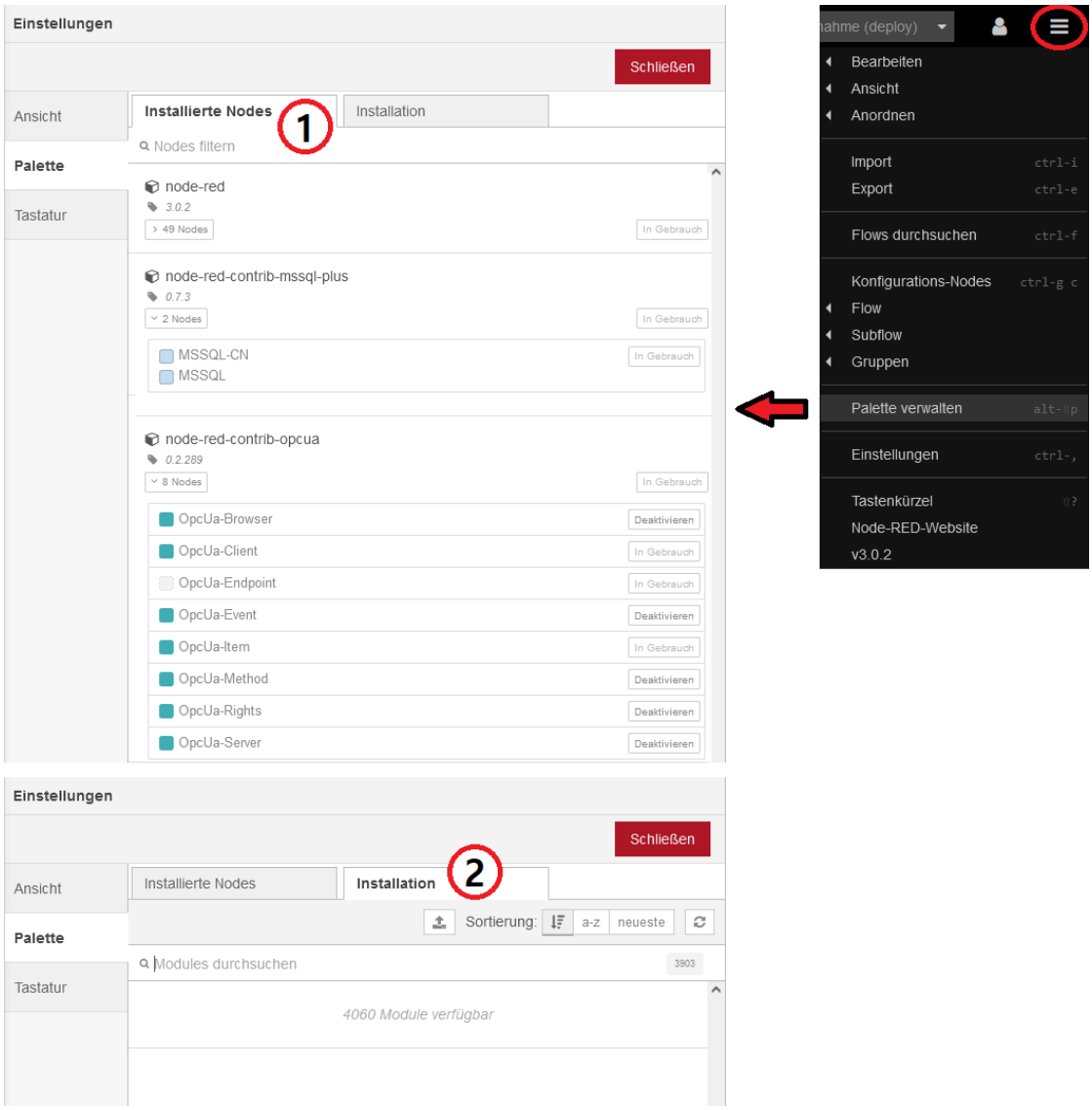
5.1.1 Flow editor

Node-RED is accessed in the web browser via the address https://hivemqserver.feste-ip.net:1880 (user: lernfabrik / password: L5nf1br|k). On the left side are the installed nodes, which are dragged and dropped into the flow editor. With a double click the

node specific settings can be defined and by connecting the different nodes the data flow can be defined.



Via the main menu (top/right icon with three stripes) under “Palette verwalten” (“manage palette”) you can access the settings of the installed nodes.



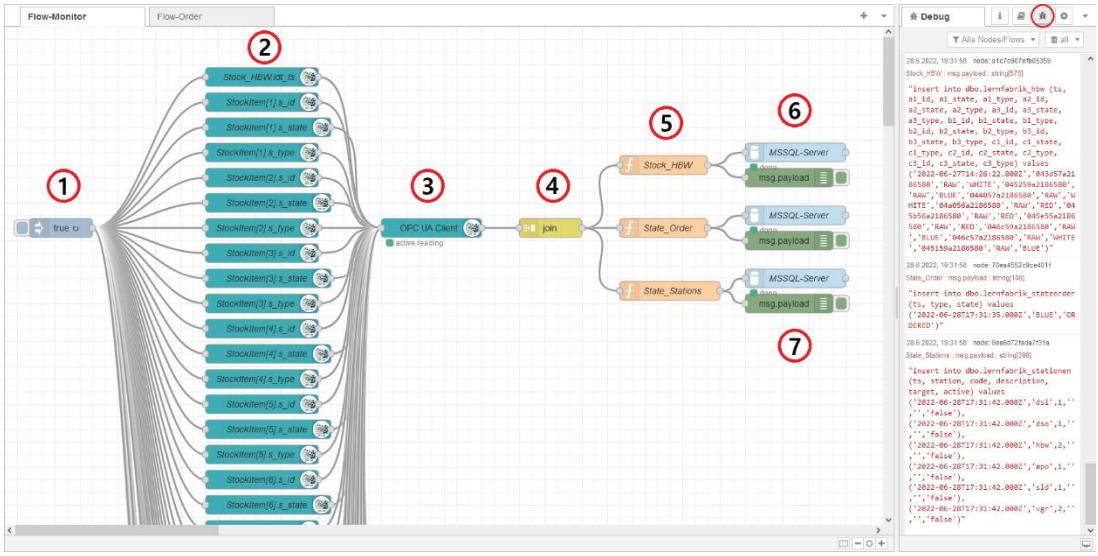
Here you get an overview of the installed nodes and can see which ones are currently in use (1). By default, 49 nodes of "node-red" are available. For our project, the nodes "node-red-contrib-opcua" for the OPC/UA communication with the learning factory and "node-red-contrib-mssql-plus" for the storage of the OPC values in our MSSQL database were additionally installed.

Additional nodes can be searched and installed in the second tab "Installation" (2). There are currently more than 4000 modules available for this purpose.

5.2 Reading from the learning factory

For reading the learning factory the flow “Flow-Monitor” was created. It consists of the following nodes:

1. inject
2. OpcUa-Item
3. OpcUa-Client
4. join
5. function
6. MSSQL
7. debug



The flow is started every 10 seconds with the "inject" node (1). The values from the "OPC items" (2) are read out with the "OPC-UA client" node (3) and written to an array with a "join" node (4).

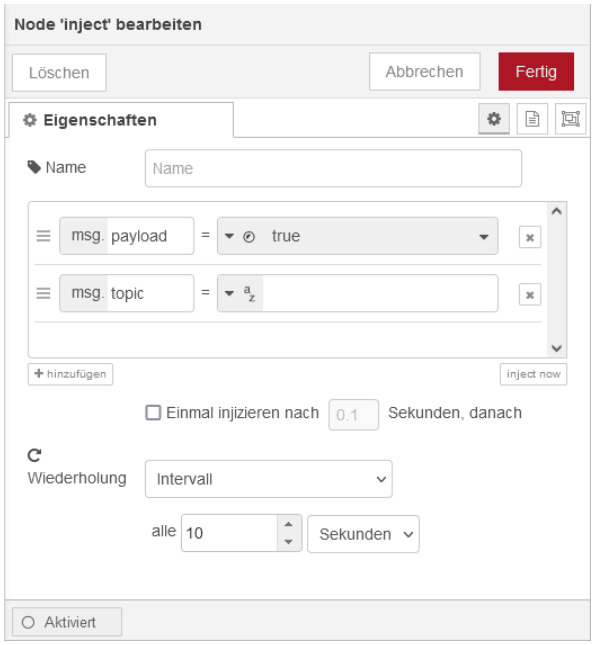
With the help of the "function" nodes (5), the values are read from the array and the corresponding SQL insert string is assembled. This is then passed to an "MSSQL" node (6) as a query, where the SQL insert is finally executed in the corresponding table.

With the "debug" nodes (7) the passed SQL insert strings are output in the right debug tab. The debug tab button is located on the right/top (fig. above - circled in red).

5.2.1 inject

The "inject" node starts the flow. This can be injected either manually with a click on the left button, at specific times or at regular time intervals. For our project a time interval of 10 seconds was chosen.

In "msg.payload" the payload of the flow is stored during the injection.



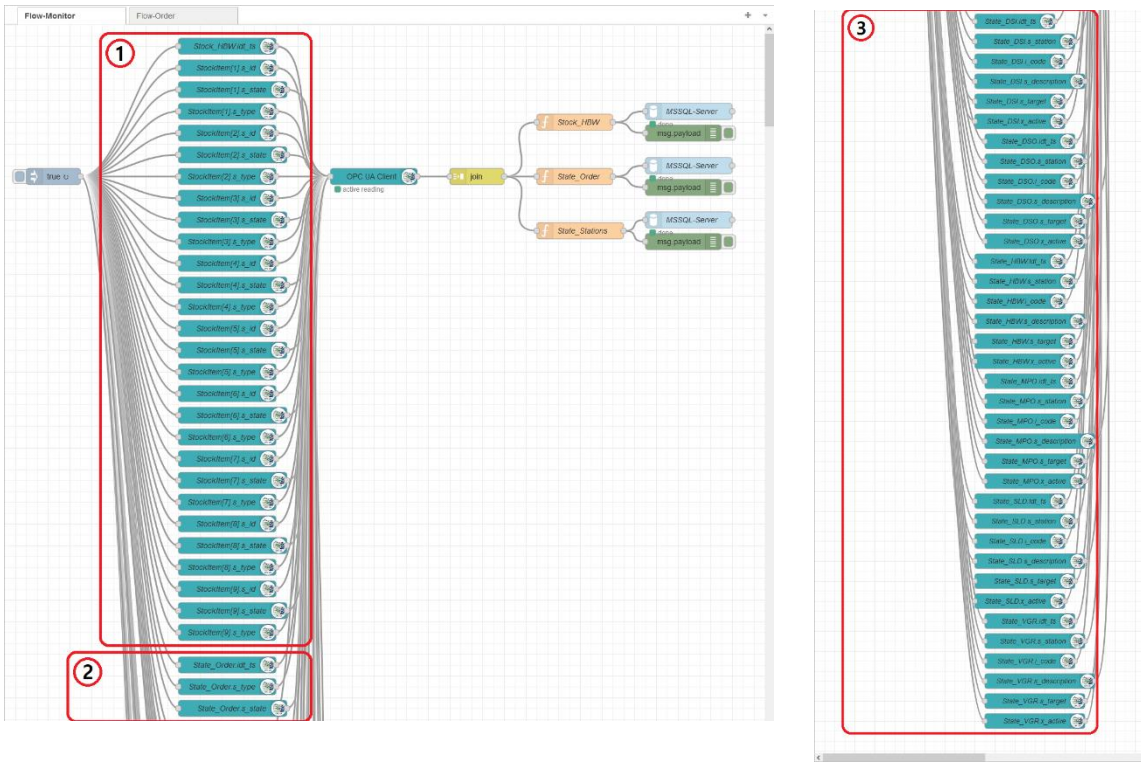
5.2.2 OpcUa item

Every single OPC data value to be read out is defined with an "OpcUa-Item" node.

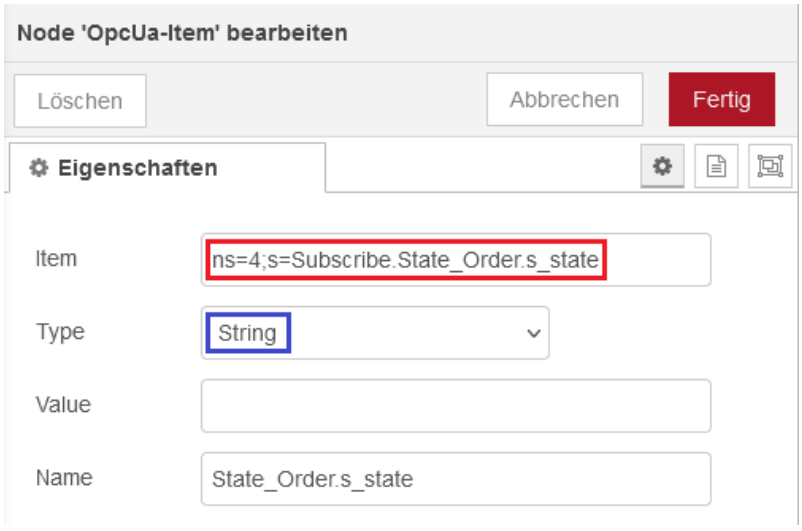
In our flow, a total of 67 values are read out. These are:

1. Stock of the high bay warehouse
2. Current order status
3. Status data of the various learning factory stations

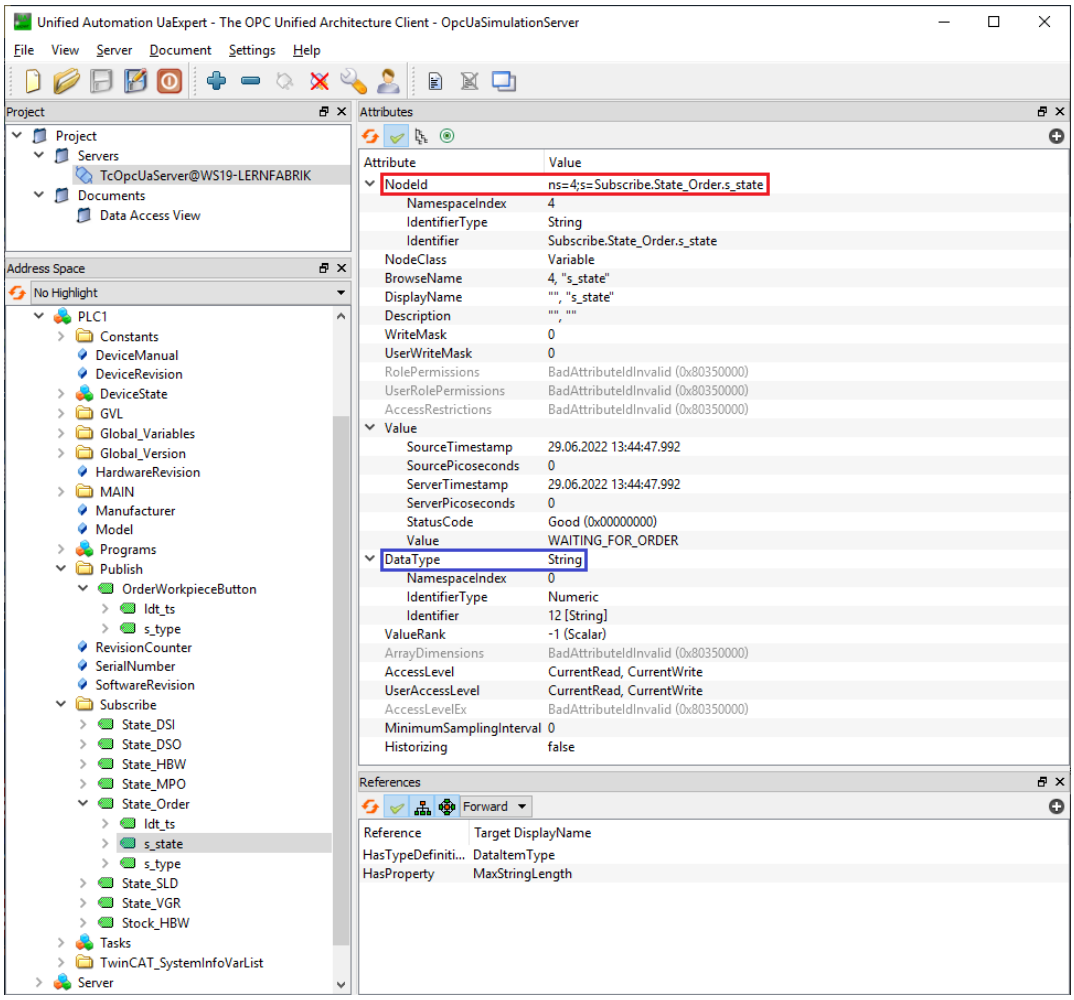
APPENDIX A



In the node settings the NodeID is given under “Item”. The NodeID consists of the Namespace-Index and the Identifier-String. Under “Type” the data type is given.

