Semester Project Report - WS22/23

Workpiece Transfer Unit

First Author:	Henrik Meyer
Matriculation No.:	7012152
Second Author:	Peeranut Noonurak
Matriculation No.:	7023582

Course of Studies:	Master's in Industrial Informatics
First examiner:	Prof. Dr.-Ing. Armando Walter Colombo
Second examiner:	M.Eng. Jeffrey Werman
Submission date:	26.03.2023

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Department of Electrical Engineering and Computer Science
Master Degree in Industrial Informatics
Constantiaplatz 4 · 26723 Emden · `http://www.hs-emden-leer.de`

Abstracts

This project aims to design, develop, and validate a workpiece transfer unit for integration into a digital manufacturing environment. The developed system consists of hardware components, such as a robot stand, gripper, UR5e robot, test setup, base tool, and software components, including the robot program and OPC UA server/client, for subsystem integration and communication. The validation procedure involved testing the functionality and performance of the subsystems to ensure compliance with the project's specifications and safety requirements.

Despite the fact that certain validations could not be completed due to the ongoing construction of the digital factory, the workpiece transfer unit was designed with these specifications in mind. The unit demonstrated pick-and-place operations in a simulated environment, highlighting its potential as a valuable component in developing digital manufacturing. Future work will include integrating digitalization solutions to enhance the unit's capabilities within the Industry 4.0 landscape. The project provides the groundwork for future system improvements and refinements.

The latest version of this document and resources important to this project, such as program code and CAD files, can also be found on GitHub.

Contents

List of Figures

List of Tables

Acronyms

ERP Enterprise Resource Planning

FDM Fused Deposition Modeling

I40 Industry 4.0

IT Information Technology System

OPC UA Open Platform Communications Unified Architecture

OT Operation Technology System

PLA Polylactide

RAMI 4.0 Reference Architectural Model Industry 4.0

SDK Software Development Kit

UR Universal Robots

URP Universal Robots Program

1. Introduction

In the following chapter, the project is introduced, its objective is described and the structure of the following project documentation is explained.

1.1. Introduction and Motivation

The starting point for this project work is the conversion or reconstruction of the digital factory in the technical center of the University of Applied Sciences HS-Emden-Leer. At the time of writing, the reconstruction is still in the planning stage. The first steps have been taken, but a functioning factory does not yet exist.

The concept for the basic physical layout of the factory is shown in figure 1.1

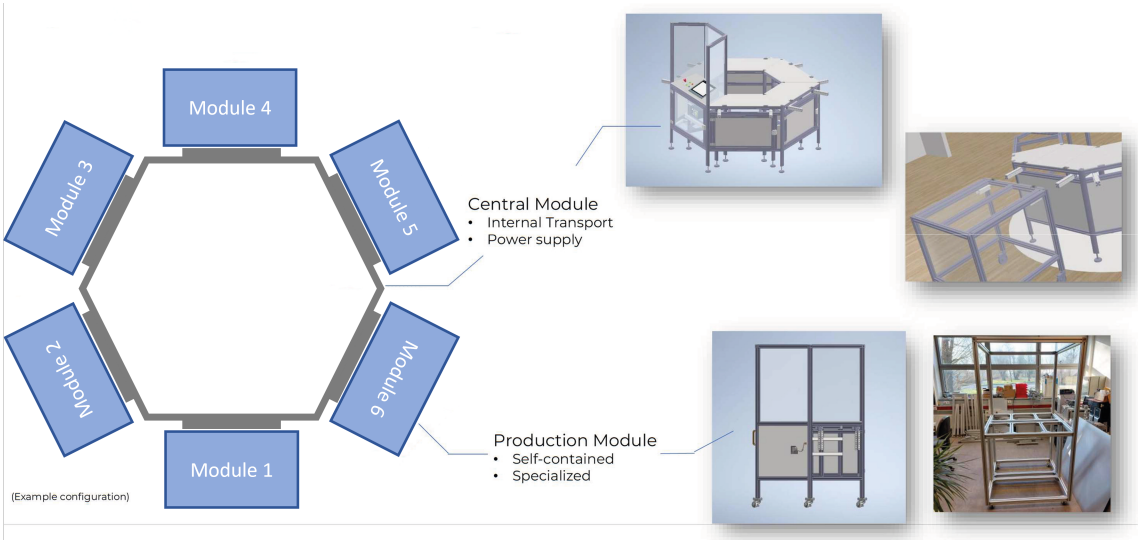


Figure 1.1.: Concept of digital factory at HS Emden Leer, based on [HS 22]

The idea of the concept is the provision of different modules, each covering a specific task. For example, a module has already been realised in which components can be cut and engraved with the help of a laser cutter. Other planned modules include assembly and quality control.

The frame of the factory is a hexagon to which the modules can be connected. The connection includes both the physical and the digital connection. Depending on the required processing steps, it should be possible to exchange the modules with each other or completely replace them with others. [HS 22]

One question that arises with this concept is how the workpieces to be processed are transported to the individual modules.

1.2. Aim and Structure of the work

This work aims to develop and implement a solution based on the given requirements that enable the transport of the given workpieces between the modules. In addition to the physical transport, great attention should be paid to the solution's ability to adapt to changing conditions by swapping modules and the communication via Industry 4.0-enabled technology. In order to accomplish this goal, several steps have to be taken.

The upcoming chapter introduces and explains concepts and technologies important to this project. Then the project management starts with a requirements analysis, a first specification based on this and the implementation planning. In the implementation, hardware and software solutions are designed and finally implemented. The partial solutions developed in each case are validated, and after completion, the finished product is evaluated based on the requirements established at the beginning. In the end, a summary and outlook are given.

2. Background

This chapter aims to introduce the project's three primary components: Universal Robot, OPC UA, and RAMI 4.0. For a complete comprehension of the project, it is necessary to have a solid understanding of these components, which serve as the basis of the project and must be comprehended in terms of their characteristics, capabilities, and applications. Hence, this chapter will provide an overview of all of these components, focusing on each component's significance and its role in the overall project.

2.1. Universal Robot

Universal Robots (UR) is a danish manufacturer of collaborative robots, also known as cobots, designed to work alongside human operators. All currently available robots feature an RRR design and six degrees of freedom. The UR5e is one of their products designed for industrial automation applications [Uni18a].

2.1.1. UR5e

The UR5e Robot is a collaborative industrial robot with three main components: the teach pendant, the physical robot, and the robot control, as shown in figure 2.1

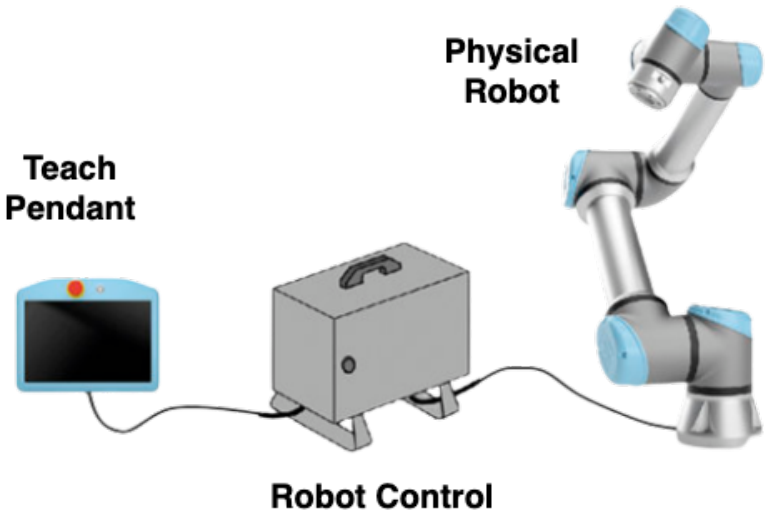


Figure 2.1.: Universal Robots - UR5e [Uni18a]

1. **Physical Robot**: The actual mechanical arm that carries out the tasks.

2. **Teach pendant**: Hand-held device that allows users to program, teach, and debug the Robot programs.

3. **Robot Control**: The computer system that runs the motion control algorithms, manages the Robot's I/O, and communicates with the Teach Pendant and other devices connected to the Robot.

In addition to the components of the robotic system listed here, in the vast majority of cases, an end effector is also required for the execution of a task, for example, a gripper. For easy commissioning and use of such an end effector or other systems, the software development platform URcaps exists.

2.1.2. URCap

URCap is a platform that includes a Software Development Kit (SDK) offering functionalities to create and publish custom software applications for collaborative robots manufactured by Universal Robots. The so-called "caps" are intended to enhance the functionality of these robots beyond the manufacturer's baseline capabilities, comparable to smartphone apps. Using URCaps, users can construct custom caps to fulfil unique application requirements, such as Gripper control and OPC UA communication [Uni21].

Gripper

URCap is frequently applied to gripper control. With URCap, developers are able to construct unique software for controlling the behavior and movements of robotic grippers. This can be especially beneficial in applications that require the robot to pick up and place objects in precise positions, requiring accurate gripper control without programming the necessary application itself [Uni21].