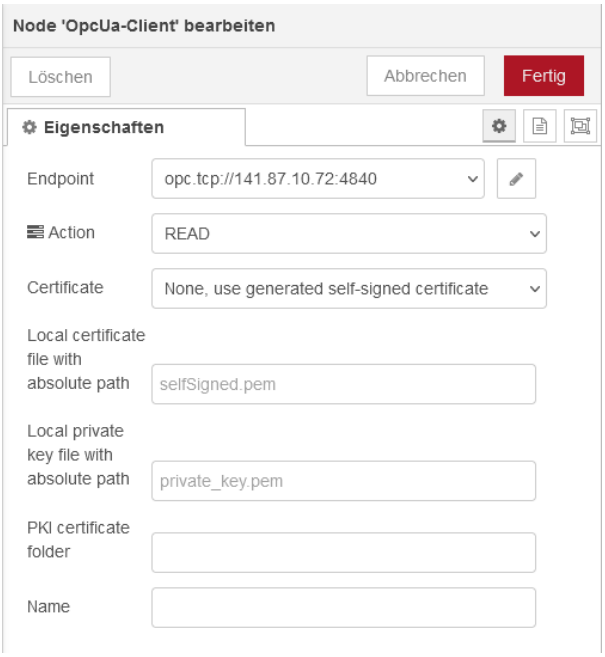


The "UA Expert" software was used to determine the values. This lists all OPC data points and values and also provides a comprehensive detailed overview:



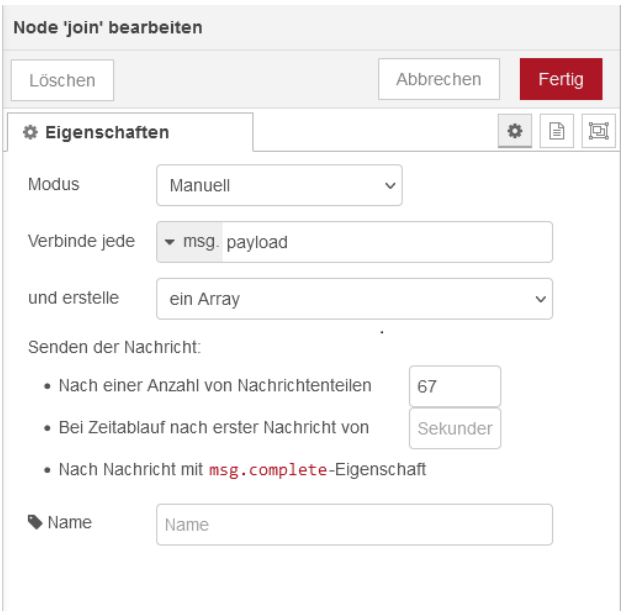
5.2.3 OpcUa-Client

The communication with the OPC/UA server is realized with the node “OpcUa-Client”. Therefore under “Endpoint” the OPC/UA server address is provided an finally the wished action is selected – in our case “READ” to read the values.



5.2.4 join

With the help of the “join” nodes the single 67 read OPC values are combined to an array “msg.payload[0-66]”.



5.2.5 function

With the "function" nodes the values are read from the array "msg.payload[0-66]", the corresponding SQL insert string is created, and finally forwarded as a message object (msg1). The message object contains a "topic" with the node name to make the output in the debug tab clearer and a "payload" with the created SQL insert string.

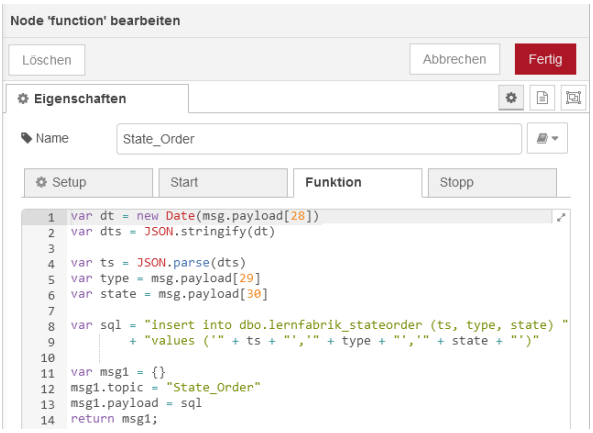
Stock_HBW:

APPENDIX A


```
var dt = new Date(msg.payload[0])
var dts = JSON.stringify(dt)

var ts = JSON.parse(dts)
var a1_id = msg.payload[1]
var a1_state = msg.payload[2]
var a1_type = msg.payload[3]
var a2_id = msg.payload[4]
var a2_state = msg.payload[5]
var a2_type = msg.payload[6]
var a3_id = msg.payload[7]
var a3_state = msg.payload[8]
var a3_type = msg.payload[9]
var b1_id = msg.payload[10]
var b1_state = msg.payload[11]
var b1_type = msg.payload[12]
var b2_id = msg.payload[13]
var b2_state = msg.payload[14]
var b2_type = msg.payload[15]
var b3_id = msg.payload[16]
var b3_state = msg.payload[17]
var b3_type = msg.payload[18]
var c1_id = msg.payload[19]
var c1_state = msg.payload[20]
var c1_type = msg.payload[21]
var c2_id = msg.payload[22]
var c2_state = msg.payload[23]
var c2_type = msg.payload[24]
var c3_id = msg.payload[25]
var c3_state = msg.payload[26]
var c3_type = msg.payload[27]

var sql = "insert into dbo.lernfabrik_hbw (ts, "
        + "a1_id, a1_state, a1_type, "
        + "a2_id, a2_state, a2_type, "
        + "a3_id, a3_state, a3_type, "
        + "b1_id, b1_state, b1_type, "
        + "b2_id, b2_state, b2_type, "
        + "b3_id, b3_state, b3_type, "
        + "c1_id, c1_state, c1_type, "
        + "c2_id, c2_state, c2_type, "
        + "c3_id, c3_state, c3_type) "
        + "values ('" + ts
        + "','" + a1_id + "','" + a1_state + "','" + a1_type
        + "','" + a2_id + "','" + a2_state + "','" + a2_type
        + "','" + a3_id + "','" + a3_state + "','" + a3_type
        + "','" + b1_id + "','" + b1_state + "','" + b1_type
        + "','" + b2_id + "','" + b2_state + "','" + b2_type
        + "','" + b3_id + "','" + b3_state + "','" + b3_type
        + "','" + c1_id + "','" + c1_state + "','" + c1_type
        + "','" + c2_id + "','" + c2_state + "','" + c2_type
        + "','" + c3_id + "','" + c3_state + "','" + c3_type + "')

var msg1 = {}
msg1.topic = "Stock_HBW"
msg1.payload = sql
return msg1;
```

State_Order:

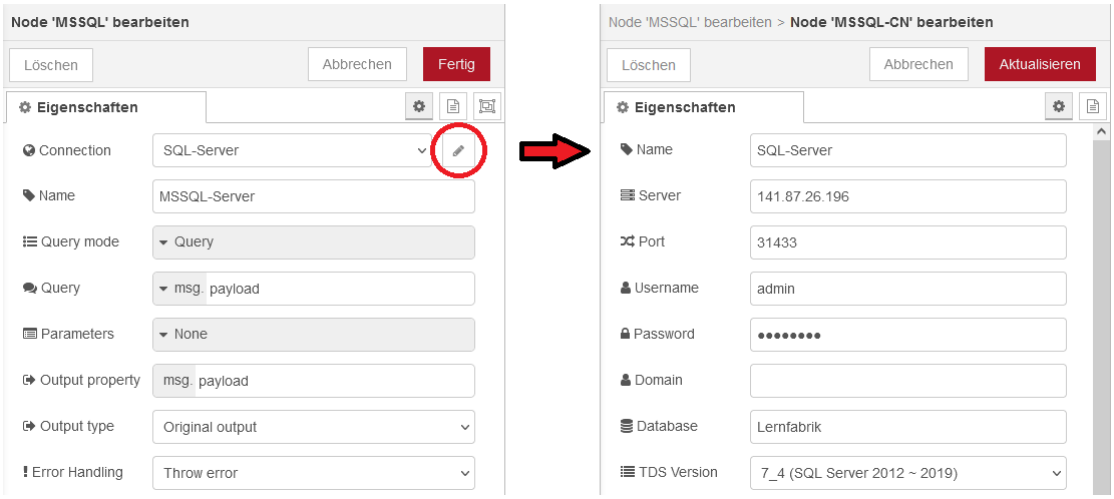
APPENDIX A



State_Stations:

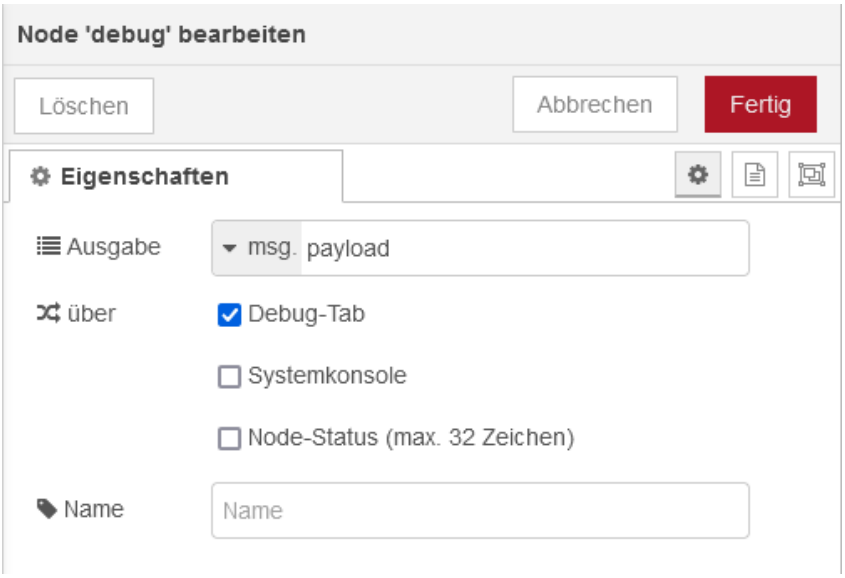
5.2.6 MSSQL

In the "MSSQL" node, the connection to the MSSQL server is defined via an MSSQL ConnectionNode. The edit button leads to the connection properties where the server and access data with the database to be used are specified.
Finally "Query" is selected as mode and with "msg.payload" the SQL statement to be executed from the "payload" string of the transferred message object is specified.



5.2.7 debug

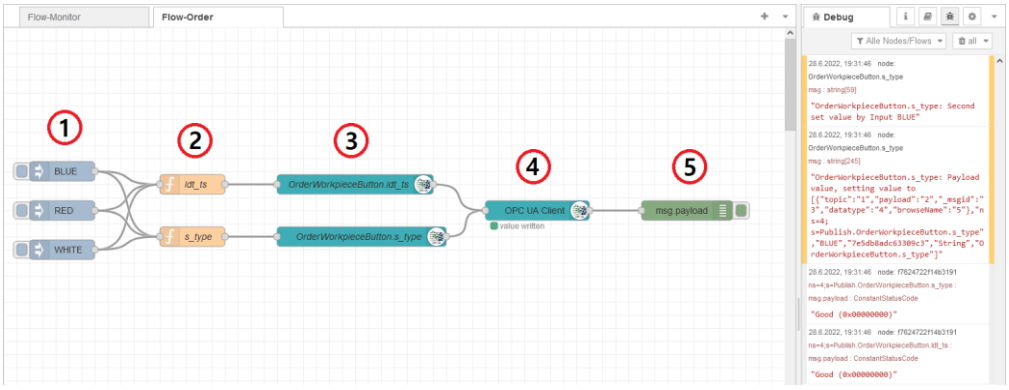
In the “debug” node the default settings with the output “msg.payload” are used by selecting the “Debug-Tab”.



5.3 Controlling the learning factory

For the open loop control of the learning factory the flow “Flow-Order” was created, that can trigger a ordering process. It consists of the following nodes:

1. inject
2. function
3. OpcUA-Item
4. OpcUa-Client
5. debug

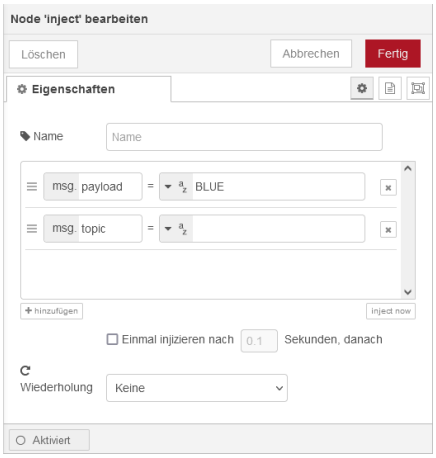


Three "inject" nodes (1) are available for the ordering process. Clicking on the left button of an "inject" node starts the ordering process with the desired workpiece color ("BLUE"/"RED"/"WHITE"). Via the function nodes (2), the time stamp (ldt_ts) and the workpiece color (s_type) are transferred to the corresponding OPC items (3), which are then sent to the OPC/UA server of the learning factory via the "OPC-UA Client" node (4).

Finally, the transferred values are output in the debug window using the "debug" node (5).

5.3.1 inject

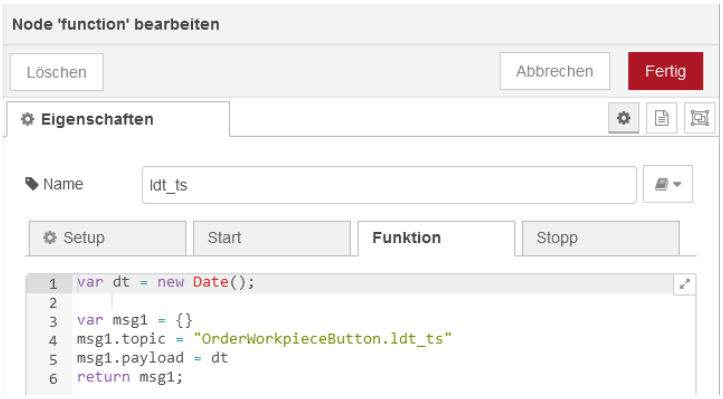
The "inject" nodes start the flow. They contain in "msg.payload" a string with the desired workpiece color. For our order flow only the manual injection is selected by switching off the automatic repetition.



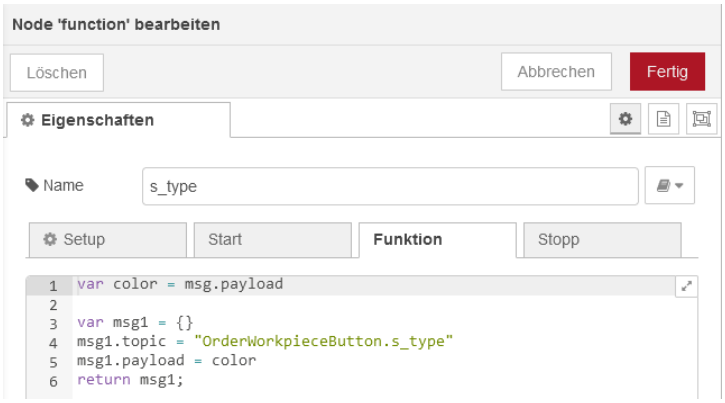
5.3.2 function

The "function" nodes are used to pass on the values required for the ordering process as a message object (msg1). These are the current time stamp in the "ldt_ts" node and the desired color, which is determined in the "s_type" node from "msg.payload".

ldt_ts:

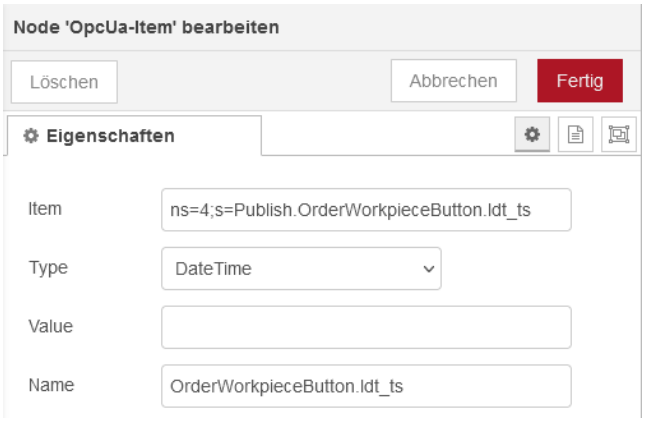


s_type

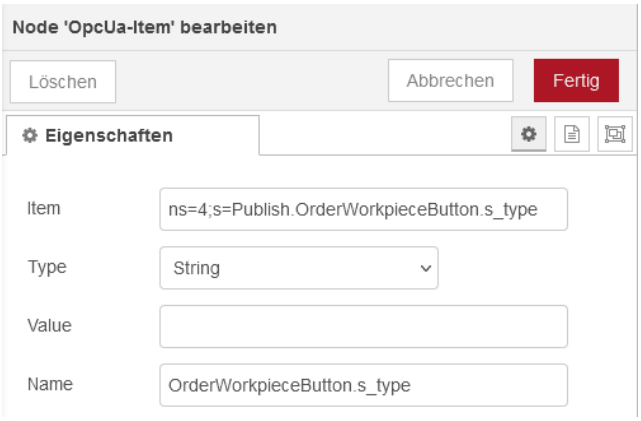


5.3.3 OpcUa-Item

OrderWorkpieceButton.ldt_ts:

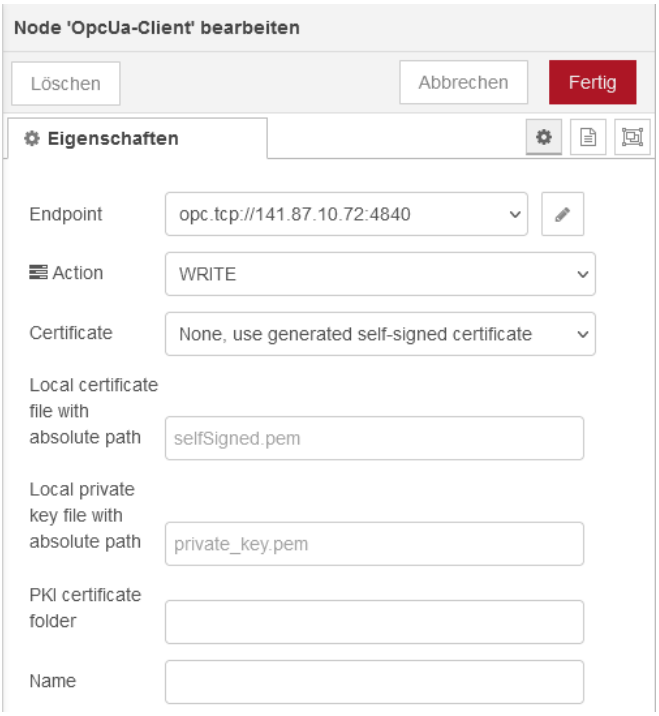


OrderWorkpieceButton.s_type:



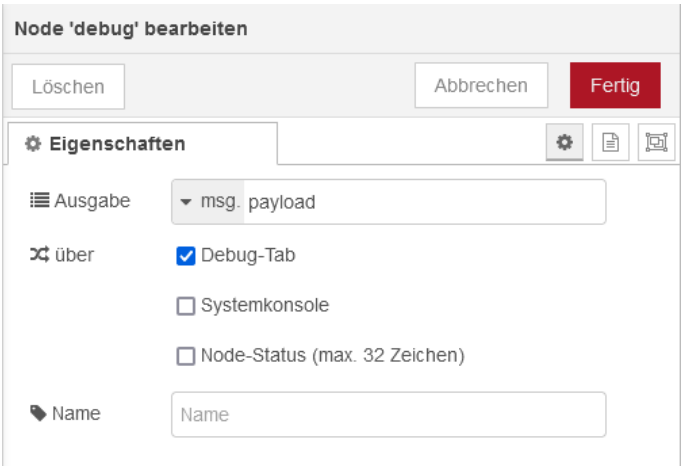
5.3.4 opcUa client

With the node "OpcUa-Client" the communication to the OPC/UA server of the learning factory simulation is realized. For this purpose, the OPC/UA server address is specified under "Endpoint" and finally the desired action is selected - in our case "WRITE" for writing the values.



5.3.5 debug

In the “debug” node the default settings with the output “msg.payload” are used by selecting the “Debug-Tab”.



5.4 Exercises

Connect to Node-Red at the following address https://hivemqserver.feste-ip.net:1880 (user: lernfabrik / password: L5nf1br|k) and solve the following tasks:

(a) Trigger an ordering process with Node-RED. (Section 5.3)

b) Determine the total duration of the order process including the putaway process using the values of the order state (State_Order) in the debug tab. (Section 5.2 / 5.3)

c) Track the transferred state changes during an order and putaway process. (Section 5.2 / 5.3)

APPENDIX A

CHAPTER 6 Module 6

Welcome to module 6.

Module 6 describes the database systems used and the data schema.

6.1 Nifi

The metadata for the two tables lernfabrik_bme680 and lernfabrik_ldr are described here.

Before the data is saved to the database using Nifi, the tables must be created. The tables are already created, the statements are only shown for completeness.

```
CREATE SCHEMA lernfabrik;
```

Contains all values of the environmental sensor

Enthält alle Werte des Umweltsensors

```
CREATE TABLE
lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Sensorwert (gas resistance [Ohm])
Luftqualität Genauigkeit [0-3]
Luftqualität [index air quality 0-500]
Luftdruck [hPa]
Luftfeuchtigkeit [raw-value]
Luftfeuchtigkeit [%]
Temperatur [raw-value]
Temperatur [°C]
```

Time stamp
Sensor value
Air quality precision
Air quality index
Air pressure
Raw humidity
Humidity
Raw temperature
Temperature

lernfabrik_bme680 (environmental sensor)

ts	gr	aq	iaq	p	rh	h	rt	t
29.09.2021 09:56:07	185450	3	144	966,799987792969	49,5200004577637	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:16	186598	3	142	966,900024414063	49,5	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:28	185762	3	144	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:34	185867	3	142	966,900024414063	49,5099983215332	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:49	185762	3	143	966,799987792969	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:58	187554	3	141	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:57:07	187013	3	142	966,900024414063	49,5	60	25,5900001525879	22,3999996185303

lernfabrik_ldr (brightness sensor)

APPENDIX A

Contains all values of the brightness sensor

Enthält alle Werte des Helligkeitssensors	

CREATE TABLE
lernfabrik.lernfabrik_ldr (
  ts datetime NOT NULL,
  ldr SMALLINT NOT NULL,
  br DOUBLE PRECISION NOT NULL
);
	Zeitstempel
	Light Dependent Resistor (0-15000[Ohm])
	Helligkeit [%] (wird aus ldr ermittelt)
	[ldr: 15000 = 0%; 7500 = 50%; 0 = 100%]

Time stamp
Brightness is determined from ldr value

ts	ldr	br
29.09.2021 09:56:04	4611	69,3000030517578
29.09.2021 09:56:16	4707	68,5999984741211
29.09.2021 09:56:28	4694	68,6999969482422
29.09.2021 09:56:34	4702	68,6999969482422
29.09.2021 09:56:49	4710	68,5999984741211
29.09.2021 09:56:58	4710	68,5999984741211
29.09.2021 09:57:07	4707	68,5999984741211

For the station on Hive the following table is created:

Contains relevant data for the stations

Enthält stationsrelevante Daten	

CREATE TABLE
lernfabrik_stationen(
  ts string,
  station string,
  code int,
  description string,
  target string,
  active boolean,
primary key(station) disable novalidate
);
	Zeitstempel
	Station (s. Tabelle unten)
	Acknowledgment-Code (s. Tabelle unten)
	Beschreibung
	Zielstation (s. Tabelle unten)
	Active-Flag [Aktiv=True; Inaktiv=False]

Time stamp
Station
Ack-Code (see below)
Description
Target station
Active flag

lernfabrik_stationen

ts	station	code	description	target	active
2021-10-21T09:39:59.088Z	dso	1			False
2021-10-21T09:40:01.094Z	dsi	1			False
2021-10-21T09:40:01.680Z	mpo	1			False
2021-10-21T09:40:02.111Z	hbw	1			False
2021-10-21T09:40:02.352Z	sld	0			True

Acknowledgment-Codes

Station	Beschreibung	Acknowledgment-Codes
dsi	Delivery and Pickup Station (Input)	-
dso	Delivery and Pickup Station (Output)	-
hbw	High-Bay Warehouse (Hochregallager)	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
mpo	Multi Processing Station (Bearbeitung)	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
sld	Sorting Line (Sortierstation)	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_SORTED
vgr	Vacuum Gripper Robot (Sauggreifer)	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

You can find these data in the HDFS directory /user/lernfabrik.

6.2 6.2 Node-Red

In Node-RED, the OPC data is written to a Microsoft SQL server cluster of the university cloud. Here, the metadata of the respective tables are described.
The following MSSQL tables were created for Node-RED:

APPENDIX A

Contains information to the storage level of the 9 containers in the high-bay warehouse

Enthält Infos zum Lagerbestand der 9 Container im Hochregallager

```
CREATE TABLE lernfabrik_hbw (
  ts datetime NOT NULL,
  a1_id VARCHAR(14),
  a1_state VARCHAR(9),
  a1_type VARCHAR(5),
  a2_id VARCHAR(14),
  a2_state VARCHAR(9),
  a2_type VARCHAR(5),
  a3_id VARCHAR(14),
  a3_state VARCHAR(9),
  a3_type VARCHAR(5),
  b1_id VARCHAR(14),
  b1_state VARCHAR(9),
  b1_type VARCHAR(5),
  b2_id VARCHAR(14),
  b2_state VARCHAR(9),
  b2_type VARCHAR(5),
  b3_id VARCHAR(14),
  b3_state VARCHAR(9),
  b3_type VARCHAR(5),
  c1_id VARCHAR(14),
  c1_state VARCHAR(9),
  c1_type VARCHAR(5),
  c2_id VARCHAR(14),
  c2_state VARCHAR(9),
  c2_type VARCHAR(5),
  c3_id VARCHAR(14),
  c3_state VARCHAR(9),
  c3_type VARCHAR(5)
);
```

```
Zeitstempel
RFID ['123456789ABCDE']
Status ['RAW'/'PROCESSED']
Werkstückfarbe ['BLUE'/'RED'/'WHITE']
…
```

Time stamp

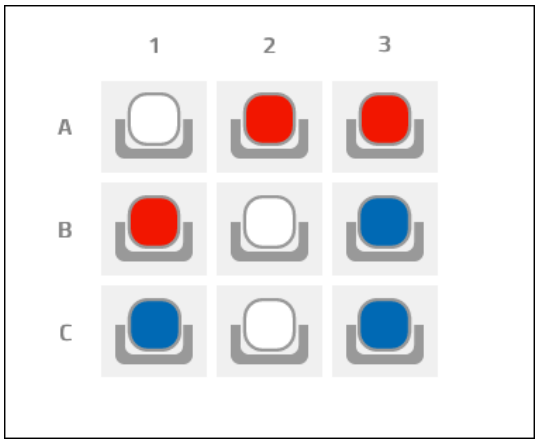
RFID

State

Work piece color

lernfabrik_hbw (high-bay warehouse)

#	ts	a1_id	a1_state	a1_type	a2_id	a2_state	a2_type	a3_id	a3_state	a3_type	b1_id	b1_state	b1_type	b2_id	b2_state	b2_type	b3_id	b3_state	b3_type	c1_id	c1_state	c1_type	c2_id	c2_state	c2_type	c3_id	c3_state	c3_type
1	2022-07-05 16:26:30.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
2	2022-07-05 16:26:30.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
3	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
4	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
5	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
6	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
7	2022-07-05 16:32:25.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
8	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
9	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
10	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
11	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
12	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
13	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
14	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
15	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
16	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
17	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
18	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
19	2022-07-05 16:33:14.000	043d57a2186580	RAW	WHITE	0			045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
20	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
21	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
22	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
23	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
24	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
25	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
26	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
27	2022-07-05 16:35:16.000	043d57a2186580	RAW	WHITE				045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
28	2022-07-05 16:36:35.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE
29	2022-07-05 16:36:35.000	043d57a2186580	RAW	WHITE	04a056a2186580	RAW	RED	045b56a2186580	RAW	RED	045e55a2186580	RAW	RED	044057a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	046c59a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045199a2186580	RAW	BLUE



lernfabrik_stateorder (ordering state)

APPENDIX A

Contains ordering state

Enthält den Bestellstatus

```
CREATE TABLE lernfabrik_stateorder (
  ts datetime NOT NULL,
  type VARCHAR(5),
  state VARCHAR(20)
);
```

```
Zeitstempel
Bestellte Farbe ['BLUE'/'RED'/'WHITE']
Status ['WAITING_FOR_ORDER'/'ORDERED'/
        'IN_PROCESS'/'SHIPPED']
```

Time stamp
Ordered color
State

#	ts	type	state
1	2022-07-05 16:26:30.000		WAITING_FOR_ORDER
2	2022-07-05 16:31:56.000	RED	ORDERED
3	2022-07-05 16:33:02.000	RED	IN_PROCESS
4	2022-07-05 16:34:42.000	RED	SHIPPED
5	2022-07-05 16:36:35.000		WAITING_FOR_ORDER

Contains relevant data for the stations

Enthält stationsrelevante Daten

```
CREATE TABLE lernfabrik_stationen (
  ts datetime NOT NULL,
  station VARCHAR(3),
  code smallint,
  description VARCHAR(20),
  target VARCHAR(3),
  active BIT DEFAULT 'false'
);
```

```
Zeitstempel
Station (s. Tabelle unten)
Acknowledgment-Code (s. Tabelle unten)
Beschreibung
Zielstation (s. Tabelle unten)
Active-Flag [Aktiv=True; Inaktiv=False]
```

Time stamp
Station
Ack-Code
(see below)
Description
Target station
Active flag

lernfabrik_stationen (state of stations)

#	ts	station	code	description	target	active
1	2022-07-05 16:33:55.000	dsi	1			☐
2	2022-07-05 16:33:55.000	dso	1			☐
3	2022-07-05 16:33:55.000	hbw	1			☐
4	2022-07-05 16:33:55.000	mpo	1			☐
5	2022-07-05 16:33:55.000	sld	1			☑
6	2022-07-05 16:33:55.000	vgr	2			☐
7	2022-07-05 16:34:16.000	dsi	1			☐
8	2022-07-05 16:34:16.000	dso	1			☐
9	2022-07-05 16:34:16.000	hbw	1			☐
10	2022-07-05 16:34:16.000	mpo	1			☐
11	2022-07-05 16:34:16.000	sld	1			☐
12	2022-07-05 16:34:16.000	vgr	2		dso	☑

Acknowldgment-Codes:

Beschreibung	Station	Acknowledment-Codes
Delivery **S**tation **I**nput	**dsi**	0 = *Lichtschranke unterbrochen*
Delivery **S**tation **O**utput	**dso**	1 = *Lichtschranke nicht unterbrochen*
High-**B**ay **W**arehouse	**hbw**	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
Multi **P**rocessing Station with **O**ven	**mpo**	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
Sorting **L**ine with Color **D**etection	**sld**	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_PRODUCED
Vacuum **G**ripper **R**obot	**vgr**	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

6.3 OPC-Router

The OPC router writes the data to a Microsoft SQL server cluster in the university cloud. Here the metadata of the respective tables are described.