OPC UA Server/Client

OPC UA communication is one of the custom software types that can be used using URCaps. Open Platform Communications Unified Architecture (OPC UA) is an industrial communication protocol that facilitates secure and reliable data transmission between devices in a smart factory environment[Roc20]. By using an OPC UA URcap, developers can increase the robot's connectivity and interconnectivity in a smart manufacturing environment, aligning with the objectives of Industry 4.0. This allows the robot to communicate and exchange data with other equipment and systems within the facility. This can be especially beneficial for applications requiring monitoring and managing the robot's actions, such as variable pick-and-place or maintenance tasks.

2.1.3. Universal Robots Programming

There are two intended ways to create executable programs for Universal Robots. Either at the script level using URScript or PolyScope with a graphical user interface and higher level instructions [Uni21]. The end result is a robot program in the Universal Robots Program (URP) format, which the robot can execute. It should be noted that within the programming with PolyScope, scripts can also be created and executed, should existing program instructions not be sufficient. Figure 2.2 shows an example of programming in PolyScope using the teach pendant.

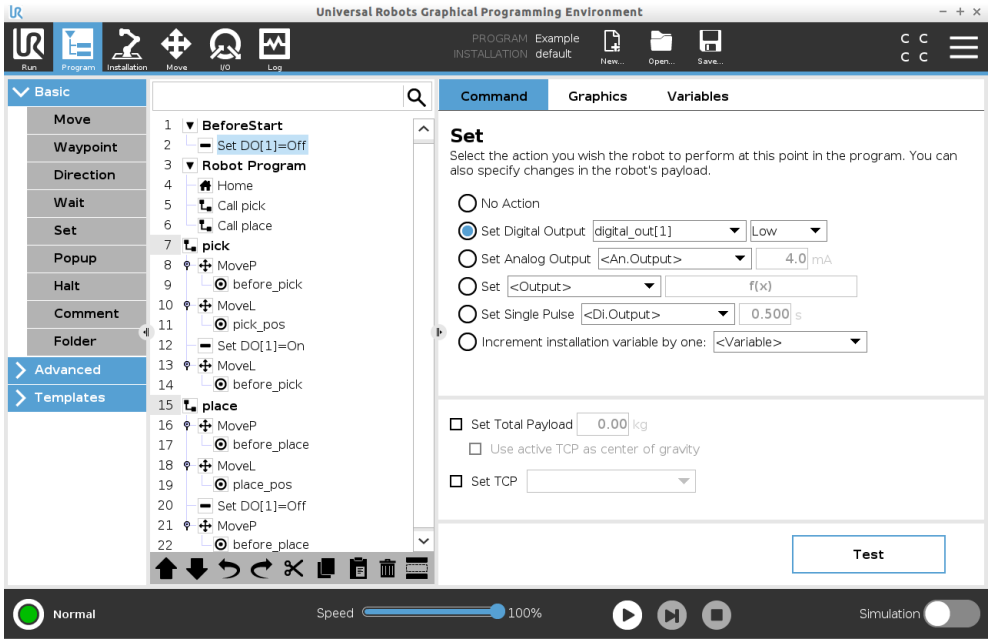


Figure 2.2.: Example of programming in PolyScope, own illustration

When programming with PolyScope, the program is built up of a series of commands carried out sequentially by the robot. The commands consist of predefined functions and parameters that can be combined to accomplish complex movements and operations. In addition, logic operations such as conditional statements and loops are supported, enabling the development of complex programs. The usage of sub-programs is advised, especially when dealing with bigger tasks. An example of the use of subprograms can be seen in Figure 2.2, with "pick" and "place". These subprograms can be utilized within the main program, improving the code's overall organization and clarity [Uni13].

2.2. OPC UA

Open Platform Communications Unified Architecture (OPC UA) is a standard communication protocol that enables secure and reliable data exchange between diversified industrial devices, applications, and platforms. The standard, which the OPC Foundation maintains, provides a standard communication interface for integrating industrial and non-industrial systems, including cloud-based systems [GHIU17] [MLD09].

OPC UA is an integral part of Industry 4.0. This fourth industrial revolution is characterized by increasingly using interconnected and intelligent devices exchanging information. OPC UA provides a secure and interoperable communication infrastructure for Industry 4.0 that enables the integration of information and Operation Technology System (OT), such as robots and automation systems, with Information Technology System (IT), such as Enterprise Resource Planning (ERP) and cloud-based systems [Col01].

opcua-asyncio

opcua-asyncio is an open-source OPC UA implementation for Python that extensively uses Python's async function. With over 2000 commits since December 2020 on GitHub, the implementation is an extensive and improving community project. Therefore, bugs in the implementation can not be ruled out but are relatively unlikely due to the high activity [opc23].

The library allows OPC UA client and server implementation in a few lines of code. It also supports importing XML files, allowing the combination with the AASX Package Explorer, simplifying the implementation of administration shells [aas23].

2.3. UaExpert

UaExpert is a software application developed by Unified Automation GmbH, which is widely used for testing and debugging applications that utilize the OPC UA protocol. It allows users to browse and interact with OPC UA servers, view and analyze the data they provide, and diagnose any issues that may arise.

The application offers a graphical user interface for connecting to OPC UA servers and navigating their address space, as shown in 2.3. It allows users to read and write variable values, view historical data, and monitor the server status. Additionally, it supports multiple security policies, such as authentication, encryption, and signing, ensuring that OPC UA servers and clients communicate securely.

Furthermore, UaExpert supports multiple transport protocols, including TCP and HTTPS, making it adaptable to various network environments. This allows developers, system integrators, and end-users interacting with OPC UA servers and clients to benefit from the software's interface and various features.

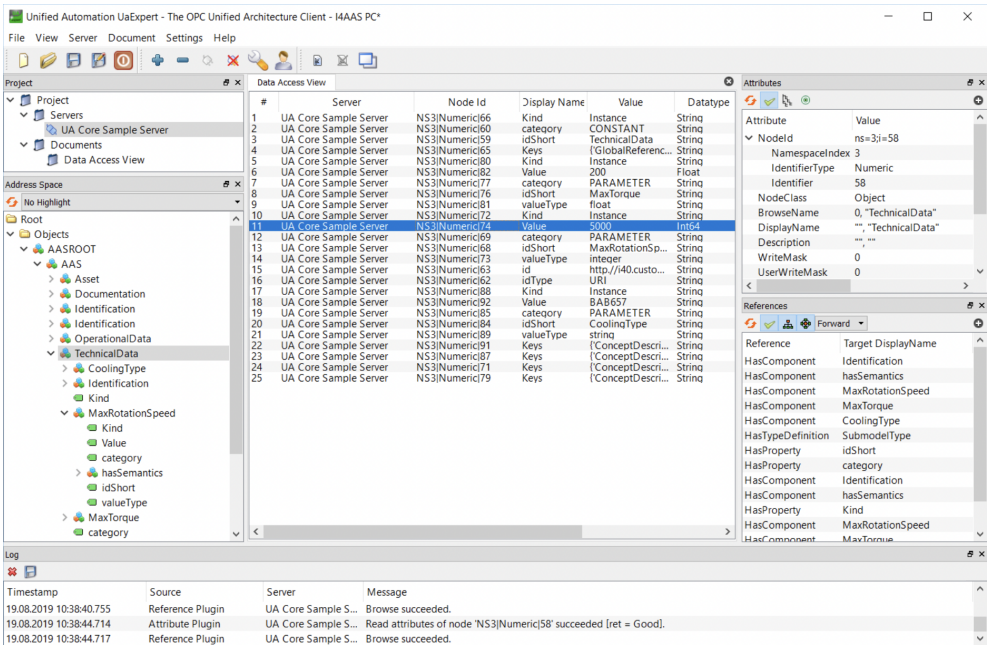


Figure 2.3.: Example of UaExpert Interface [Pla18]

2.4. RAMI 4.0

Reference Architectural Model Industry 4.0 (RAMI 4.0) is a reference architecture model that enables the development and integration of Industry 4.0 (I40). The model is standardized within DIN SPEC 91345. RAMI 4.0 has three dimensions that together map all the essential aspects of Industry 4.0.

Layers: The six layers help to divide the integration of a digital representation into logical units. They help answering the question of which aspects must be considered in each layer and how the respective layers are influenced.

Life Cycle Value Stream: The Life Cycle Value Stream structures systems along their life cycle and is based on IEC 62890. The axis is divided into type and instance, where the type is similar to a class in programming languages. It is like a blueprint for the instances, whose life typically starts with their production. The axis helps not to only focus on certain stages of the life cycle, like the usage phase of the instance, but to keep the whole life in mind. Also elements like the end of the lifecycle of type and instance are important overall.

Hierarchy Levels: The hierarchy levels are standardized according to IEC 62264/IEC 61512 and represent the levels of a company or its IT systems in a similar way to the ISA 95 pyramid. These have to be considered due to the different requirements of the levels. For example, because of their different cycle times, which can range from milliseconds in the lower levels to years in the corporate management level. In addition to the familiar corporate levels, the product level and the connected world have been added to complete the view.