The following MSSQL tables are created for the OPC router:

Contains all values of the environmental sensor

Enthält alle Werte des Umweltsensors

```
CREATE TABLE
lernfabrik.lernfabrik_bme680 (
  ts datetime NOT NULL,
  gr DOUBLE PRECISION NOT NULL,
  aq SMALLINT NOT NULL,
  iaq SMALLINT NOT NULL,
  p DOUBLE PRECISION NOT NULL,
  rh DOUBLE PRECISION NOT NULL,
  h DOUBLE PRECISION NOT NULL,
  rt DOUBLE PRECISION NOT NULL,
  t DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Sensorwert (gas resistance [Ohm])
Luftqualität Genauigkeit [0-3]
Luftqualität [index air quality 0-500]
Luftdruck [hPa]
Luftfeuchtigkeit [raw-value]
Luftfeuchtigkeit [%]
Temperatur [raw-value]
Temperatur [°C]
```

Time stamp
Sensor value
Air quality precision
Air quality index
Air pressure
Raw humidity
Humidity
Raw temperature
Temperature

lernfabrik_bme680 (environmental sensor)

ts	gr	aq	iaq	p	rh	h	rt	t
29.09.2021 09:56:07	185450	3	144	966,799987792969	49,5200004577637	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:16	186598	3	142	966,900024414063	49,5	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:28	185762	3	144	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:34	185867	3	142	966,900024414063	49,5099983215332	59,9000015258789	25,5900001525879	22,3999996185303
29.09.2021 09:56:49	185762	3	143	966,799987792969	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:56:58	187554	3	141	966,900024414063	49,5099983215332	60	25,5900001525879	22,3999996185303
29.09.2021 09:57:07	187013	3	142	966,900024414063	49,5	60	25,5900001525879	22,3999996185303

APPENDIX A

Contains all values of the brightness sensor

Enthält alle Werte des Helligkeitssensors

```
CREATE TABLE
lernfabrik.lernfabrik_ldr (
  ts datetime NOT NULL,
  ldr SMALLINT NOT NULL,
  br DOUBLE PRECISION NOT NULL
);
```

```
Zeitstempel
Light Dependent Resistor (0-15000[Ohm])
Helligkeit [%] (wird aus ldr ermittelt)
[ldr: 15000 = 0%; 7500 = 50%; 0 = 100%]
```

Time stamp
Brightness is determined from ldr value

lernfabrik_ldr (brightness sensor)

ts	ldr	br
29.09.2021 09:56:04	4611	69,3000030517578
29.09.2021 09:56:16	4707	68,5999984741211
29.09.2021 09:56:28	4694	68,6999969482422
29.09.2021 09:56:34	4702	68,6999969482422
29.09.2021 09:56:49	4710	68,5999984741211
29.09.2021 09:56:58	4710	68,5999984741211
29.09.2021 09:57:07	4707	68,5999984741211

Contains information to the storage level of the 9 containers in the high-bay warehouse

Enthält Infos zum Lagerbestand der 9 Container im Hochregallager

```
CREATE TABLE lernfabrik_hbw (
  ts datetime NOT NULL,
  a1_id VARCHAR(14),
  a1_state VARCHAR(9),
  a1_type VARCHAR(5),
  a2_id VARCHAR(14),
  a2_state VARCHAR(9),
  a2_type VARCHAR(5),
  a3_id VARCHAR(14),
  a3_state VARCHAR(9),
  a3_type VARCHAR(5),
  b1_id VARCHAR(14),
  b1_state VARCHAR(9),
  b1_type VARCHAR(5),
  b2_id VARCHAR(14),
  b2_state VARCHAR(9),
  b2_type VARCHAR(5),
  b3_id VARCHAR(14),
  b3_state VARCHAR(9),
  b3_type VARCHAR(5),
  c1_id VARCHAR(14),
  c1_state VARCHAR(9),
  c1_type VARCHAR(5),
  c2_id VARCHAR(14),
  c2_state VARCHAR(9),
  c2_type VARCHAR(5),
  c3_id VARCHAR(14),
  c3_state VARCHAR(9),
  c3_type VARCHAR(5)
);
```

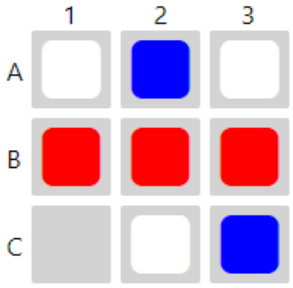
```
Zeitstempel
RFID ['123456789ABCDE']
Status ['RAW'/'PROCESSED']
Werkstückfarbe ['BLUE'/'RED'/'WHITE']
…
```

Time stamp
RFID
State
Work piece color

lernfabrik_hbw (high-bay warehouse)

ts	a1_id	a1_state	a1_type	a2_id	a2_state	a2_type	a3_id	a3_state	a3_type	b1_id	b1_state	b1_type	b2_id	b2_state	b2_type	b3_id	b3_state	b3_type	c1_id	c1_state	c1_type	c2_id	c2_state	c2_type	c3_id	c3_state	c3_type
18.02.2022 14:36:35	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	0			046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:25	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	0			046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:17	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:36:07	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:35:57	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED				046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE
18.02.2022 14:35:47	043d57a2186580	RAW	WHITE	045259a2186580	RAW	BLUE	044057a2186580	RAW	WHITE	04a056a2186580	RAW	RED	049b56a2186580	RAW	RED	045e55a2186580	RAW	RED	046c58a2186580	RAW	BLUE	046c57a2186580	RAW	WHITE	045159a2186580	RAW	BLUE

APPENDIX A



Contains ordering state

Enthält den Bestellstatus

```
CREATE TABLE lernfabrik_stateorder (
  ts datetime NOT NULL,
  type VARCHAR(5),
  state VARCHAR(20)
);
```

```
Zeitstempel
Bestellte Farbe ['BLUE'/'RED'/'WHITE']
Status ['WAITING_FOR_ORDER'/'ORDERED'/
        'IN_PROCESS'/'SHIPPED']
```

Time stamp
Ordered color
State

lernfabrik_stateorder (ordering state)

ts	type	state
18.02.2022 14:36:25	BLUE	IN_PROCESS
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:35:30	BLUE	ORDERED
18.02.2022 14:33:00		WAITING_FOR_ORDER

Contains relevant data for the stations

Enthält stationsrelevante Daten

```
CREATE TABLE lernfabrik_stationen (
  ts datetime NOT NULL,
  station VARCHAR(3),
  code smallint,
  description VARCHAR(20),
  target VARCHAR(3),
  active BIT DEFAULT 'false'
);
```

```
Zeitstempel
Station (s. Tabelle unten)
Acknowledgment-Code (s. Tabelle unten)
Beschreibung
Zielstation (s. Tabelle unten)
Active-Flag [Aktiv=True; Inaktiv=False]
```

Time stamp
Station
Ack-Code
(see below)
Description
Target
station
Active flag

lernfabrik_stationen (State of the stations)

ts	station	code	description	target	active
18.02.2022 14:36:25	dso	1			False
18.02.2022 14:36:25	hbw	2			False
18.02.2022 14:36:25	mpo	2			True
18.02.2022 14:36:25	vgr	2		mpo	True
18.02.2022 14:36:24	dsi	1			False
18.02.2022 14:36:23	sld	1			False

Acknowledgment-Codes:

APPENDIX A

Station	Beschreibung	Acknowledgment-Codes
dsi	Delivery and Pickup Station (Input)	-
dso	Delivery and Pickup Station (Output)	-
hbw	High-Bay Warehouse (Hochregallager)	0 = HBW_EXIT 1 = HBW_FETCHED 2 = HBW_STORED 3 = HBW_CALIB_NAV 4 = HBW_CALIB_END
mpo	Multi Processing Station (Bearbeitung)	0 = MPO_EXIT 1 = MPO_STARTED 2 = MPO_PRODUCED
sld	Sorting Line (Sortierstation)	0 = SLD_EXIT 1 = SLD_STARTED 2 = SLD_SORTED
vgr	Vacuum Gripper Robot (Sauggreifer)	0 = VGR_EXIT 1 = VGR_HBW_FETCHCONTAINER 2 = VGR_HBW_STORE_WP 3 = VGR_HBW_FETCH_WP 4 = VGR_HBW_STORECONTAINER 5 = VGR_HBW_RESETSTORAGE 6 = VGR_HBW_CALIB 7 = VGR_MPO_PRODUCE 8 = VGR_SLD_START

APPENDIX A

CHAPTER 7 Module 7

Welcome to Module 7.

In Module 7 we will learn how to create a neural network using a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) and how to prepare the data for this neural network model.

Learning Objectives

- Preparation of the data so that TensorFlow can handle it.
- Creating and training a neural network model.
- Application of a trained model in a live environment.

Materials

- various Python scripts for data preparation
- TensorFlow CNN model
- TensorFlow RNN model

7.1 Neural Network Model as CNN and RNN with Tensorflow and Keras

We try to use the data produced by the learning factory by means of a neural network model to automatically determine the current state of the learning factory. We take the MQTT messages that are sent every two seconds for each topic as our data.

Unfortunately, it's not like we take a bunch of big data, in our case the messages, put it into the model and say we want to get a certain result. It's not that simple. The data has to be processed and cleaned beforehand so that the model can handle it at all.

7.2 System requirement

A Jupyter Notebook is required for training. In our case, it is an Ubuntu 20 machine with Jupyter Notebook already set up with all the necessary modules.

The Tensorflow module is easily installed with the pip command. Keras is already included.

```
pip install --upgrade tensorflow
```

7.3 Data collection

We first write a simple Python script with which we record all MQTT messages of the learning factory during an order and write them to a file. Then we don't have to permanently run an order when developing the model afterwards, but can always refer back to the recorded raw data.

The Python script is located on the kubernetes-master machine in the folder:

/home/lernfabrik/ki-campus/modul_7/mqtt_client_hs_threads_stati_raw.py

Below is the function that writes the messages to a file. The messages are already in JSON format. We just add the topic in front of it and save the whole thing. Because of the JSON format, we can then quickly and easily go through the data.

```
# prueft den Inhalt der Message, nur wenn neuer inhalt update
def check_message_payload(message):
  try:
      data = { 'topic': message.topic,
               'message': json.loads(message.payload) }

      if print_flag: print(message.topic)
      f_out.write(json.dumps(data) + "\n")

   except Exception as e:
      if print_flag: print("Exception check_message_payload()")
      raise e
### end check_message_payload()
```

We start the script and run an order in parallel. After the order has been completed and the workpiece has been put back into storage, we end the script.

The file that is created is called:

```
data_file="/home/lernfabrik/ki-
campus/modul_7/data_file_big_raw.json"
```

Example:

```
{"topic": "f/i/state/hbw",
"message": {"ts": "2022-04-06T18:43:06.453Z", "station": "hbw",
"code": 1, "description": "", "active": false, "target": ""}}
```

7.4 Data preparation first part

If we go into the raw data, we see that the messages are not sorted by timestamp. Below is an example:

```
{"topic": "f/i/state/vgr",
"message": {"ts": "2022-04-06T18:43:08.792Z", "station": "vgr",
"code": 1, "description": "", "active": false, "target": ""}}
{"topic": "f/i/state/sld",
"message": {"ts": "2022-04-06T18:43:07.542Z", "station": "sld",
"code": 1, "description": "", "active": false, "target": ""}}
```

This is probably due to the fact that the intervals between the topics are sometimes only a few milliseconds and the data therefore gets a bit mixed up when writing. But this is not a problem. We write a simple script which sorts the data correctly according to the timestamps.

The script is located on the kubernetes-master machine under Jupyterhub in the following folder:

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CrearteFileOrderedByTime.ipynb

The script reads all data into a list:

```
import numpy as np
import json
import pandas as pd

# Datei einlesen
data_file = "data_file_big_raw_blue_without_cam_new.json"
f_in = open(data_file, "r")
file = []

for line in f_in:
    line = line.strip()
    file.append(line)

f_in.close()

print(type(file))
print(len(file))
file
```

```
<class 'list'>
1107
['{"topic": "f/i/state/vgr", "message":
```

```
{"ts": "2022-04-06T13:30:17.130Z", "station": "vgr", "code": 1,
"description": "", "active": false, "target": ""}}',
'{"topic": "f/i/state/sld", "message":
{"ts": "2022-04-06T13:30:16.829Z", "station": "sld", "code": 1,
"description": "", "active": false, "target": ""}}',
'{"topic": "f/i/state/mpo", "message":
{"ts": "2022-04-06T13:30:17.121Z", "station": "mpo", "code": 1,
"description": "", "active": false, "target": ""}}' …]
```

Then we create a DataFrame with two columns and insert the data into the DataFrame and write the timestamp in the first column.

```
# DataFrame bilden
df = pd.DataFrame(columns=['Zeitstempel', 'Message'])

for line in file:
    dict_value = json.loads(line)
    df = df.append({'Zeitstempel': dict_value['message']['ts'],
'Message': dict_value}, ignore_index=True)
df
```

	Time stamp	**Message**
0	2022-02-06 15:43:42.470	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
1	2022-02-06 15:43:39.454	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
...	...	...

Now we can easily sort the date after the time stamp

```
# Nach Zeitstempel sortieren
df_sort = df.sort_values(by=['Zeitstempel'])
df_sort
```

	Time stamp	**Message**
23	2022-02-06 15:43:42.470	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
4	2022-02-06 15:43:39.454	{'topic': 'f/i/state/vgr', 'message': {'ts': '...
...	...	...

The data is now sorted and can be written into a file. To do so we have to change the DataFrame into a simple list.

```
# Aus DataFrame wieder Array umwandeln in neu Datei schreiben
np_array = df_sort['Message'].to_numpy()

data_file_2 = "data_file_big_raw_blue_without_cam_new_ordered.json"
f_out = open(data_file_2, "w")

for line in np_array:
    f_out.write(json.dumps(line) + "\n")
f_out.close()
```

7.5 Data analysis

First we have to consider which messages are suitable to feed the model. The most suitable ones are the statuses that the stations themselves already send.

These are the following topics:

- f/i/state/vgr -> vacuum gripper robot
- f/i/state/sld -> sorting station
- f/i/state/mpo -> multi processing station
- f/i/state/hbw -> high-bay warehouse
- f/i/state/dsi -> input station
- f/i/state/dso -> output station

Let's first take a closer look at a message:

```
{"ts":"2022-03-18T13:26:42.391Z", "station":"dsi", "code":1,
"description":"", "target":"", "active":false}
```

We see the message has a timestamp, the name of the station, a code column, a description column, a target column and an active column. The columns ts, description and station are not interesting for us. The columns code, target and active are more interesting.

The model does not handle different data types very well. What also doesn't work well are too big distances between the input numbers e.g. 1 and 1000 or 0.001 and 1. Therefore we would have to think about how to normalize the values. The Code column can only take the states 0, 1, 2. The Active column can only accept "false" or "true", which we can convert to 0 and 1. Thus we see already at the first two columns, that small integer values would fit here best to our model. They already exist or can easily be converted to integer values.

Remains the column Target. We go into the raw data and see what values this column can take. The values are "" (empty), mpo, hbw, dso. These can also be easily converted to integer values to 0, 1, 2, 3. When looking through the raw data, we notice that the Target column is only set by the station vgr. Thus this column is omitted for all other stations.

Let's look at an intermediate state of the normalized input data.

Topic	Station	active (state)	code	target
f/i/state/vgr	Vacuum gripper robot	0,1	1,2	0,1,2,3
f/i/state/sld	Sorting line	0,1	1,2	
f/i/state/mpo	Multi processing station	0,1	1,2	
f/i/state/hbw	High-bay warehouse	0,1	1,2	
f/i/state/dsi	Input station	0,1	1,0	
f/i/state/dso	Output station	0,1	1,0	

From this you will get the following input data:

```
hbwstate, hbwcode, vgrstate, vgrcode, vgrtarget, mpostate, mpocode,
sldstate, sldcode, dsistate, dsicode, dsistate, dsocode
```

We have a total of 13 different states that we can take as input data. These are actually sufficient to create a first model. If they are not enough, we can add more topics to make our model more precise.

7.6 Data preparation second part

In the previous chapter we thought about what we can take as input data and determined these states. Now we have to create these pure states from the received MQTT messages and save them in a new file. With this we can finally create our model.

We accomplish this again with a Python script. This is located on kubernetes-master under the following path:
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CreateFileForNNTensorflow.ipynb
The function getIntToTarget() returns an integer value.

```
import json

# Multiple Topics Subscribe muessen zwingend eine Liste mit Tuples
sein
#topics = [("f/i/state/vgr", 0), ("f/i/state/sld", 0),
```

```
("f/i/state/mpo", 0), ("f/i/state/hbw", 0), ("f/i/state/dsi", 0),
("f/i/state/dso", 0)]
#targets = ["null", "mpo", "hbw", "dso"]
#          0       1     2       3

def getIntToTarget(target):
    if(len(target) == 0):
        return 0
    elif(target == "mpo"):
       return 1
    elif(target == "hbw"):
       return 2
    elif(target == "dso"):
        return 3
# end def
```

In the next section we define 13 variables. These are our input values. Then we define the cleaned and sorted file to be read and the new file to which the data will be written. First we write the column names into the new file.

```
try:
    hbwstate = -1
    hbwcode = -1

    vgrstate = -1
    vgrcode = -1
    vgrtarget = -1

    mpostate = -1
    mpocode = -1

    sldstate = -1
    sldcode = -1

    dsistate = -1
    dsicode = -1

    dsostate = -1
    dsocode = -1

    path = "/home/lernfabrik/ki-campus/modul_7/"
    data_file_in = path +
"data_file_big_raw_white_without_cam_new_ordered.json"
    f_in = open(data_file_in, "r")

    data_file_out = path + "data_file_big_raw_white_nn.csv"
    f_out = open(data_file_out, 'w+')

f_out.write('hbwstate,hbwcode,vgrstate,vgrcode,vgrtarget,mpostate,m
pocode,sldstate,sldcode,dsistate,dsicode,dsostate,dsocode\n')
```

We go through the individual topics of the file and let us always output all states at once in one line in the new file. However, only when one of the six topics comes up.

```
for line in f_in:
        json_value = json.loads(line)
        topic = json_value['topic']
        message = json_value['message']
        write_flag = False

        if (topic == "f/i/state/vgr"):
            vgrstate = 0 if message['active'] == Falseelse 1
            vgrcode = message['code']
            vgrtarget = getIntToTarget(str(message['target']))
            write_flag = True
        elif (topic == "f/i/state/sld"):
            sldstate = 0 if message['active'] == Falseelse 1
            sldcode = message['code']
            write_flag = True
        elif (topic == "f/i/state/mpo"):
            mpostate = 0 if message['active'] == False else 1
            mpocode = message['code']
            write_flag = True
        elif (topic == "f/i/state/hbw"):
            hbwstate = 0 if message['active'] == Falseelse 1
            hbwcode = message['code']
            write_flag = True
        elif (topic == "f/i/state/dsi"):
            dsistate = 0 if message['active'] == Falseelse 1
            dsicode = message['code']
            write_flag = True
        if (topic == "f/i/state/dso"):
            dsostate = 0 if message['active'] == Falseelse 1
            dsocode = message['code']
            write_flag = True

        if write_flag:
            line = str(hbwstate) + "," + str(hbwcode) + "," + \
                     str(vgrstate) + "," + str(vgrcode) + "," +
str(vgrtarget) + "," + \
                       str(mpostate) + "," + str(mpocode) + "," + \
                         str(sldstate) + "," + str(sldcode) + "," +
\
                            str(dsistate) + "," + str(dsicode) + ","
+ \
                              str(dsostate) + "," + str(dsocode) +
",\n"
            f_out.write(line)
            #print(message)
            #print(line)
    f_in.close()
    f_out.close()
except Exception as e:
```

```
    f_in.close()
    f_out.close()
    raise e
```

The result looks something like this. We see how the topics run in and gradually change from "-1" to the correct state. What should be immediately noticeable is that the entry "0,1,0,1,0,0,1,0,1,0,1," occurs very frequently. This seems to be the default state of the learning factory.

hbwstate,hbwcode,vgrstate,vgrcode,vgrtarget,mpostate,mpocode,sldstate,sldcode,dsi state,dsicode,dsostate,dsocode

-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,1,

-1,-1,-1,-1,-1,-1,0,1,-1,-1,0,1,

-1,-1,-1,-1,-1,-1,-1,0,1,0,1,0,1,

-1,-1,-1,-1,-1,0,1,0,1,0,1,0,1,

-1,-1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1,

0,1,0,1,0,0,1,0,1,0,1,0,1, ...

With this data, we can build our model. We first want to create a simple CNN (Convolutional Neural Network) model. This will be computed from the current states only.

7.7 Data preparation third part

That means we first try to find all possible states and if possible reduce them to a very short list so that we have a better overview of the data. For this we can take out all duplicate entries that are between the states themselves. We can do this manually, but again to avoid errors we write a simple Python script that does the work for us.

The script is located on kubernetes-master under the following path:

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/CreateFileForCNNTensorflow.ipynb

So we read the data back in and remove duplicate entries in the state transitions. The script is self-explanatory. It omits the first line with the columns. Then it always compares the current row with the previous one. If they differ it is written to a file. Finally the last status is also saved.

```
path = "/home/lernfabrik/ki-campus/modul_7/"
data_file = path + "data_file_big_raw_white_nn.csv"
f_in = open(data_file, "r")

data_file = path + "data_file_white_for_cnn.csv"
f_out = open(data_file, "w")

i=0
line_bevore = ""

for line in f_in:
    if(i == 0):
        i = i + 1
        continue
    if(i == 1):
        line_bevore = line

    if(line != line_bevore):
        print(i)
        print(line_bevore)
        f_out.write(line_bevore)
        line_bevore = line

    i = i + 1

print(line)
f_out.write(line)

f_in.close()
f_out.close()
```

Now, out of the 653 states, only 40 remain. Let's take a closer look at these numbers. We can delete the lines with "-1" manually. This is the initialization of the state variables, which is completed after the sixth station.

```
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,1,
-1,-1,-1,-1,-1,-1,-1,0,1,-1,-1,0,1,
-1,-1,-1,-1,-1,-1,-1,0,1,0,1,0,1,
-1,-1,-1,-1,-1,0,1,0,1,0,1,0,1,
-1,-1,0,1,0,0,1,0,1,0,1,0,1,
```

Now we have to consider which states the learning factory can assume and assign them to the respective line. We have to do this classification once otherwise the

model does not know which changes lead to which states. The first and the last one is simple. This is our basic state, we call it "Ruhend" (“Resting”). What is also easy to see is the output station in the middle where the last state changes from 1 to 0 in two places. The second line must be picking. This is because when an order is placed, the component must first be removed from storage. Using the targets, we can also define some states quite precisely. So, based on the code changes, we try to determine the states little by little.

The following is the finished data:

```
0,1,0,1,0,0,1,0,1,0,1,0,1,Ruhend
0,1,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
0,2,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,2,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,1,0,2,0,0,1,0,1,0,1,0,1,Auslagerung
1,1,0,2,0,0,2,0,1,0,1,0,1,Bearbeitung
0,2,0,2,0,0,2,0,1,0,1,0,1,Bearbeitung
0,2,1,2,1,0,2,0,1,0,1,0,1,TransportToMPO
0,2,1,2,1,1,2,0,1,0,1,0,1,TransportToMPO
0,2,0,2,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,2,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,1,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,0,2,0,1,0,1,0,1,Bearbeitung
0,1,0,1,0,0,2,1,2,0,1,0,1,Sortierung
0,1,0,1,0,0,1,1,2,0,1,0,1,Sortierung
0,1,0,2,0,0,1,1,2,0,1,0,1,Sortierung
0,1,0,2,0,0,1,1,1,0,1,0,1,Transport
0,1,0,2,0,0,1,0,1,0,1,0,1,Transport
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,0,1,0,0,AusgabeStation
0,1,1,2,3,0,1,0,1,0,1,1,0,AusgabeStation
0,1,1,2,3,0,1,0,1,0,1,0,1,TransportToDSO
0,1,1,2,3,0,1,0,1,1,0,0,1,TransportToDSO
0,1,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,0,2,0,0,1,0,1,1,0,0,1,Transport
0,2,1,2,2,0,1,0,1,1,0,0,1,TransportToHBW
0,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
0,1,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,1,2,2,0,1,0,1,0,1,0,1,TransportToHBW
1,2,0,2,0,0,1,0,1,0,1,0,1,Einlagerung
1,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
1,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,2,0,1,0,0,1,0,1,0,1,0,1,Einlagerung
0,1,0,1,0,0,1,0,1,0,1,0,1,Ruhend
```

7.8 Data Preparation fourth part

Now we have to check if there are states that could belong to more than one category. This is done again with a script. All states are read in. During the reading we check if this state is already assigned to a category. If not then it is written into the DataFrame. If the key already exists, an output is made. The script is located under jupyterhub-gpu at.

https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/create_file_for_rnn_all.ipynb

```
import numpy as np
import json
import pandas as pd

# Datei einlesen
data_file = "data_file_white_for_cnn.csv"
f_in = open(data_file, "r")

# DataFrame bilden
df_cat = pd.DataFrame(columns=['Kategorie', 'Key'])

# Kategorien durchgehen, doppelte auslassen
for line in f_in:
    cat = line[26:].strip()
    key = line[:26]

    flag = True
    for df in df_cat.values:
        if key == df[1]:
            print("Kategorie: " + cat + " vorhanden in " + df[0] +
" bei Zustand: " + key)
            flag = False
            break

    if flag:
        df_cat = df_cat.append({'Kategorie': cat, 'Key': key},
ignore_index=True)

f_in.close()
```

We see in the output that the storage in (Einlagerung) and storage out (Auslagerung) use identical states. Furthermore, there is a hit on storage out and transport.

```
Kategorie: Transport vorhanden in Auslagerung bei Zustand:
0,1,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: TransportToDSO vorhanden in TransportToDSO bei Zustand:
0,1,1,2,3,0,1,0,1,0,1,0,1,
```

```
Kategorie: Einlagerung vorhanden in Auslagerung bei Zustand:
1,2,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: Einlagerung vorhanden in Einlagerung bei Zustand:
1,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Einlagerung vorhanden in Einlagerung bei Zustand:
0,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ruhend vorhanden in Ruhend bei Zustand:
0,1,0,1,0,0,1,0,1,0,1,0,1,
```

We can combine the categories " Storage in" and "Storage out" into one category "Storage in/out". In case of storage out and transport, the combination does not make sense. Furthermore, we see that transportation occurs approximately in the middle of the order. It would not make sense to define the " Storage out " state in the middle of the order. So we change " Storage out" at the beginning to "Transport". We do this manually and save it in a new file.

/home/learning_factory/ki-campus/modul_7/data_file_white_for_cnn_new_kategory.csv

We run the script over the new file. The output shows that there are no more states assigned to more than one category.

```
Kategorie: Transport vorhanden in Transport bei Zustand:
0,1,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: TransportToDSO vorhanden in TransportToDSO bei Zustand:
0,1,1,2,3,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 1,2,0,2,0,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 1,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ein-/Auslagerung vorhanden in Ein-/Auslagerung bei
Zustand: 0,2,0,1,0,0,1,0,1,0,1,0,1,
Kategorie: Ruhend vorhanden in Ruhend bei Zustand:
0,1,0,1,0,0,1,0,1,0,1,0,1,
```

7.9 Data preparation the fifth part

As the last step in the data preparation we only have to insert the just defined categories into the original input file. For this we can take the previous script create_file_for_rnn_all.ipynb and extend it only by the part of the category writing.

```
ef getCatToKey(key):
    for df in df_cat.values:
        if key == df[1]:
            return df[0]

# Datei einlesen
data_file = "data_file_big_raw_white_nn.csv"
f_in = open(data_file, "r")

# Datei schreiben
data_file = "data_file_big_raw_white_nn_kategory.csv"
f_out = open(data_file, "w")

for line in f_in:
    new_line = line.strip() + str(getCatToKey(line.strip()))
    f_out.write(new_line + "\n")

f_in.close()
f_out.close()
```

The new data is now in:
/home/lernfabrik/ki-campus/modul_7/data_file_big_raw_white_nn_kategory.csv
Finally we can start to create the actual model.

7.10 CNN - Introduction

After the long data acquisition and data preparation, we finally come to the creation of the model. As a basis for this, we are guided by the example keras_iris_klassification from the book Deep Learning with TensorFlow, Keras and TensorFlow.js.
We will not go into the basic concepts of AI or neural networks here.
The completed model can be viewed on the kubernetes-master engine.
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/lernfabrik_model_mqtt_cnn_new_kategory.ipynb

7.11 CNN - Load and provide data

We load the data into a Numpy array. We see each of the individual values is automatically recognized.

```
# Laden der aufbereiteten Daten
stati_data = np.loadtxt("data_file_white_for_nn_kategory.csv", \
    delimiter=",", dtype="str", skiprows=0)
stati_data
```

```
array([['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ...,
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend'],
       ['0', '1', '0', ..., '0', '1', 'Ruhend']], dtype='<U16')
```

As we already know, Tensorflow doesn't handle different data types very well. The stats are strings so we can read them more easily. For our model, however, we need to convert them to integers.

For this we simply define an array with all our stats and replace the string values in the stati_data with integer values starting at 0. Finally we define that the whole array consists of integer values. The integers that are still stored as strings will be converted to integers.

```
# Wir erstellen die Kategorien:
stati_label_array = ["Auslagerung","TransportToMPO","Bearbeitung",
\
    "Sortierung", "AusgabeStation","Transport","TransportToDSO", \
    "TransportToHBW", "Einlagerung", "Ruhend"]
label_index = 0
for label in stati_label_array :
stati_data[np.where(stati_data[:,13]==label),13] = label_index
    label_index = label_index + 1

stati_data = stati_data.astype("int")
stati_data
```

```
array([[0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       ...,
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8],
       [0, 1, 0, ..., 0, 1, 8]])
```

Columns 1-13 are our states and therefore the input data. We create a new list in which we only extract these columns.

```
# Aufteilen der Daten in Input...
input_data = stati_data[:,0:13] # Spalten 0 bis 13 werden extrahier
input_data
```

```
array([[0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       ...,
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1],
       [0, 1, 0, ..., 1, 0, 1]])
```

The last column with the categories that we just turned into integer values is our output data. We create a new list and extract only the last column with the categories into it.

```
# ... und Output
output_data = stati_data[:,13]
output_data
```

```
array([8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
5, 5,5, 5, 5, 0, 0, 0...
```

We have nine categories defined, but only a simple list. Tensorflow cannot handle this. Tensorflow works with matrices. Therefore we have to transform the list. For this there is a function to_categorical(), this turns the simple list into a 1D array. We see it consists of 10 columns that are the categories and depending on which category it is it is set to 1 and all other columns are set to 0.

```
output_data = to_categorical(output_data)
output_data
```

```
array([[0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Now we have our input data as well as the output data. We now need to divide this into training data and evaluation data. In AI, a ratio of 4 to 1 has become common. That means 80% training data and 20% evaluation data. Keras already has a function for this. We give it our data as a transfer parameter, as well as the size of the training data and receive this divided back.

```
# Splitting des Datasets CNN
# Wir splitten in train und evaluation Daten
# 80% train und 20% Evaluation
stati_train_input, stati_test_input, \
stati_train_output, stati_test_output =
train_test_split(input_data, output_data, test_size=0.20)
```

7.12 CNN - Model Definition

Let's move on to the definition of our model. It is a sequential model. We will use "Dense" as the layer. The model has 13 input neurons, because 13 states. However, it is defined with 14. The last neuron is a slide neuron. This means that it always fires a random value. This is necessary so that a model does not over-specify. We add a hidden layer with 10 neurons in between and as output we also add 10 neurons, these are our output categories. But here we take "softmax" instead of "relu2" as activation function. Finally we output a summary of the model with the function summary(). With a total of 456 parameters, the model is one of the smaller ones.