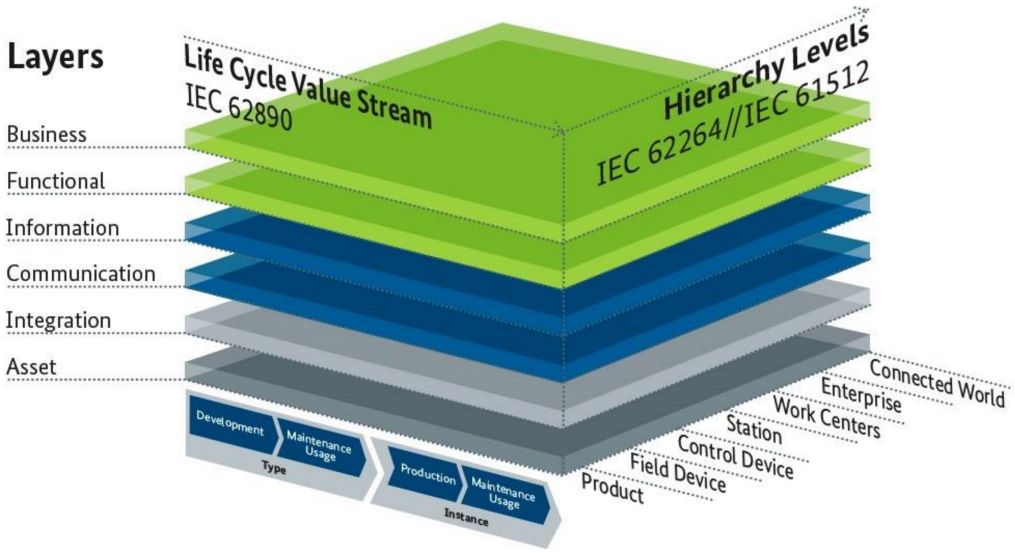


Figure 2.4.: RAMI 4.0 [Sch16]

RAMI 4.0 is an important basis for the development of Industry 4.0 systems, as it provides companies with a common architecture and uniform terminology. The model is independent of technology and manufacturer, so that companies can integrate their systems with systems from other manufacturers.

3. Project Management

This chapter outlines the project management process for developing a workpiece transfer unit and its corresponding communication capabilities. The project aims to develop a asset that satisfies specific requirements for handling workpieces and communication with other devices using Industry 4.0 standards.

The chapter begins with a requirements section, including functional, documentation, connectivity, and safety aspects. The requirements are classified by their respective type and explained by their significance.

Furthermore, the chapter provides a section for the specification, which elaborates a description of the workpiece transfer unit, including technology selection, extension tools, dimensions, workspace, transfer function, positioning accuracy, and capacity to handle workpieces of a particular weight and size. Also specified are the transfer unit's communication requirements. Then, a work plan is presented, detailing the tasks and milestones necessary to complete the project successfully.

3.1. Requirements

In order to develop a workpiece transfer unit that satisfies the demands of modern manufacturing environments, it is necessary to define and analyze the project's requirements thoroughly. This section describes the transfer unit's primary functional, documentation, connectivity, and safety requirements. By clarifying these, it is possible to ensure that the transfer unit is designed and implemented in such a way that all of these requirements are kept in mind during the process and are ultimately met, as well as possible.

As a result, the table 3.1 summarises the most important requirements for the technical solution. These requirements are the results of a task analysis and agreements with the client. The requirements are checked for their fulfillment at the end of the project and thus form a benchmark for its success.

ID	Priority	Requirement	Type	Description
Req.1	High	Transfer function	Functional	Ability to transfer workpiece from/to desired positions, while the amount of these positions is limited.
Req.2	Medium	Transfer variability	Functional	The positions between which the workpiece is to be transported can be determined by an unspecified system (e.g. Manufacturing Execution System).
Req.3	High	Asset's workspace	Functional	Ability to transfer the workpiece in a given workspace (A.1).
Req.4	Medium	Asset's dimension	Functional	The dimensions of the asset must not exceed a given space (A.2). This does not include temporary interventions in this area (e.g. robot arm) or necessary structures for supplying the modules.
Req.5	High	Asset's definition	Documentation	Clear definition of the asset, its components and features, like the workspace.
Req.6	High	Safety measures	Safety	The potential injury of persons is prevented as far as possible. No admissions are necessary.
Req.7	High	Structure of positioning area	Functional	The workpiece can be transfered from/to a flat surface and a conveyor belt (A.3).
Req.8	Medium	Positioning's accuracy	Functional	A positioning accuracy of less than 1.25 mm is maintained.
Req.9	High	Workpiece's dimension	Functional	Ability to handle a workpiece with certain dimensions (A.4).
Req.10	Medium	Workpiece's weight	Functional	Ability to handle a workpiece with a weight of up to 2 kg.
Req.11	Medium	Time for transfer function	Functional	A transport time of less than 30 seconds on average is maintained.
Req.12	Medium	Digital interaction between nodes	Connectivity	Possibility for other nodes to interact with the asset, to exchange information.
Req.13	Medium	Industry 4.0 protocol and interface	Connectivity	The used protocol and interface for connecting nodes should match Industry 4.0 requirements and be an industry standard. [Ind]

Table 3.1.: *Major Requirements*

3.2. Specification

In the specification section of this project management chapter, the principle hardware characteristics of the workpiece transfer unit and its communication capabilities are described. However, before the actual specifications are explained, the project is placed in front of the RAMI 4.0 model.

3.2.1. RAMI4.0 Specification

When considering the project in relation to its classification in the RAMI 4.0 model, two perspectives can be adopted. The perspective of the manufacturer/developer and the perspective of the user, the operator of the digital factory. Even if the current version of the project at hand is not directly an Industry 4.0 project, it is nevertheless worthwhile to already make a classification.

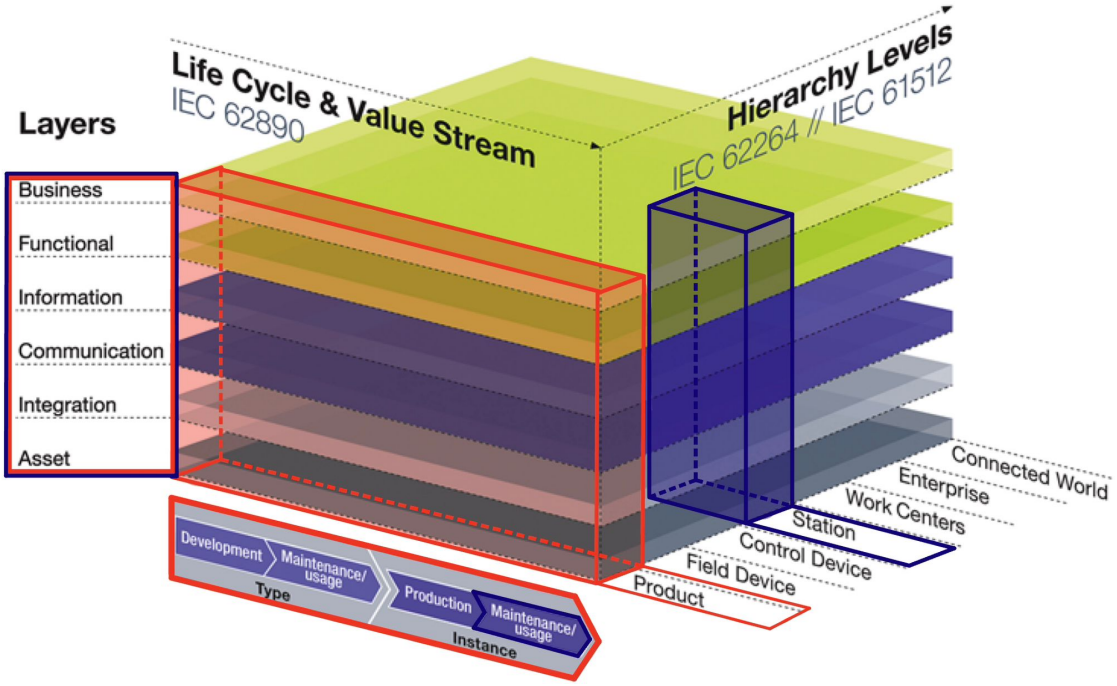


Figure 3.1.: RAMI 4.0 specification for the developers (red highlighted) and the user (blue highlighted), based on [Sch16]

Developer's view

As the developer of the workpiece transfer unit for the digital factory, the RAMI 4.0 model can give important hints to structure the project and pay attention to all important features and phases.

– **Layer axis**: The workpiece transfer unit covers all of the layers, since it affects all of these.

- **Life cycle axis**: The module can be placed in all phases from type (development and maintenance/usage) to instance (production and maintenance/usage). As the hardware and software are implemented, including the test and validation. The main part can be classified as development and production of type and instance, but the validation phase can also be seen as a small usage phase.

- **Hierarchy levels axis**: From the developer's point of view, the workpiece transfer unit is a product that is developed, implemented and tested.

User's view

As a component of the digital factory, the workpiece transfer unit is positioned within the RAMI 4.0 model.

- **Layer axis**: The workpiece transfer unit can be included in all layers, since it affects all of these. It can be seen in a top down approach, where for each of the layers the integration can be seen as a small part of the overall project. Starting with the integration of the system in the business layer.

- **Life cycle axis**: The workpiece transfer unit can be positioned at the instance level, which encompasses maintenance usage, as it is utilised and maintained throughout the digital factory's operation.

- **Hierarchy levels axis**: The workpiece transfer unit can be positioned beneath the station hierarchy level, as one of the digital factory's systems. A categorization as a work center could also be done, but since the asset is one component of the digital factory, consisting out of many of these, the positioning as a station is more reasonable.

3.2.2. Physical Transfer Technology

In the section on physical transfer technology, the physical characteristics of the workpiece transfer unit are chosen. The minimum required hardware components are laid out, and the general concept is introduced.

Robot

The decision to use a robot as the core of the transfer unit is based on the need for versatility and adaptability when manipulating the workpieces. The joint arm robot, specifically the UR5e, is selected due to its capacity to handle workpieces weighing up to 2 kg (5 kg), its positioning accuracy of less than 1.25 mm (0.1 mm), and its 6 degrees of freedom [Uni18a]. In addition to these performance data, a UR5e is already available at the digital factory, thus eliminating long waiting times and high costs when procuring an alternative.

Gripper

In terms of gripper design, a fixed gripper is chosen based on the requirements of the workpiece transfer unit. The fixed gripper will provide the strength and stability to transfer the workpieces effectively. In order to accommodate the size and shape of the workpieces, the gripper is designed based on these.

Workspace

The robot meets, without end effector, the minimum required workspace of 800 mm (radius) with its reach of 850 mm [Uni18b]. The final available workspace is determined based on the manufacturer's specification sheet and the data of the gripper. Since the robot without a gripper only serves the minimum necessary workspace, conveyor belts must be provided in the modules to be developed. These then enable transport to the intended position in the module. The robot workspace in its exact dimensions is defined later to ensure it is known to subsequent projects in the digital factory.

3.2.3. Connectivity Specification

In the section on the connectivity specification, the used communication technology is chosen. The minimum required software components are laid out, and the general concept is introduced.

OPC UA

OPC UA is the chosen communication technology for the workpiece transfer unit based on the project's requirements and Industry 4.0 standards. OPC UA is an open standard that enables secure and reliable data exchange between smart factory devices and systems. It provides a scalable and interoperable communication framework that enables the transfer unit to interact with other network nodes and devices. To ensure the confidentiality and integrity of the transferred data, OPC UA also provides enhanced security features, such as encryption and authentication.

Connecting the transfer unit to other nodes and devices requires both an OPC UA server and client. At least an OPC UA server is installed on the transfer unit, while the OPC UA client is installed on the nodes or devices requiring communication with the transfer unit.

For this project, the UR5e robot control component already includes OPC UA functionality (URcap), enabling wireless communication with the robot via the OPC UA protocol [Roc20]. This component is essential for transporting workpieces in this project since it allows the transfer of variables from the OPC UA client to the server and, in the end, to the URP program running in the robot. OPC UA also enables monitoring of the status and operating functions of the robot.

3.3. Work Plan

The V-model is a common systems development life cycle model that prioritizes testing and verification to ensure system quality. This model emphasizes the necessity of conducting testing activities concurrently with system development, with testing activities intensifying as the system approaches completion. Figure 3.2 depicts the project's work plan phases, which outline the schedule and tasks necessary to complete the project.

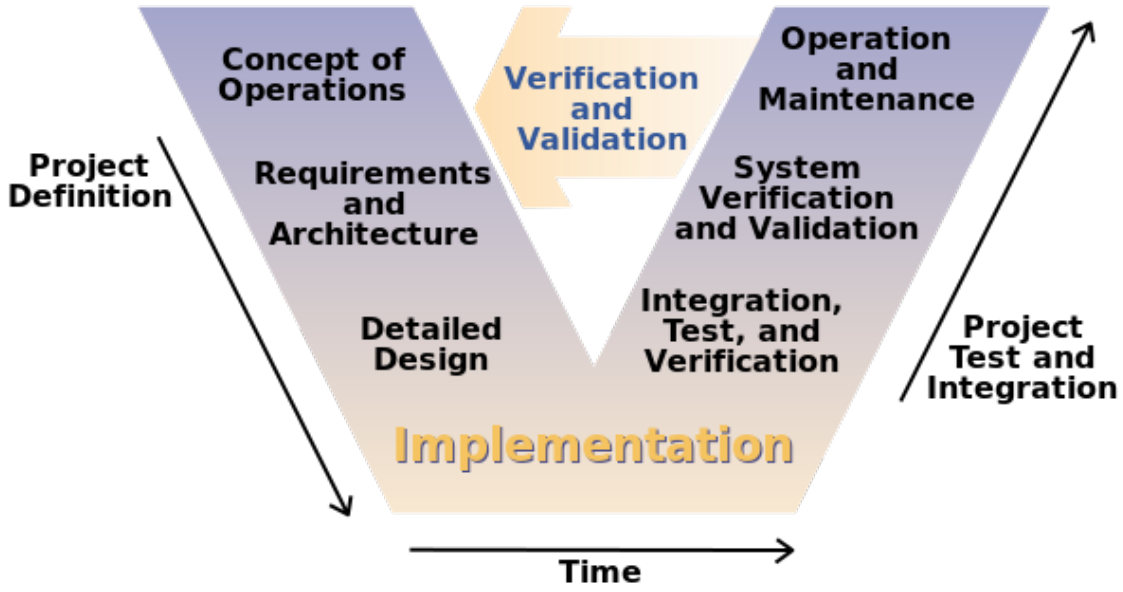


Figure 3.2.: V-model for project management[Col21]

By adopting the V-model approach, the workpiece transfer system project can benefit from a structured development process that ensures requirements are clearly defined and met and the system is thoroughly tested. It is deployed and maintained to align with user needs and expectations. The phases in the work plan for the workpiece transfer system project are as follows:

Phase 1: Requirements Analysis

The workpiece transfer system's requirements are defined and analyzed during this phase. This includes collecting and analyzing user requirements, developing a detailed requirements specification, and validating the requirements with stakeholders. Tasks in this phase include:

- Analyzing the requirements to identify any conflicts, ambiguities, or missing information.
- Prioritizing the requirements based on the importance of the project and the user.

Phase 2: Detailed Design

The requirements defined in Phase 1 are translated into a system design during this phase. This design includes the workpiece transfer system's software and hardware architecture. Included in this phase are:

- Developing a detailed system architecture design that includes the OPC UA protocol-based communication architecture and the system's various components.
- Creating designs for each of the system's components, including the OPC UA Client & Server (PC), and the OPC UA server (robot control).

Phase 3: Implementation

During this phase, the system design is achieved through the development of software and hardware components of the workpiece transfer system. Included in this phase are:

- Designing the required hardware components, manufacturing them and then assembling all components.
- Realizing a hardware and software environment for the implementation and subsequent validation of the system.
- Implementing the robot program.
- Developing the software code for the OPC UA server & client (PC), and the OPC UA server (Robot Control).
- Configuring the OPC UA server (PC) to receive variables from clients and pass them to the OPC UA server (robot control) via the OPC UA client (PC).
- Configuring the OPC UA server (Robot Control) to receive variables from the OPC UA client (PC) and pass them to the robot program in the robot control.

Phase 4: Integration, Test, and Verification

During this phase, the workpiece transfer system is tested and verified to ensure that it meets the project requirements and performance specifications. Tasks in this phase include:

- Conducting unit tests to verify the functionality of individual system components and validate the subsystems, such as the OPC UA server/client, robot arm, maintenance/service function, and gripper.
- Conducting integration tests to verify the functionality of the integrated system and ensure that the various components of the workpiece transfer system operate together correctly. This includes validating the connectivity requirements of the workpiece transfer unit.

Phase 5: System Verification and Validation

During this phase, user acceptance testing is conducted to ensure that the system meets user requirements. Tasks in this phase include:

- Verify the system requirements and performance. This includes validating the functional requirements, documentation requirements, connectivity requirements, and safety requirements of the workpiece transfer unit.
- Validating the workpiece transfer unit against the original requirements specification, including the functional requirements, documentation requirements, connectivity requirements, and safety requirements, as well as any additional requirements identified during the development process.

Phase 6: Deployment and Maintenance

During this phase, the workpiece transfer system will be deployed to the production environment and maintained to ensure it continues to operate correctly and meet user requirements. However, since the digital factory is currently under construction, parts of this phase will be postponed until the completion of the factory. Once the digital factory is ready, all of the following tasks will be carried out:

- Handing over the system to the production team, who will take responsibility for its ongoing operation and maintenance.
- Conducting maintenance and support activities to ensure the system continues to operate correctly and meet user requirements. This includes tasks such as monitoring system performance, conducting regular maintenance tasks, and addressing any issues that arise.

Overall, the V-model provides a structured approach to the development of the workpiece transfer system, ensuring that requirements are clearly defined and met, the system is thoroughly tested, and it is deployed and maintained in a way that meets user needs and expectations.

4. Solution Design

The solution architecture combines both hardware and software components. The hardware design entails selecting and integrating functional system components, such as the UR5e robot, robot stand, gripper, and base tool. The software architecture of the workpiece transfer system is centered on the OPC UA protocol, which facilitates communication between system components, such as the asset and other customers.