```
# Aufbau des Modells mit Keras Convolutional CNN
stati_model = Sequential()
# Eingabeschicht 13 Neuronen + 1 Dias Neuron
stati_model.add(Dense(14,input_shape=(13,),activation="relu"))
stati_model.add(Dense(9,activation="relu"))
# Ausgabeschicht: 9 Zustaende
stati_model.add(Dense(9,activation="softmax"))
stati_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 14)                196

 dense_1 (Dense)             (None, 9)                 135
```

```
 dense_2 (Dense)             (None, 9)                 90

=================================================================
Total params: 421
Trainable params: 421
Non-trainable params: 0
_________________________________________________________________
```

The model is defined and we can now train it. With the compile function we configure our model. As "loss" function we take "categorical_crossentropy". We change the optimizer from SGD to the standard optimizer, the Adam, because we achieve better values with our model. Under "metrics" we indicate that we expect "categorical" as output, i.e. a unique result for one of the ten categories. In this case, "mae" means Mean Absolute Error.

```
# CNN
stati_model.compile(loss="categorical_crossentropy",
optimizer='adam', \
    metrics=["accuracy",tf.keras.metrics.mae]) # categorical
```

7.13 CNN - Model Training

With the function fit() we start the training of the model. We give the input data and the output data. As "batch-size" we take the common default1 value of 32. Under Epochs we choose 1000. Since we have a good GPU this should not be a problem. Also we want to have a closer look at the status output by doing this. With the parameter "verbose" we can output the status output during training.

We can re-train the model a few times, and will always get similar results. From the 100th epoch the accuracy is > 99%. The mean_absolute_error is <2%. From the 200th epoch the accuracy is 100% and the mean_absolute_error is 0%. From the 200th epoch on, no improvement of the model is possible. Better values cannot be achieved. This is probably due to the very precisely defined states that do not allow any room for multiple interpretations.

```
stati_model.fit(x=stati_train_input, y=stati_train_output,
batch_size=32, epochs=1000, verbose=1)
```

```
Epoch 1/1000
```

```
17/17 [==============================] - 1s 2ms/step - loss: 2.1947
- accuracy: 0.0494 - mean_absolute_error: 0.1961
Epoch 2/1000
17/17 [==============================] - 0s 2ms/step - loss: 2.0551
- accuracy: 0.1863 - mean_absolute_error: 0.1926
Epoch 3/1000
17/17 [==============================] - 0s 2ms/step - loss: 1.9491
- accuracy: 0.3365 - mean_absolute_error: 0.1891
. . .
Epoch 100/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0894
- accuracy: 0.9924 - mean_absolute_error: 0.0150
Epoch 101/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0872
- accuracy: 0.9924 - mean_absolute_error: 0.0146
Epoch 102/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0853
- accuracy: 0.9924 - mean_absolute_error: 0.0144
. . .
17/17 [==============================] - 0s 2ms/step - loss: 0.0108
- accuracy: 0.9981 - mean_absolute_error: 0.0022
Epoch 212/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 1.0000 - mean_absolute_error: 0.0022
Epoch 213/1000
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 1.0000 - mean_absolute_error: 0.0022
...
Epoch 998/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.3315e-06 - accuracy: 1.0000 - mean_absolute_error: 2.9760e-07
Epoch 999/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.3045e-06 - accuracy: 1.0000 - mean_absolute_error: 2.9208e-07
Epoch 1000/1000
17/17 [==============================] - 0s 2ms/step - loss:
1.2961e-06 - accuracy: 1.0000 - mean_absolute_error: 2.8876e-07
```

7.14 CNN - Model tests

We test our model with the test data and get what we have already seen in the intermediate output. We get an accuracy of 100% with a mean_absolute_error of 0%.

```
# Evaluation auf Test Daten
evaluation_results = stati_model.evaluate(stati_test_input,
stati_test_output)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 1.2138e-
06 - accuracy: 1.0000 - mean_absolute_error: 2.7068e-07
```

We want to run our own test and define some custom states for the test. For this we simply take some states from our input file and write them into a numpy array. With the function predict() we can output the predictions. And with the function argma()x from Numpy the highest value of these.

```
# Eigener Test
test_data = np.array([[0,1,0,1,0,0,1,1,2,0,1,0,1], # Sortierung
                      [0,1,1,2,3,0,1,0,1,0,1,0,1], # TransportToDSO
                      [1,2,0,2,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [0,2,0,1,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [0,1,1,2,3,0,1,0,1,0,1,0,0], # AusgabeStation
                      [0,1,0,1,0,0,1,0,1,0,1,0,1], # Ruhend
                      [0,2,0,2,0,0,1,0,1,0,1,0,1], # Ein-
/Auslagerung
                      [1,2,0,2,0,0,1,0,1,0,1,0,1] ]) # Ein-
/Auslagerung
predictions = stati_model.predict(test_data)
index_max_predictions = np.argmax(predictions,axis=1)
```

We iterate through the predictions and output the highest calculated category. Our own test confirms the very good result.

```
# Eigener Test Ausgabe
for i, e in enumerate(index_max_predictions):
    print("Datenreihe {} ergibt den Zustand: {}".format(
    test_data[i],
    stati_label_array[e]))
```

```
Datenreihe [0 1 0 1 0 0 1 1 2 0 1 0 1] ergibt den Zustand:
Sortierung
Datenreihe [0 1 1 2 3 0 1 0 1 0 1 0 1] ergibt den Zustand:
TransportToDSO
Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [0 2 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [0 1 1 2 3 0 1 0 1 0 1 0 0] ergibt den Zustand:
AusgabeStation
Datenreihe [0 1 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ruhend
Datenreihe [0 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-
/Auslagerung
```

Thus, we have successfully created the model for status detection. We save the model so that we can use it in another script.

```
# Modell speichern
stati_model.save("lernfabrik_model_cnn_saved.h5")
```

With that, we are done with this model. for completeness, we take a look at a RNN Model (Recurrent Neuronal Network)

7.15 RNN - Introduction

With a CNN, only the state matters. With an RNN, the previous states can be included in the prediction. This leads to a more accurate and faster result. However, we need to make a lot of adjustments to our model.
The definition of the categories is identical to that of the CNN model. The output data as well.

7.16 RNN - Data preparation

The model is stored under the following path.
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/lernfabrik_model_mqtt_rnn.ipynb
In RNN, however, we need to further adjust our input data. First, we determine how many states - including the current one - should be considered. That means, if we want to consider two previous states, we take two. The reorganization is done by the function arrange_data(). It goes through all entries of the given data and writes the defined number of previous values to the current state into a separate array. This is stored as a multidimensional array.

```
ZUSTAND_BEFORE = 2

# Reorganisiert die Daten fuer RNN
def arrange_data(data, days):
    days_before_values = [] # T - days
    days_values = []  # T
    for i in range(len(data) - days):
        days_before_values.append(data[i:(i+days)])
```

```
        days_values.append(data[i + days])
    return np.array(days_before_values),np.array(days_values)

days_before_values_input, days_values_input =
arrange_data(input_data,ZUSTAND_BEFORE)
days_before_values_output, days_values_output =
arrange_data(output_data,ZUSTAND_BEFORE)
```

In the sample output we see the array days_values_input which contains the actual states. It is a one-dimensional array.

```
print(len(days_values_input))
print(days_values_input)
```

```
656
[[0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 ...
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]
 [0 1 0 ... 1 0 1]]
```

And the multidimensional array days_before_values_input that always has two states for each entry, because we have chosen two states that should be included in the prediction.

```
print(len(days_before_values_input))
print(days_before_values_input)
```

```
656
[[[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]
...

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]

 [[0 1 0 ... 1 0 1]
  [0 1 0 ... 1 0 1]]]
```

Splitting of the data works the same.

```
# Splitting des Datasets RNN
X_train, X_test, Y_train, Y_test = \
train_test_split(days_before_values_input, days_values_output,
test_size=0.20)
```

7.17 RNN - Model Definition

In an RNN we have to use a different layer for the input. This is called LSTM (Long short-term Memory). Also, for input_shape we have to keep in mind that we now have a multidimensional array as input. For the probability output of our categories we stay with Dense. We output information about the model. We see that with the LSTM layer alone, our model parameters have almost quadrupled compared to CNN.

```
# Aufbau des Modells mit Keras RNN
stati_model = Sequential()
# Eingabeschicht 13 Neuronen + 1 Dias Neuron | + 2 vorherige
Zustände
stati_model.add(LSTM(14, return_sequences=False,
input_shape=(ZUSTAND_BEFORE, 13)))
# Ausgabeschicht: 9 Wahrscheinlichkeitswerte fuer jeden der 9
Zustaende
stati_model.add(Dense(9))
stati_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 14)                1568

 dense_1 (Dense)             (None, 9)                 135

=================================================================
Total params: 1,703
Trainable params: 1,703
Non-trainable params: 0
_________________________________________________________________
```

7.18 RNN - Model Training

In model training there is no difference to CNN. The result already looks very good after the 50th epoch. We let the model train a few times always with the same result accuracy at >98%. The mean_absolute_error however remains between 1-2%. In fact, for our model the CNN seems to fit better. As we have already noticed with the CNN, we are dealing with well-defined states and a simple CNN already finds the solution very quickly.

```
stati_model.fit(x=stati_train_input, y=stati_train_output,
batch_size=32, epochs=1000, verbose=1)
```

```
17/17 [==============================] - 2s 3ms/step - loss: 0.1484
- accuracy: 0.2691 - mean_absolute_error: 0.2450
Epoch 2/500
17/17 [==============================] - 0s 3ms/step - loss: 0.1115
- accuracy: 0.4580 - mean_absolute_error: 0.1932
Epoch 3/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0968
- accuracy: 0.4714 - mean_absolute_error: 0.1765
. . .
Epoch 50/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0107
- accuracy: 0.9656 - mean_absolute_error: 0.0512
Epoch 51/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0104
- accuracy: 0.9656 - mean_absolute_error: 0.0505
Epoch 52/500
17/17 [==============================] - 0s 2ms/step - loss: 0.0102
- accuracy: 0.9656 - mean_absolute_error: 0.0492
. . .
Epoch 498/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0039
- accuracy: 0.9828 - mean_absolute_error: 0.0169
Epoch 499/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0038
- accuracy: 0.9828 - mean_absolute_error: 0.0169
Epoch 500/500
17/17 [==============================] - 0s 3ms/step - loss: 0.0038
- accuracy: 0.9828 - mean_absolute_error: 0.0158
```

7.19 RNN – Model tests

The test data shows a good result.

```
evaluation_results = stati_model.evaluate(X_test, Y_test)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.0060 -
accuracy: 0.9697 - mean_absolute_error: 0.0168
```

Let's try our own data for testing. Here we have to be careful to pass a multidimensional array as well. We choose two consecutive states from our input file. The result should be the state after the two selected.

```
# Eigener Test
test_data = np.array([ [ [0,1,0,1,0,0,2,1,2,0,1,0,1],      #
(Sortierung) -2
                         [0,1,0,1,0,0,2,1,2,0,1,0,1], ],   #
(Sortierung) -1
                         #0,1,0,1,0,0,2,1,2,0,1,0,1,
Sortierung   0

                       [ [0,1,1,2,3,0,1,0,1,0,1,0,1],    #
(TransportToDSO) -2
                         [0,1,1,2,3,0,1,0,1,0,1,0,1], ], #
(TransportToDSO) -1
                         #0,1,1,2,3,0,1,0,1,0,1,0,1,
TransportToDSO   0

                       [ [0,2,0,2,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                         [0,2,0,2,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                         #0,2,0,2,0,0,1,0,1,0,1,0,1,        Ein-
/Auslagerung   0

                       [ [1,2,0,1,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                         [0,2,0,1,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                         #1,2,0,1,0,0,1,0,1,0,1,0,1,        Ein-
/Auslagerung   0

                       [ [0,1,1,2,3,0,1,0,1,0,1,0,0],    #
(AusgabeStation) -2
                         [0,1,1,2,3,0,1,0,1,0,1,1,0], ], #
(AusgabeStation) -1
                         #0,1,1,2,3,0,1,0,1,0,1,1,0,
AusgabeStation   0
```

```
                        [ [0,1,0,1,0,0,1,0,1,0,1,0,1],    # (Ruhend)
-2
                          [0,1,0,1,0,0,1,0,1,0,1,0,1], ], # (Ruhend)
-1
                          #0,1,0,1,0,0,1,0,1,0,1,0,1,
Ruhend   0

                        [ [0,2,0,1,0,0,1,0,1,0,1,0,1],    # (Ein-
/Auslagerung) -2
                          [0,2,0,1,0,0,1,0,1,0,1,0,1], ], # (Ein-
/Auslagerung) -1
                          #0,1,0,1,0,0,1,0,1,0,1,0,1,        Ruhend

                        [ [1,2,1,2,2,0,1,0,1,0,1,0,1],     #
(TransportToHBW) -2
                          [1,2,1,2,2,0,1,0,1,0,1,0,1], ]]) #
(TransportToHBW) -1
                          #1,2,0,2,0,0,1,0,1,0,1,0,1,         Ein-
/Auslagerung 0

predictions = stati_model.predict(test_data)
index_max_predictions = np.argmax(predictions,axis=1)
```

We see, that the RNN is really very good, but it has its Problems with the category transitions.

```
for i, e in enumerate(index_max_predictions):
    print("Datenreihe {} ergibt den Zustand: {}".format(
    test_data[i],
    stati_label_array[e]))
```

```
Datenreihe [[0 1 0 1 0 0 2 1 2 0 1 0 1]
 [0 1 0 1 0 0 2 1 2 0 1 0 1]] ergibt den Zustand: Sortierung
Datenreihe [[0 1 1 2 3 0 1 0 1 0 1 0 1]
 [0 1 1 2 3 0 1 0 1 0 1 0 1]] ergibt den Zustand: TransportToDSO
Datenreihe [[0 2 0 2 0 0 1 0 1 0 1 0 1]
 [0 2 0 2 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[1 2 0 1 0 0 1 0 1 0 1 0 1]
 [0 2 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[0 1 1 2 3 0 1 0 1 0 1 0 0]
 [0 1 1 2 3 0 1 0 1 0 1 1 0]] ergibt den Zustand: AusgabeStation
Datenreihe [[0 1 0 1 0 0 1 0 1 0 1 0 1]
 [0 1 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ruhend
Datenreihe [[0 2 0 1 0 0 1 0 1 0 1 0 1]
 [0 2 0 1 0 0 1 0 1 0 1 0 1]] ergibt den Zustand: Ein-/Auslagerung
Datenreihe [[1 2 1 2 2 0 1 0 1 0 1 0 1]
 [1 2 1 2 2 0 1 0 1 0 1 0 1]] ergibt den Zustand: TransportToHBW
```

Now that we have looked at both possibilities, we want to apply one of the models during an order process. This is part of the next chapter.

7.20 Model application in live mode

For the model application we need our saved model from chapter 3 or 4. Furthermore we need the source code from chapter 7.1.3. (data acquisition) and the source code from chapter 7.1.4. (data preparation). We need to combine these two scripts. For one, we need to receive the MQTT messages, convert them to the states that the neural network model can work with, and then make a prediction from them.
The finished script is on kubernetes-master at:
https://hivemqserver.feste-ip.net:9001/user/lernfabrik/notebooks/ki-campus/modul_7/Lernfabrik_MQTT_Stati_Modell_Load.ipynb
This sounds more complicated than it is. We just need to put the data preparation in the check_message_payload() function, as well as the prediction. The script for data retrieval is already running in an infinite loop. As soon as a new state message comes, the prediction is made and the state is output.