4.1. Asset

The so-called asset comprises all hardware components and software developed during this work that contribute to achieving the project goal. More precisely, the asset's physical part consists of the robot, robot control, teach pendant, gripper, base tool, robot stand, and, strictly speaking, auxiliary components such as screws used. In addition, software elements such as the robot program, the OPC UA server in the robot control, and the OPC UA server and client in the server/PC are included.

It does not include elements used only for the development of the asset, such as dummy pick and place positions (e.g., tables) or software that is only needed for testing and validating the function (e.g., UaExpert client).

4.2. Hardware Design

The physical part of the workpiece transfer unit consists of multiple components, each of which serves a specific purpose in fulfilling the task. These components are the UR5e robot, the robot control, the teach pendant, the robot stand, the gripper, and the base tool. Figure 4.1 illustrates the conceptual physical structure of the workpiece transfer unit, excluding components where positioning is not critical, such as the robot control.

Although additional supporting components such as screws and washers are present, they are not covered in this section. The following subsubsections will describe each main hardware component of the asset and its respective role.

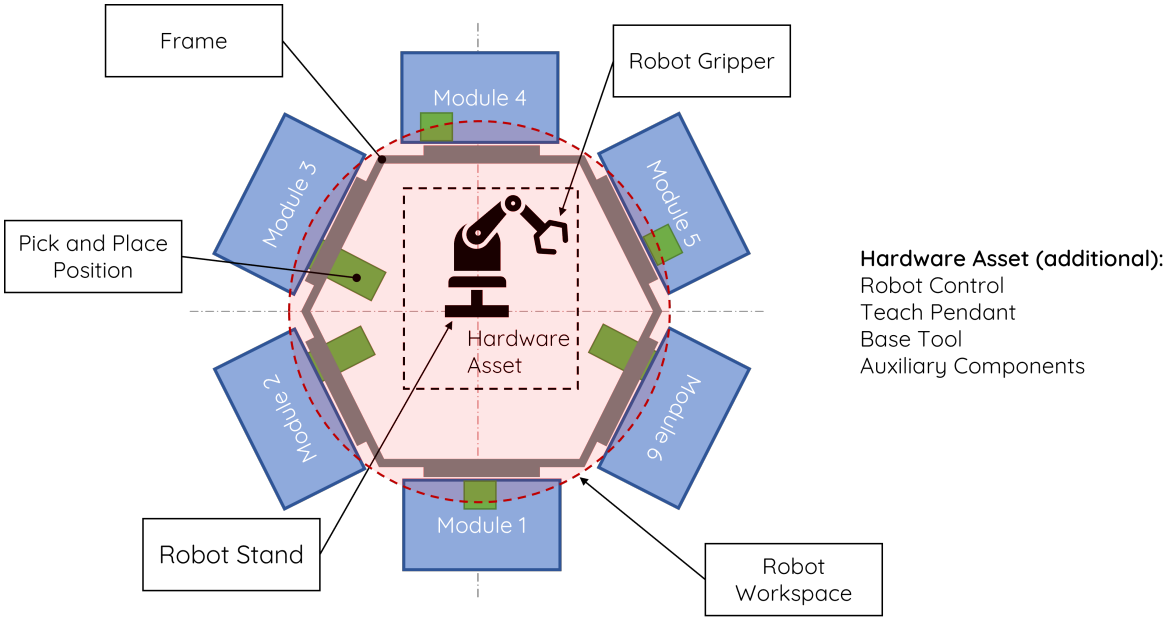


Figure 4.1.: Design of the physical structure, dimensions not to scale, own illustration

UR5e Robot (Robot, Robot Control and Teach Pendant)

The previously selected robot, the UR5e, is a collaborative 6-axis robot. It is an almost universally applicable robot whose task is determined by the end effector it guides. Possible tasks include palletizing, glue application, or pick-and-place operations. The robot itself consists of various components, more specifically the robot control, the robot arm and the teach pendant [Uni13] [Uni18b].

The UR5e has various safety features to ensure safe operation in industrial environments. These safety features include collision detection, which prevents the robot from hard collisions with objects in its workspace, and an emergency stop button, which can be used to immediately halt the robot's movements in the event of an emergency. Due to these properties, further safety measures, such as a protective fence, are unnecessary.

In addition to its physical capabilities, the UR5e robot also possesses OPC UA-based communication capabilities as a URcaps add-on application. The robot control component's OPC UA function enables wireless communication with the robot via the OPC UA protocol. This enables the transfer of variables from an OPC UA Client to the URP language program running in the robot control. This function enables indirect control and monitoring of the robot program. [Roc20]

The robot will perform pick-and-place or service operations (e.g., maintenance position) based on instructions from the OPC UA Client that is wirelessly connected to the robot. The URP language program is taught via the teach pendant to specify its movements and several pick-and-place positions. In the robot program, bases/planes features are used, which can be configured using the base tool. As a result, the features mentioned earlier allow the implementation in a way that all requirements can be met.

Robot Stand

The robot stand fulfils the task of fixing the robot in its position and bringing it to an optimal height in relation to the modules. Since the workspace of the robot is almost a sphere, it can be better utilised. The important thing when implementing this is therefore a suitable height and sufficient mechanical stability so that the accuracy of the robot is affected as little as possible.

Gripper

The gripper enables the robot to perform the task at hand. The gripper must be adapted to the geometry of the workpiece. Its geometry and material must be adapted to the mechanical requirements so that the accuracy during positioning is not significantly affected. In addition, requirements such as the ability to place the workpiece on flat surfaces and fast closing and opening must be considered. The availability of energy sources must also be taken into account. Compressed air or hydraulics, for example, have significant disadvantages and are not readily available in the system under consideration.

Based on these requirements, an electromechanical operating principle is selected. Due to the availability and suitability, a Robotiq Hand-E gripper is used as the basis. The standard fingers are replaced by a design adapted to the task. The gripper type is a parallel motion two-jaw gripper.

Base Tool

In order to offer the possibility of docking modules at any point of the six sides of the system, as shown in Figure 4.1, the use of bases/planes is necessary. These can be used in the robot program to reuse the same program for each of these six sides. So it is unnecessary to teach the program for a module several times. The programming effort for commissioning a new module is thus greatly reduced. Setting these bases is impossible with the gripper or not with acceptable repeat accuracy. Therefore, another tool is needed to set these bases.

The design of this tool requires a simple and positionally safe mounting on the robot and a tip with which the reference points for the bases can be set safely. The base tool, as it is called here, is therefore designed to meet the above requirements.

4.3. Software Design

The software architecture of the workpiece transfer unit is designed around the OPC UA protocol, which facilitates communication between various system components, including the asset and other clients.

Hence, using OPC UA enables the system to achieve interoperability between different machines and devices within a manufacturing environment or the digital factory, achieving the project objective. The communication architecture is depicted in Figure 4.2, which illustrates the overall system design.

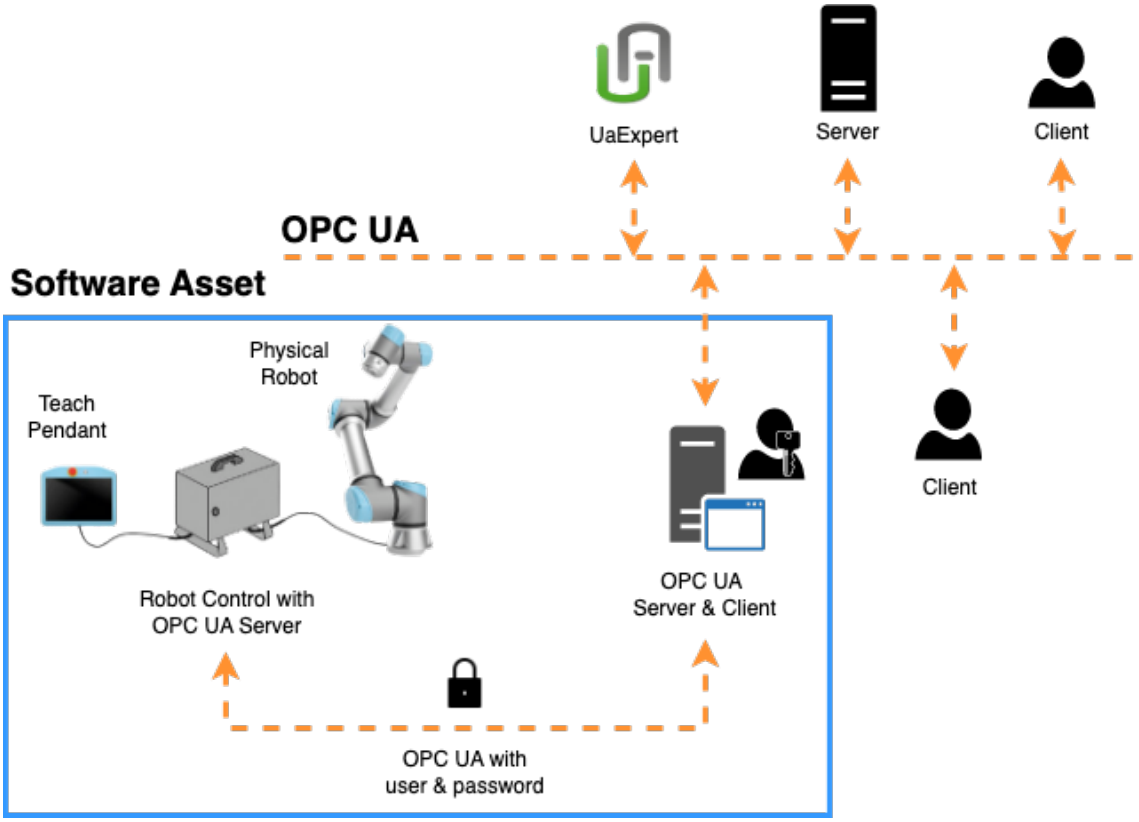


Figure 4.2.: Overall Communication Architecture, own illustration

Client

The clients are an integral part of the system. They can access the asset through the interface provided by UaExpert in order to interact with the workpiece transfer unit and provide variable operating instructions. UaExpert is introduced in Chapter 2.3 and is used in this project primarily as a system for simulating possible clients. The interface is a gateway to the OPC UA Server (PC), an intermediary between the user and the OPC UA Server (robot control). The Client can send variable instructions for the robot's movements and operations through this interface. While UaExpert is

one approach to interacting with the asset as a client via an intuitive interface, there are other ways. Instead of a human user interacting with it, in production, mainly other assets like the modules of the factory or an MES System will communicate with this asset.

Software Asset

The asset in the workpiece transfer system comprises three primary components:

1. **OPC UA Server (PC):**
   This server is responsible for receiving variables from clients or users, which can be other modules in the digital factory or external devices. The server then passes these variables to the OPC UA Client (PC).

2. **OPC UA Client (PC):**
   This client communicates with the OPC UA Server (robot control) via a Python program using opcua-asyncio library. Functions provided by the client (PC) are used by the server (PC) to send variables to the OPC UA server (robot control).

3. **OPC UA Server (Robot Control):**
   This server is responsible for receiving variables and transferring them to the robot control's URP. The URP then processes the variables in order to execute the required operations.

A security layer is used to ensure secure communication between the OPC UA Client (PC) and the OPC UA Server (robot control). This layer restricts access to the workpiece transfer system and enables only authorized users to send variable robot commands. In order to achieve this, a username and password approach is adopted. These credentials are declared directly in the OPC UA Server (Robot control), and the same username and password are provided in the OPC UA Client (PC). This approach ensures a certain level of security, so only authorized users can access the OPC UA server of the robot control.

5. Implementation

The implementation part explains the methods used to create and integrate the system components into a working collective. This part describes the software and hardware components used to build the workpiece transfer unit and how they are combined to provide a solid system.

It should be noted in the implementation that the digital factory still needs to be put in place. For example, the hexagonal frame to which the modules are to be docked still needs to be created in reality. This applies to the physical part as well as to the software part. Due to the circumstances described, the implementation uses auxiliary constructions that simulate the later system as well as possible and thus prove the asset's functionality.

5.1. Hardware Implementation

This chapter provides a detailed account of the physical components that are designed, manufactured and assembled to launch the hardware asset.

Robot Stand

The robot's stand fulfills the task of fixing the robot in its position and bringing it to a suitable height. It is positioned precisely in the middle of the future plant. The modules the robot supplies with workpieces in the finished factory have a working height of about one meter. Most of the modules are expected to have a conveyor belt for transport into the module and out of the module itself. These conveyor belts can be positioned higher or lower than the working area of the modules. Because of the lack of further information and the fact that no complete module is available yes (one module is functional but lacks the conveyor belt), the described height of one meter is chosen as the reference. The final platform should therefore be designed so that the robot's workspace in a plane at the height of one meter fills a maximum area, i.e., is as large as possible.

Since the modules are not yet available, the exact height of the pick and place positions is unknown and the he digital factory is not finally positioned in the room; an existing platform is used as a base for the time being. At 1.05 m, this is higher than planned but does not need to be anchored to the floor due to its very stable and heavy construction. In order to be able to connect the robot to this platform in the following steps, an adapter plate is designed and manufactured out of carbon steel

(S235JR). The technical drawing can be seen in B.1. It uses the existing hole pattern of the platform to be mounted on and has threaded holes for mounting the UR5e.

So that the adapter plate shown does not have to be disposed of when a more optimal robot stand is used later, it already has holes drilled in it to allow further use. Further use is intended in conjunction with the construction shown in Figure 5.1.

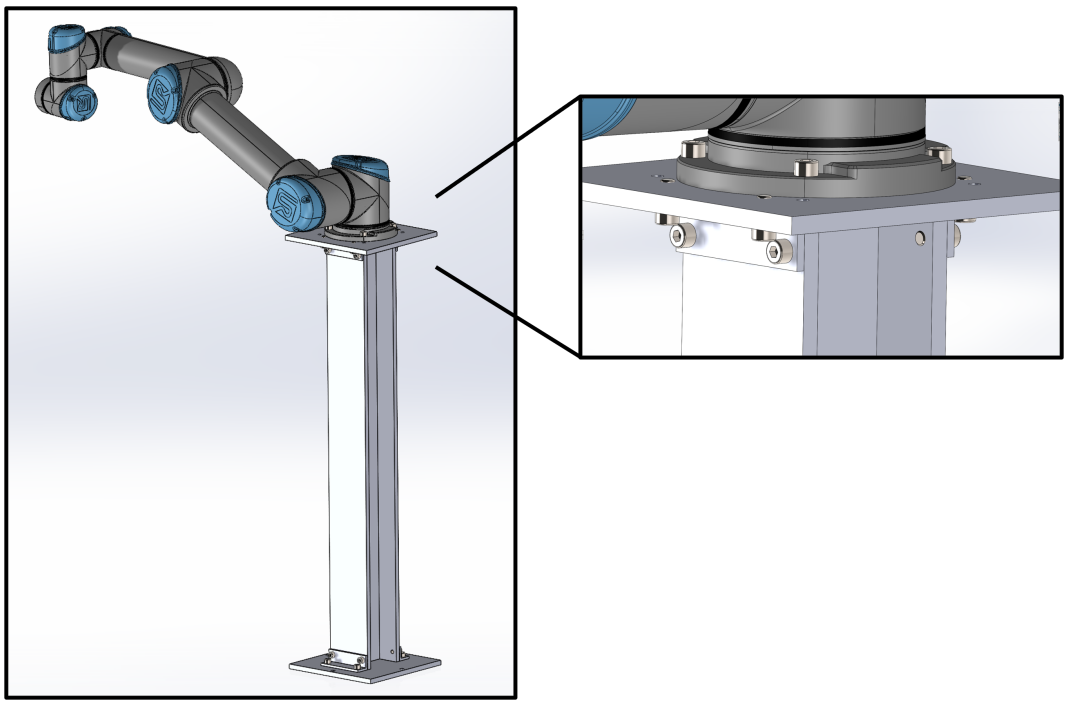


Figure 5.1.: Proposed robot stand for later use, own illustration

The robot stand shown has a height adjusted to one-meter module height, but the middle part can be exchanged to be able to change the height relatively easily in case of modified conditions. When specifying an elevation of one meter, it should be noted that this refers to the center of the robot's workspace, shown later in this documentation, not the robot's base. The platform shown in Figure 5.1 raises the center of the workspace to precisely one meter. In addition, the adapter plate has holes for mounting a UR10 in case a larger working area or payload is needed.

Gripper

The gripper/end effector performs the actual task. It is moved in space by the robot and holds the workpiece. The gripper has to adapt to the workpiece and other conditions like the area where the workpiece is picked up from and placed to. The functional principle used here is that the gripper slides under an edge of the workpiece in a horizontal motion to lift the workpiece in a vertical motion. It should be noted that some additional component is placed on top of the workpiece. Gripping from above is therefore avoided here so that there is as much clearance as possible for a

geometry placed on top. Another circumstance to consider is that according to the requirements, the workpiece can lie on a flat plane and a conveyor belt, as shown in Figure 5.2.