```
def check_message_payload(message):
    try:
        # Zeit tauschen zum vergleichen
        global hbwstate,hbwcode, \
             vgrstate,vgrcode,vgrtarget, \
             mpostate,mpocode, \
             sldstate,sldcode, \
             dsistate,dsicode, \
             dsostate,dsocode #, nfc

        if print_flag: print("\nNeue Message mit Topic: " +
str(message.topic))
        mess_dict = json.loads(message.payload.decode("utf-8"))

        if (message.topic == "f/"+s_i_let+"/state/vgr"):
            vgrstate = 0 if mess_dict['active'] == False else 1
            vgrcode = mess_dict['code']
            vgrtarget = getIntToTarget(str(mess_dict['target']))
        elif (message.topic == "f/"+s_i_let+"/state/sld"):
            sldstate = 0 if mess_dict['active'] == False else 1
            sldcode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/mpo"):
            mpostate = 0 if mess_dict['active'] == False else 1
            mpocode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/hbw"):
            hbwstate = 0 if mess_dict['active'] == False else 1
```

```
            hbwcode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/dsi"):
            dsistate = 0 if mess_dict['active'] == False else 1
            dsicode = mess_dict['code']
        elif (message.topic == "f/"+s_i_let+"/state/dso"):
            dsostate = 0 if mess_dict['active'] == False else 1
            dsocode = mess_dict['code']
        #elif (message.topic == "f/i/nfc/ds"):
            #nfc = 0 if mess_dict['workpiece'] == None else 1

        # Stati durch Modell und deren Wahrscheinlichkeiten
anzeigen lassen
        data = np.array([[int(hbwstate),int(hbwcode),

int(vgrstate),int(vgrcode),int(vgrtarget),
                         int(mpostate),int(mpocode),
                         int(sldstate),int(sldcode),
                         int(dsistate),int(dsicode),
                         int(dsostate),int(dsocode),]
                         #int(nfc),], # aktuellste Datenreihe
                         ])
        predictions = model.predict(data)
        index_max_predictions = np.argmax(predictions,axis=1)

        for i, e in enumerate(index_max_predictions):
            print("aktualisierte Datenreihe {} ergibt den Zustand:
{}".format(
            data[i],
            stati_label_array[e]))

    except Exception as e:
        if print_flag: print("Exception check_message_payload()")
        raise e
### end check_message_payload()<
```

When we start the script in the idle mode of the learning factory, only the "Idle" status is always displayed. We need to run an order to see a change in the output. It looks something like this:

```
. . .
Neue Message mit Topic: f/s/state/mpo
aktualisierte Datenreihe [1 2 1 2 2 0 1 0 1 0 1 0 1] ergibt den
Zustand: TransportToHBW

Neue Message mit Topic: f/s/state/hbw
aktualisierte Datenreihe [1 2 1 2 2 0 1 0 1 0 1 0 1] ergibt den
Zustand: TransportToHBW

Neue Message mit Topic: f/s/state/vgr
aktualisierte Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den
Zustand: Ein-/Auslagerung
```