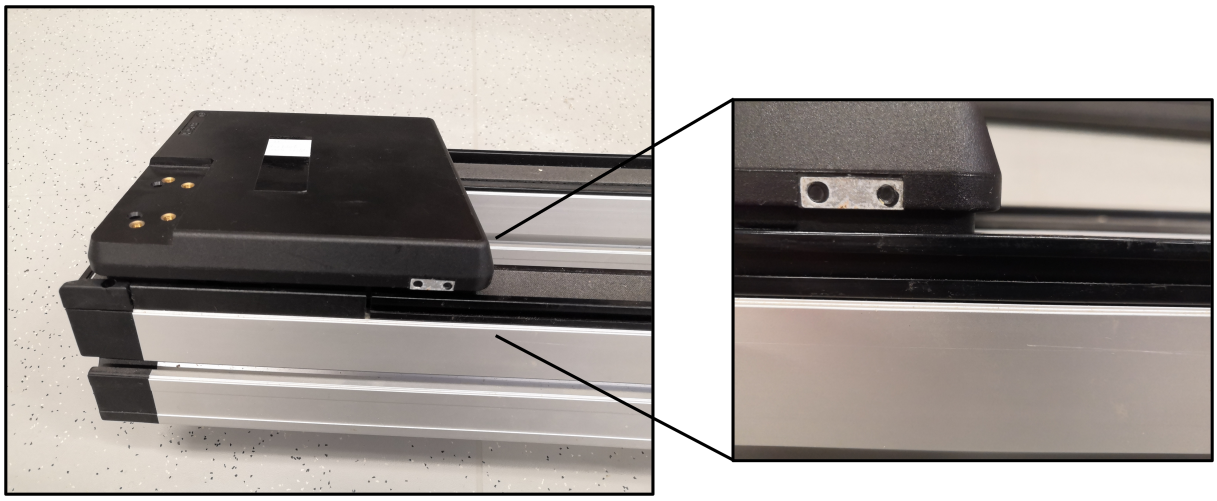


Figure 5.2.: Conveyor belt with workpiece in place, own illustration

The usable, free edge length is reduced to 3.5 mm if it lies on the conveyor belt. This must be taken into account when designing the gripper. The fingers must fit into this gap and still leave enough play for deviations in the robot's guidance.

In order to be able to lift the workpiece from a flat surface, the gripper is designed to be guided over a certain distance above the flat plane. For this purpose, the fingers drop off to one side and thus lie below the plane defined by the basic body of the gripper. The design of the gripper concerning the workpiece can be seen in Figure 5.3. The displacement of the TCP is 200.5 mm in the Z-direction and -41.25 mm in the Y-direction. The TCP for the gripper is thus in the center of the lower surface of the workpiece. These values can be used later to adjust the gripper TCP in the robot configuration.

The base element of the gripper is a Robotiq Hand-E parallel motion gripper. The fingers are attached with screws. [Rob18] Fused Deposition Modeling (FDM) is initially used as a low-cost manufacturing process for producing the fingers out of Polylactide (PLA). In a later step, however, they should be milled from aluminum, for example, to achieve the required mechanical stiffness.

The parameters for the printing process using the FDM method are configured in Ultimaker Cura. The printing itself is carried out with the help of an Anycubic I3 Mega S. The material used is PLA.

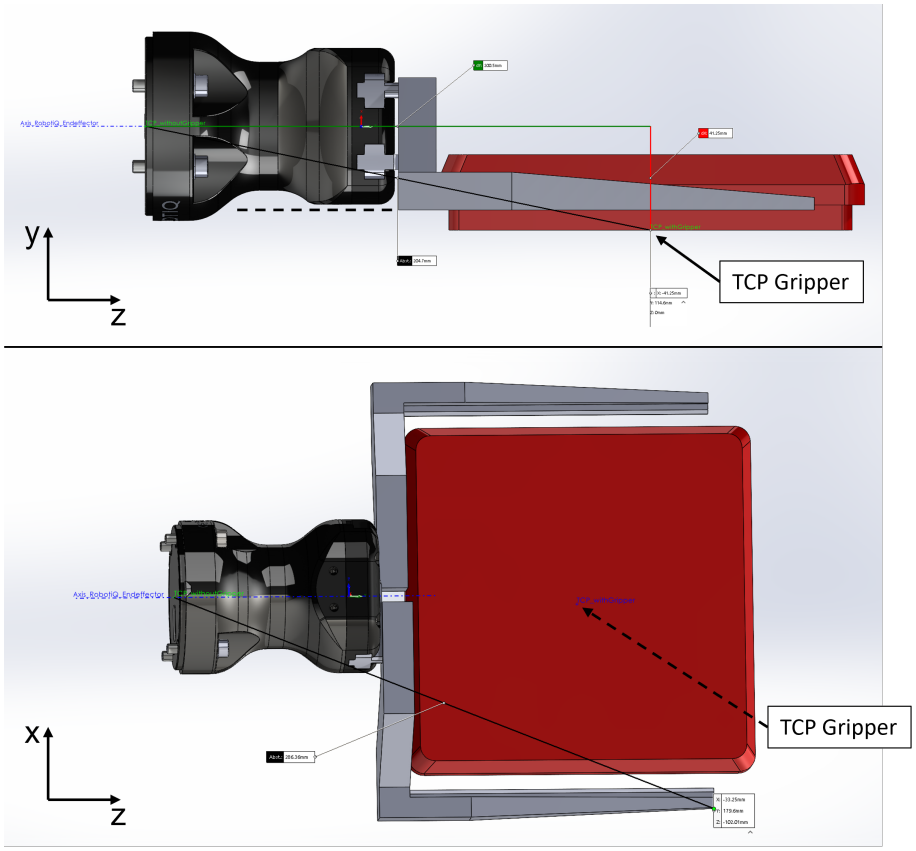


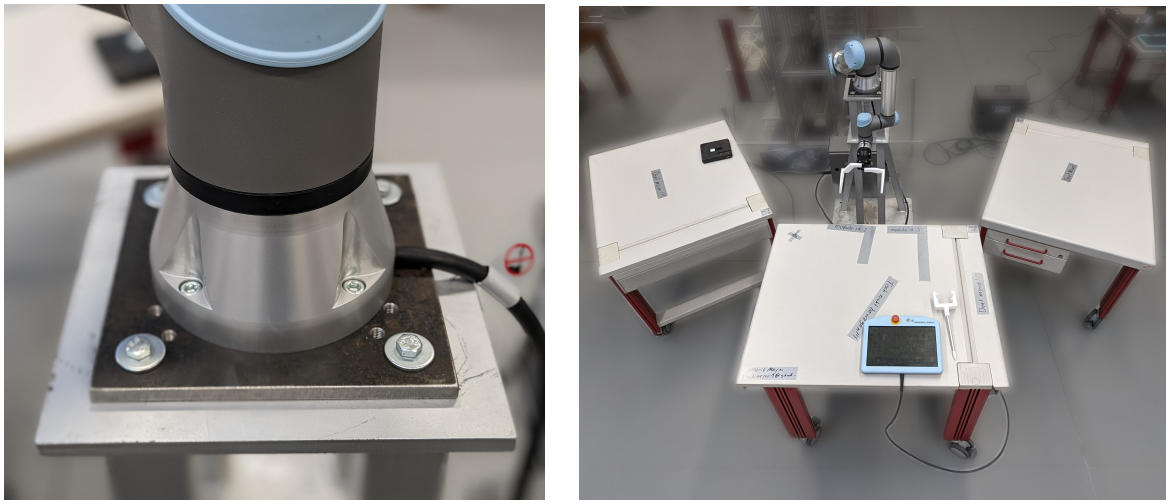
Figure 5.3.: Gripper construction with the base body in black (Robotiq Hand E) and the workpiece marked red, own illustration

UR5e Robot

After positioning the robot base and mounting the adapter plate on it, the UR5 is placed on the base and screwed down (see Figure 5.4a). The robot itself is only connected to the robot control, as is the teach pendant. The robot control must then be connected to the power outlet using a conventional protective contact plug (CEE 7/4) and to the network using an Ethernet cable. Then the gripper is mounted on the robot, or in this case, only the previously printed fingers of the gripper need to be screwed to the basic body of the gripper (Robotiq Hand-E), which is already connected to the robot.

Test Setup

The test setup's implementation aims to demonstrate the workpiece transfer unit's functionality as realistically as feasible. Therefore, a test configuration consisting of three tables is created. Each table represents one of the designed hexagonal system's six sides. The tables are placed at the appropriate height and distance to simulate the future arrangement of the modules.



(a) Robot base assembly (b) Overall test setup

Figure 5.4.: Robot base assembly & Overall test setup, own illustration

During the testing phase, the robot moves between tables and executes pick-and-place operations on each table. This setup permits actual testing of the robot's movements and functionality in a controlled environment and serves as a foundation for developing and refining the robot's programming. Figure 5.4b depicts the overall test setup.

Base Tool

Since the robot program is to work with planes (also called bases), a tool is needed to configure these planes. For the configuration, it is necessary to move to three points with the current TCP of the robot. This "touchup" must be carried out precisely to make the resulting plane usable in the robot program. Figure 5.5 shows the tool constructed for this purpose.

The shown tool has stops that allow it to be applied to the robot with high repeatable accuracy. The tool is secured with two screws on the top and bottom of the gripper's base body. The fingers of the gripper do not have to be dismantled for assembly, which facilitates handling and speeds up the configuration of new planes during operation.

The TCP of this tool is only shifted in the Z-direction by 350 mm, there is no shift in the X- and Y-direction. This has the advantage that it does not matter how the tool is rotated around the Z-axis. The tool has a tip at the front end, produced using FDM. With this, the points of the plane can then be probed. A machined aluminum tip would be more suitable for very high precision requirements. For this project, however, the PLA tip is sufficient.

The basic body of this tool, like the gripper fingers, is printed from PLA using FDM. The process parameters are set in Ultimaker Cura (version 5.2.2). Figure 5.6 shows the sliced model.