```
Neue Message mit Topic: f/s/state/dsi
aktualisierte Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den
Zustand: Ein-/Auslagerung
. . .
```

The model is ready and works in live mode. We can now add more topics to further refine the categories. For example, the topic "f/i/nfc/ds" which manages the readout on the NFC chip.

7.21 Exercise

Apply the neural network model during an order.

APPENDIX B

SOLUTION EXERCISES STEP BY STEP

An appendix to the thesis submitted in partial
fulfillment of the requirement for the award of the
Bachelor's Degree of Mechanical Engineering with Honours

Faculty of Mechanical and Manufacturing Engineering
Universiti Tun Hussein Onn Malaysia

JULY 2023

APPENDIX B

Abstract

This document is an English explanation on how to solve the Learning Factory. Each of the exercises has a chapter where answers are provided step by step. Solutions are provided in the following format: On the left-hand side figures show illustrate the steps. On the right-hand side the reader can find instructions.

APPENDIX B

CONTENTS

CHAPTER 1	Glossary	155
CHAPTER 2	Exercise 1	156
CHAPTER 3	Exercise 2	158
CHAPTER 4	Exercise 3	160
CHAPTER 5	Exercise 4	161
CHAPTER 6	Exercise 5	163
CHAPTER 7	Exercise 7	167

APPENDIX B

CHAPTER 1

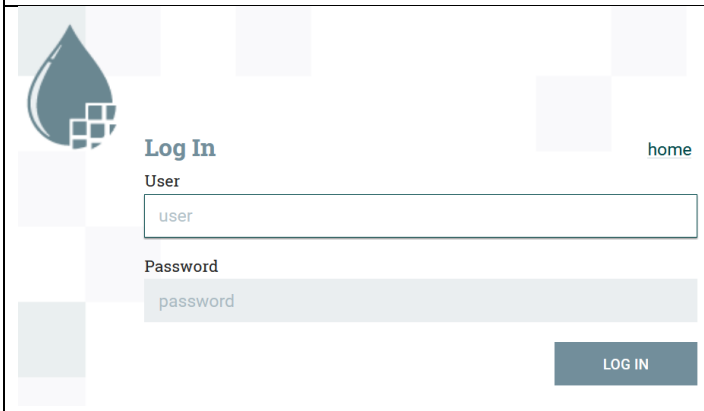
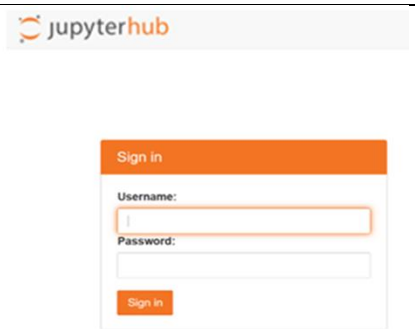
Glossary

German	**English**
Lern	Learn
Fabrik	Factory
Lernfabrik	Learning factory
Aktuell	Current
Künstliche Intelligenz / KI	Artificial Intelligence / AI
Modul	Module
Modell	Model

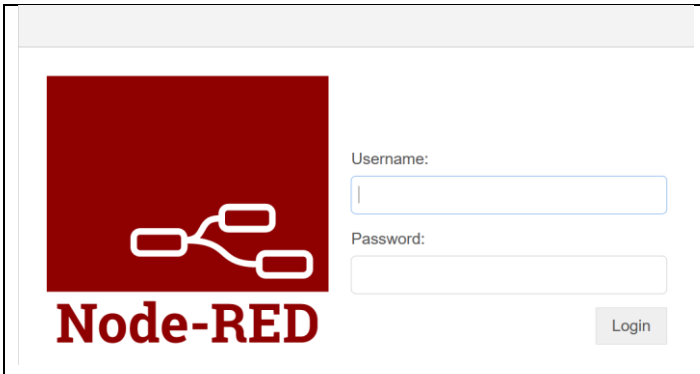
APPENDIX B

CHAPTER 2

Exercise 1

hivemqserver.feste-ip.net:9443/nifi/	Open link to nifi page
	Insert the following credentials: user: lernfabrik pw: L5nf1br|k Now you can take a look
hivemqserver.feste-ip.net:9001	Open the link to access Jupyter
	Insert the following credentials: user: lernfabrik pw: L5nf1br|k From here you can open new terminals and view the used notebooks
hivemqserver.feste-ip.net:1880	Open the link to access Node-RED online tool

APPENDIX B

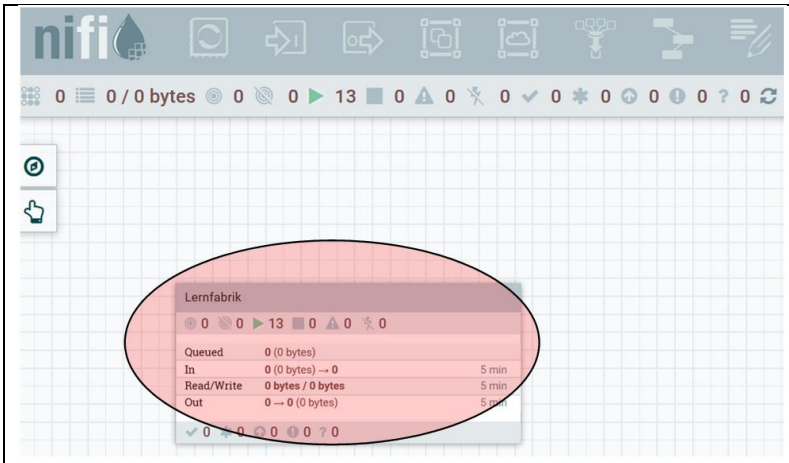
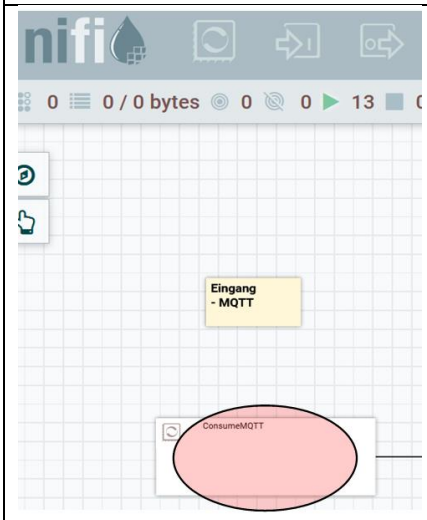
	Insert the following credentials: user: lernfabrik pw: L5nf1br|k This tool establishes the communication to the factory

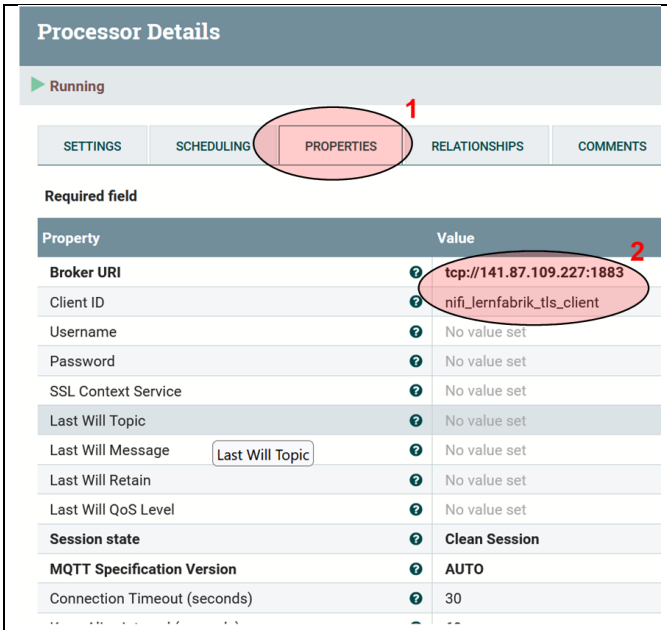
APPENDIX B

CHAPTER 3

Exercise 2

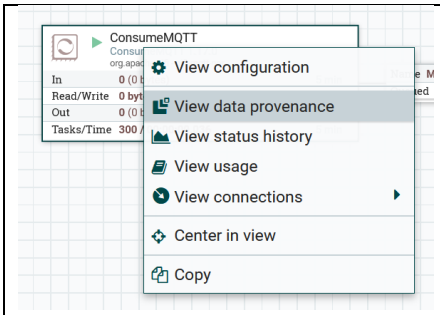
a)

	Double Click
	Click

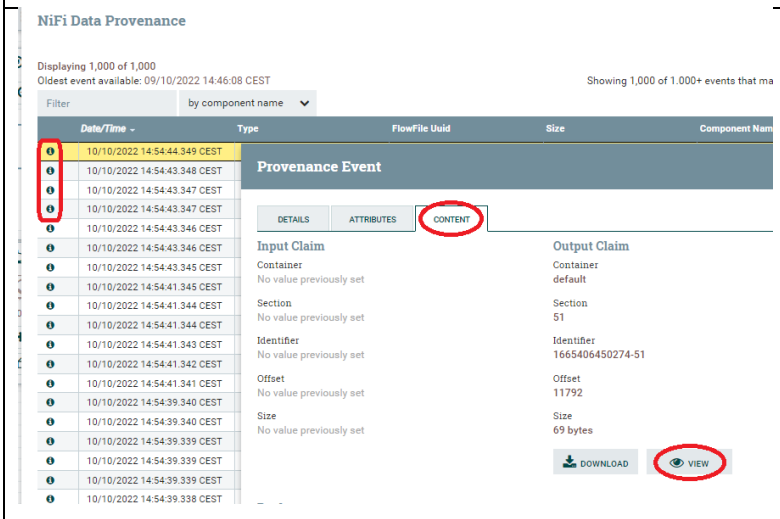


1) Click
2) Read

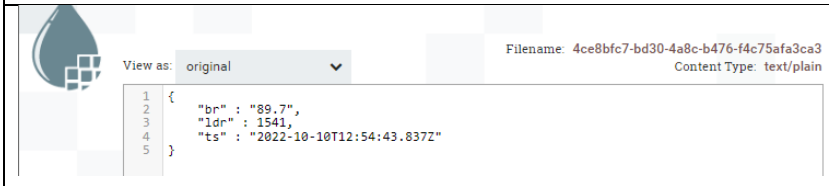
b)



1) Right Tap on Consume MQTT
2) Tap View data provenance



1) Click the info symbol of the message you are interested
2) Tap Content
3) Tap View

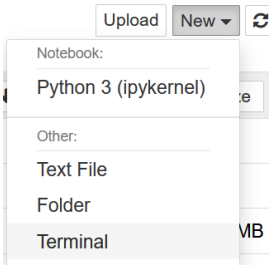


Complete MQTT-Message

CHAPTER 4

Exercise 3

a)

	1) Click new 2) Click terminal

jupyterhub
`lernfabrik@hivemqserver:~$ cd cli-client/`
`lernfabrik@hivemqserver:~/cli-client$ python3 lernfabrik.py -o BLUE`
	Type into terminal
	1) Go to directory
	2) Execute order

`Es wird ein Werkstück mit der Farbe "BLUE" bestellt.`
 	
`Connected to broker via TCP`
`Vorgang war erfolgreich!`
 	
A workpiece with color “BLUE” has been ordered.
Connected to broker via TCP.
Process was successful.
	Answer of terminal

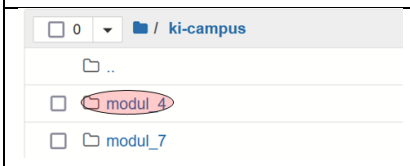
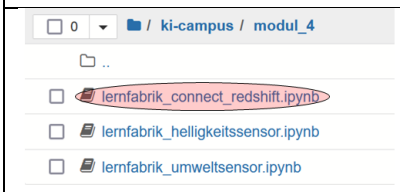
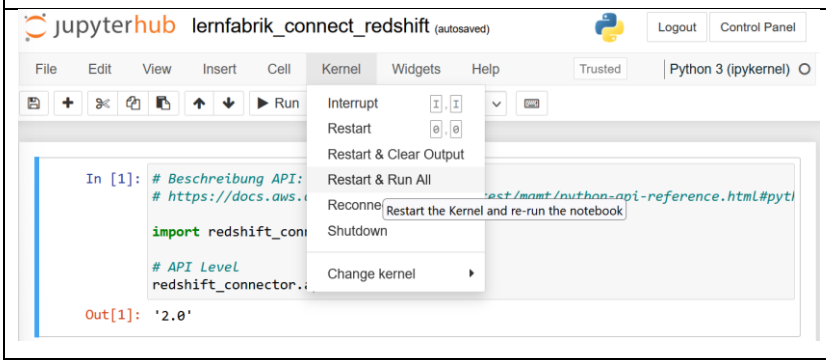
b)

`lernfabrik@hivemqserver:~/cli-client$ python3 lernfabrik.py -f`	Type
`Aktueller Fabrikstatus: ORDERED` Current factory state: ORDERED	Response of terminal

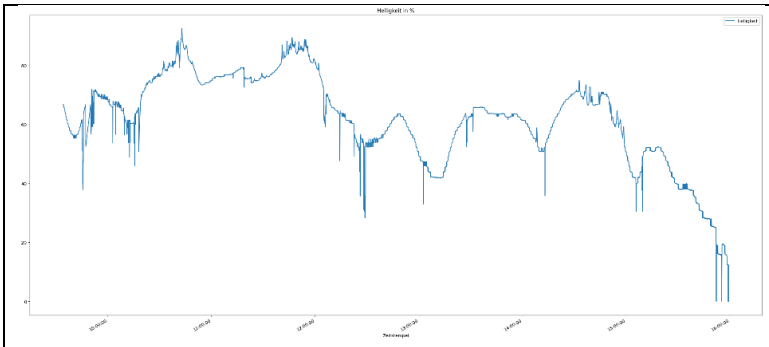
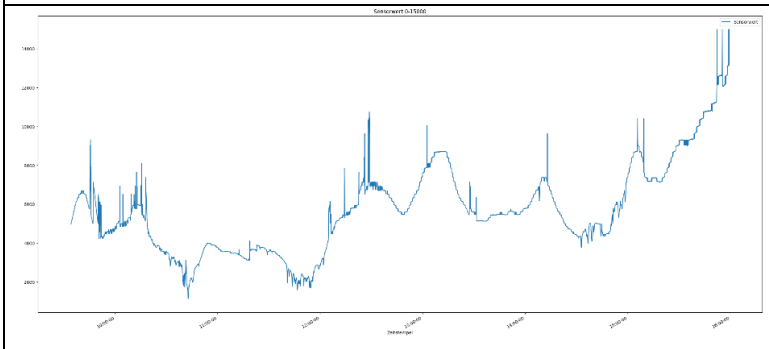
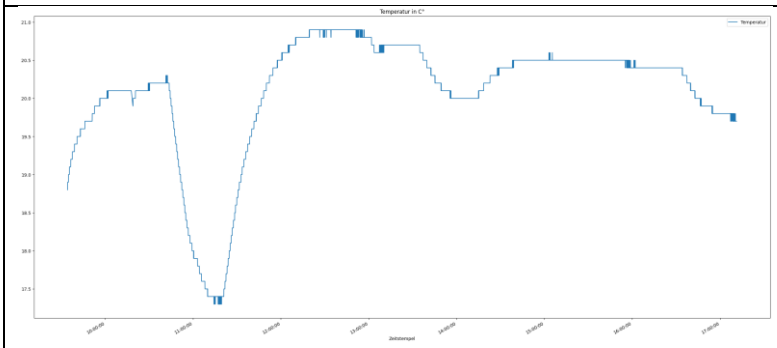
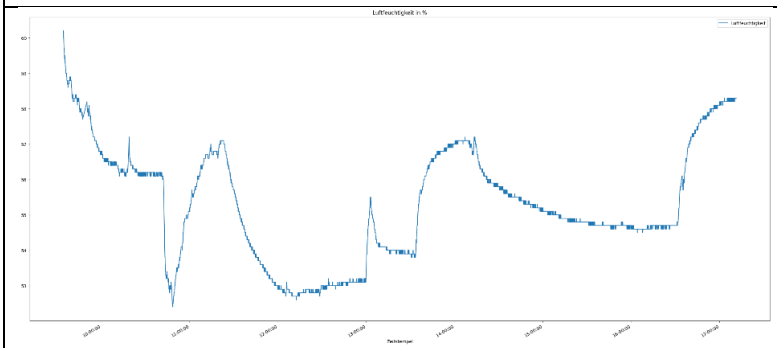
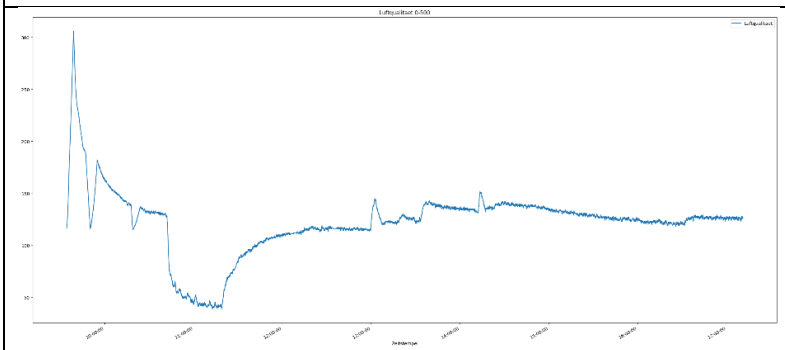
APPENDIX B

CHAPTER 5

Exercise 4

	1) Open JupyterHub 2) Double click ki-campus folder
	Double click modul_4 folder
	Open notebook by double click
	Execute the notebook by clicking Restart & Run All

APPENDIX B

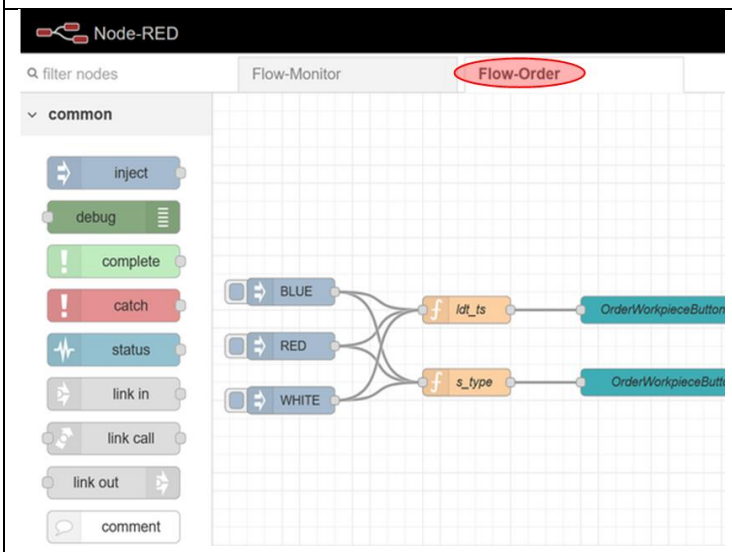
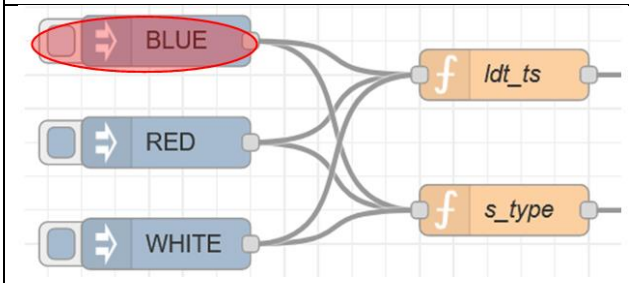
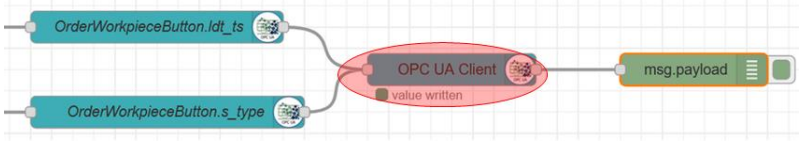
	Brightness in % is sinkings
	Brightness sensor value is increasing
	Temperature in °C is increasing in the beginning, then falling and increasing rapidly and changing slightly in the end
	Humidity is falling in the beginning, then rising and falling over the time
	Air quality value is decreasing in the beginning, then rising softly towards an asymptote

APPENDIX B

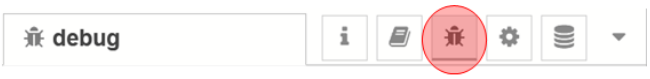
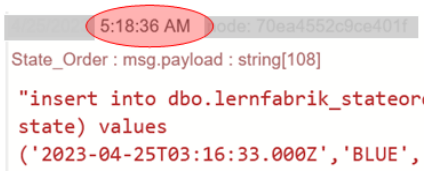
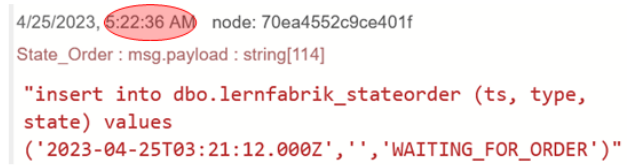
CHAPTER 6

Exercise 5

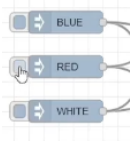
a)

Open Node RED, sign in	
	Click on Flow order to reach the flow, that is used to control the factory
	
	OPC UA Client node should show “value written”

b)

Trigger order like shown in a)	
	In the upper right corner, click on the bug to open the debug feed
	Wait for confirmation that a order has been placed Read the time the order has been placed: 5:18,36AM
	Wait during messages that show “IN_PROCESS” and “SHIPPED”
	Read the time in the message showing “WAITING_FOR_ORDER”: 5:22,36AM
5:22,36AM - 5:18,36AM = 4min	Calculate the duration

c)

-Trigger the same process like in b) -Goal of this task is to understand the messages issued in the debug window	
	Order triggered for red piece
5.7.2022, 18:32:00 node: 70ea4552c9ce401f State_Order : msg.payload : string[107] "insert into dbo.lernfabrik_stateorder (ts, type, state) values ('2022-07-05T16:31:56.000Z','RED','ORDERED')"	Confirmation that red piece is ordered
5.7.2022, 18:32:30 node: a1c7c907efb05359 Stock_HBW : msg.payload : string[555] "insert into dbo.lernfabrik_hbw (ts, a1_id, a1_state, a1_type, a2_id, a2_state, a2_type, a3_id, a3_state, a3_type, b1_id, b1_state, b1_type, b2_id, b2_state, b2_type, b3_id, b3_state, b3_type, c1_id, c1_state, c1_type, c2_id, c2_state, c2_type, c3_id, c3_state, c3_type) values ('2022-07-05T16:32:25.000Z','043d57a2186580','RAW','WHITE','','','','045b56a2186580','RAW','RED','045e55a2186580','RAW','RED','044057a2186580','RAW','WHITE','045259a2186580','RAW','BLUE','046c59a2186580','RAW','BLUE','046c57a2186580','RAW','WHITE','045159a2186580','RAW','BLUE')"	- This section shows the current stock - Every container has the following attributes: RFID, state if raw or already worked on and colour - Courser shows an empty space, the space is empty because of the placed order
5.7.2022, 18:33:00 node: 8ee6d72fada7f31a State_Stations : msg.payload : string[400] "insert into dbo.lernfabrik_stationen (ts, station, code, description, target, active) values ('2022-07-05T16:32:55.000Z','dsi',1,'','','false'), ('2022-07-05T16:32:55.000Z','dso',1,'','','false'), ('2022-07-05T16:32:55.000Z','hbw',2,'','','false'), ('2022-07-05T16:32:55.000Z','mpo',2,'','','false'), ('2022-07-05T16:32:55.000Z','sld',1,'','','false'), ('2022-07-05T16:32:55.000Z','vgr',2,'','mpo','true')"	- This section shows the states of each production step - The second column show the name of each station - The courser is pointing on the column indicating the state of each station 1: Ready 2: Currently working - The last column is indicating the station where the workpiece is at
5.7.2022, 18:35:00 node: 70ea4552c9ce401f State_Order : msg.payload : string[107] "insert into dbo.lernfabrik_stateorder (ts, type, state) values ('2022-07-05T16:34:42.000Z','RED','SHIPPED')"	Indicates that production is finished

APPENDIX B

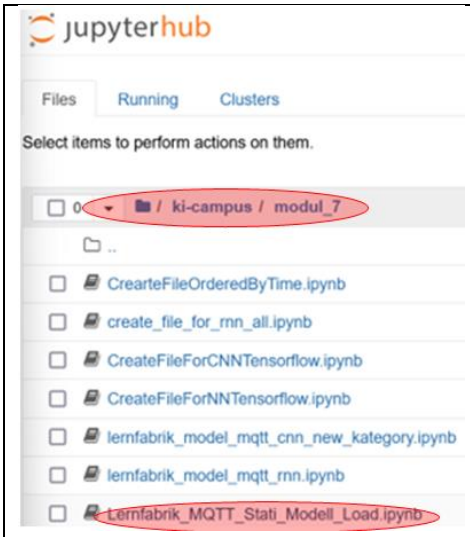
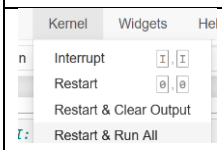
5.7.2022, 18:36:50 node: a1c7c907efb05359 Stock_HBW : msg.payload : string[575] "insert into dbo.lernfabrik_hbw (ts, a1_id, a1_state, a1_type, a2_id, a2_state, a2_type, a3_id, a3_state, a3_type, b1_id, b1_state, b1_type, b2_id, b2_state, b2_type, b3_id, b3_state, b3_type, c1_id, c1_state, c1_type, c2_id, c2_state, c2_type, c3_id, c3_state, c3_type) values ('2022-07-05T16:36:35.000Z','043d57a2186580','RAW','WHITE','04a056a2186580','RAW','RED','045b56a2186580','RAW','RED','045e55a2186580','RAW','RED','044057a2186580','RAW','WHITE','045259a2186580','RAW','BLUE','046c59a2186580','RAW','BLUE','046c57a2186580','RAW','WHITE','045159a2186580','RAW','BLUE')"	High rack storage has no empty slots anymore

Abbreviation table for stations

Delivery **S**tation **I**nput	**dsi**
Delivery **S**tation **O**utput	**dso**
High-**B**ay **W**arehouse	**hbw**
Multi **P**rocessing Station with **O**ven	**mpo**
Sorting **L**ine with Color **D**etection	**sld**
Vacuum **G**ripper **R**obot	**vgr**

CHAPTER 7

Exercise 7

	1) Navigate to modul_7 folder in jupyter 2) Open the marked notebook
	Execute the notebook by clicking Restart & Run All
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 58ms/step aktualisierte Datenreihe [0 1 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ruhend New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 58ms/step Actualized series of data [0 1 0 1 0 0 1 0 1 0 1 0 1] results in state: Neutral	- Scroll down to the end of the script until the message on the left - It says that the factory is currently resting

	Open a new terminal and navigate to cli-client folder
python3 lernfabrik.py -o BLUE	Insert the text on the left, this will trigger the order of a blue piece
Neue Message mit Topic: f/s/state/hbw current threads = 9 1/1 [==============================] - 0s 56ms/step aktualisierte Datenreihe [0 2 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-/Auslagerung New Message with topic: f/s/state/hbw current threads = 9 1/1 [==============================] - 0s 56ms/step Actualized series of data [0 2 0 1 0 0 1 0 1 0 1 0 1] results in state: to (take out of) warehouse	- Go back to the notebook - Wait for new messages to show up - "Ein-/Auslagerung" means that the production process has begun as a workpieces is taken out of the warehouse
Comment: A series of messages will pop up, as a data query is send periodically. It is possible that single messages show a wrong state which can be ignored.	
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 72ms/step aktualisierte Datenreihe [1 1 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Transport New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 72ms/step Actualized series of data [1 1 0 2 0 0 1 0 1 0 1 0 1] results in state: Transport	Next message will indicate that the gripper station is now running
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 76ms/step aktualisierte Datenreihe [0 2 0 2 0 0 2 0 1 0 1 0 1] ergibt den Zustand: Bearbeitung New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 76ms/step Actualized series of data [0 2 0 2 0 0 2 0 1 0 1 0 1] results in state: Workmanship	This message indicates that the workpiece is worked on
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 59ms/step aktualisierte Datenreihe [0 1 0 1 0 0 1 1 2 0 1 0 1] ergibt den Zustand: Sortierung New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 59ms/step Actualized series of data [0 1 0 1 0 0 1 1 2 0 1 0 1] results in state: Sorting	This message indicates that the colour sorting process is running
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 55ms/step aktualisierte Datenreihe [0 1 1 2 3 0 1 0 1 0 1 0 1] ergibt den Zustand: TransportToDSO New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 55ms/step Actualized series of data [0 1 1 2 3 0 1 0 1 0 1 0 1]	This message indicates that the workpiece is transferred to the output station

results in state: TransportToDSO	
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 67ms/step aktualisierte Datenreihe [0 2 1 2 2 0 1 0 1 1 0 0 1] ergibt den Zustand: TransportToHBW New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 67ms/step Actualized series of data [0 2 1 2 2 0 1 0 1 1 0 0 1] results in state: TransportToHBW	This message indicates that the workpiece is transported back to the warehouse
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 59ms/step aktualisierte Datenreihe [1 2 0 2 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ein-/Auslagerung New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 59ms/step Actualized series of data [1 2 0 2 0 0 1 0 1 0 1 0 1] results in state: to (take out of) warehouse	This message indicates that the piece is stored
Neue Message mit Topic: f/s/state/hbw 1/1 [==============================] - 0s 33ms/step aktualisierte Datenreihe [0 1 0 1 0 0 1 0 1 0 1 0 1] ergibt den Zustand: Ruhend New Message with topic: f/s/state/hbw 1/1 [==============================] - 0s 33ms/step Actualized series of data [0 1 0 1 0 0 1 0 1 0 1 0 1] results in state: Neutral	After the process has finished, the neutral message is shown again

APPENDIX B

REFERENCES

Itrich, A. & Klein, *M.. Lernfabrik 4.0 - Steuerung, Monitoring und NN-Modell (fischertechnik).* Albstadt-Sigmaringen. 2022