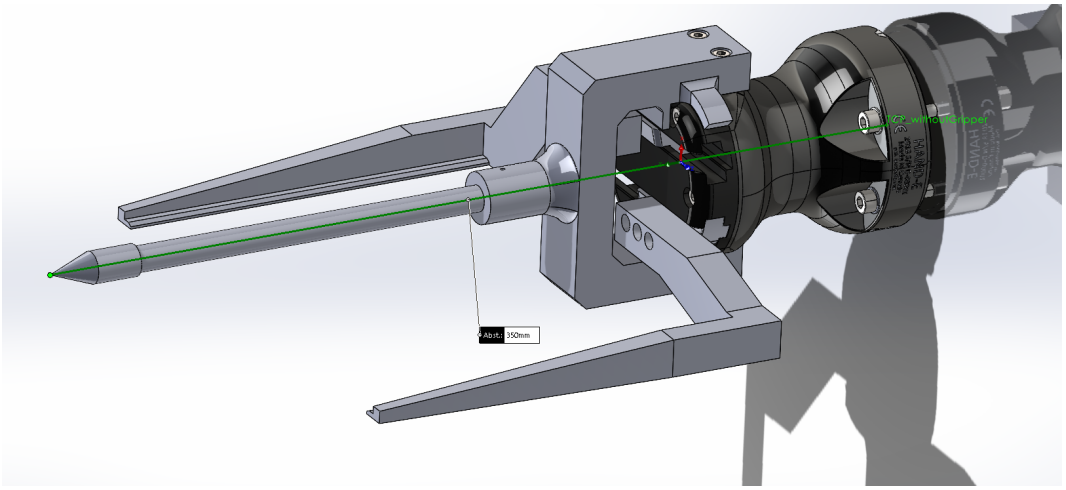


Figure 5.5.: Base tool design, own illustration

It is essential that the holes are printed without a support structure, as it is nearly impossible to remove this after printing without damaging the part. In addition, a horizontal hole expansion of 0.15 mm must be parameterized. In this way, the hole turns out so that it is unnecessary to re-drill the bore to assemble the subsequent aluminum rod.

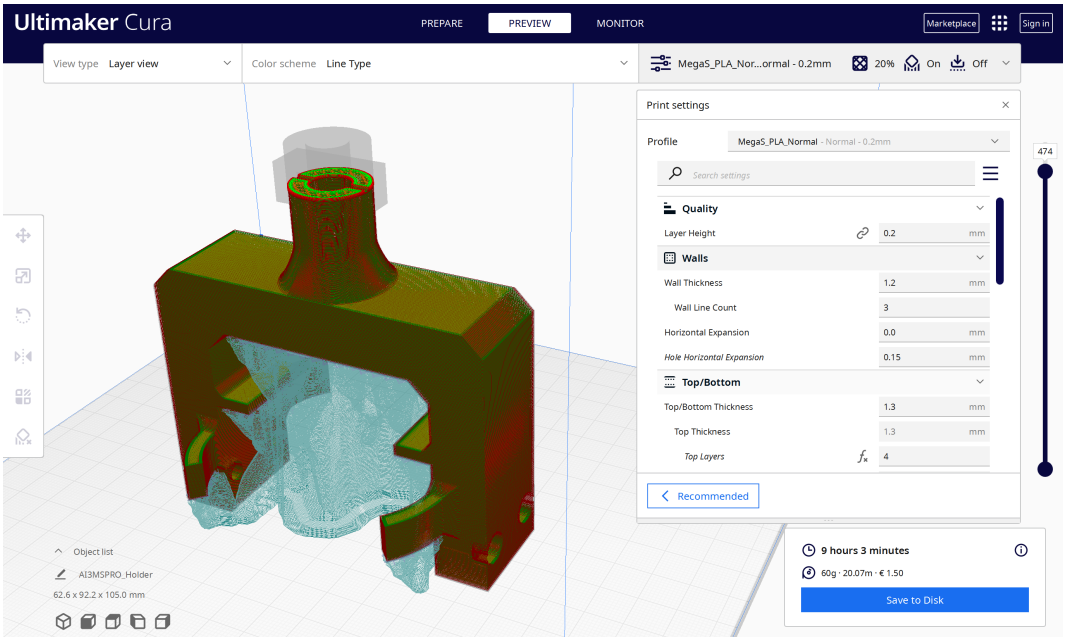


Figure 5.6.: Print Base Tool, own illustration

The ⌀10 mm aluminum rod is pushed into the hole provided after the primary body has been made. The rod has a length of approximately 200 mm. The exact length is insignificant because the tool will be measured in a later step. Then a two-millimeter hole is drilled through the aluminum shaft. The aluminum shaft must be positioned correctly and pushed as far into the bore as possible. The hole starts from the existing hole in the base body. After drilling, the aluminum shaft is fixed in position with the help of a cotter pin.

The plastic tip can then be placed on the other side of the aluminum rod due to its low weight. Gluing is also possible if needed but not needed here.

Workspace

The calculation of the workspace is essential for the determination of pick and place positions. Especially in future projects, it is needed so that it is always known which position the robot can probably reach and which position it cannot.

The following formulas provide a calculation basis for determining the workspace. Input parameters for the formula are the robot stands height, the robot's geometry, and the gripper used. Therefore, the calculated workspace refers to the TCP of the gripper, i.e., the center of the underside of the clamped workpiece.

In the first two formulas, the displacement of the center of the workspace in the Z direction (z_0) and the radius of the workspace (r_W) are calculated. The assumption here is that the workspace is a sphere.

$$z_0 = h_{Stand} + h_{Joint_1} + h_{TCPgripper} = (1060 + 90 - 41.25)mm = 1,108.75mm$$

$$r_W = r_{Robot} + r_{Gripper} = (850 + 200.5)mm = 1050.5mm$$

In the second part, two formulas are now set up in which a position can be entered that is to be reached with the robot. If both formulas are true, the robot is expected to be able to reach the position. In the second formula, a cylinder is taken out of the spherical workspace, which the robot cannot reach according to the data sheet of the robot [Uni18a] [Uni18b].

$$1,050.5mm \leq \sqrt{(x - 0mm)^2 * (y - 0mm)^2 * (z - 1,108.75mm)^2}$$

$$151mm \leq \sqrt{(x - 0mm)^2 * (y - 0mm)^2}$$

If the formulas are fulfilled, additional obstacles and other workspace restrictions must be considered. It is advisable to carefully approach the desired position with the robot to detect difficulties. Figure 5.7 shows the resulting workspace related to the planned layout of the digital factory.

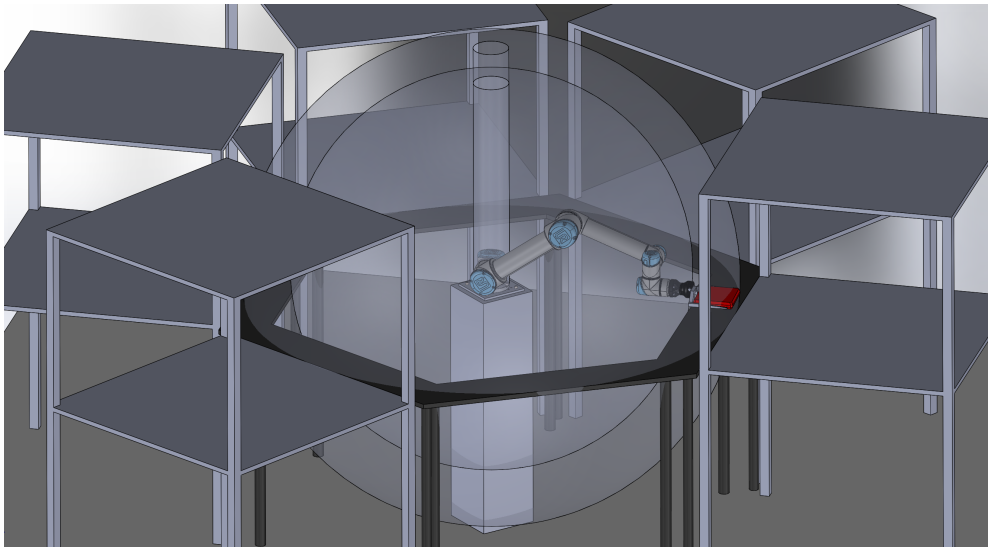


Figure 5.7.: Workspace in relation to the planned digital factory, own illustration

In the figure, the smaller workspace shows the standard workspace of the robot, and the larger one shows the workspace with the attached gripper. If the input parameters change, e.g. if the robot stand is changed to the proposed design, the workspace must be redesigned accordingly.

5.2. Software Implementation

The concept of this project is to create a system that allows transporting workpieces from one place/module to another. The core feature here is that this transport is not programmed statically but can be "controlled" by others outside the system. This control from outside of the system is no control in the ordinary sense but happens with the provision of variables that allow changing the flow of the robot program. The functionality is implemented and explained in the bottom-up approach. This means that first, the robot program is described, and then the communication path from the robot program, to the OPC UA server of the robot control, to the OPC UA client (PC), and finally to the OPC UA server (PC).

5.2.1. Robot Programming

The Robot Programming subsection describes the steps in programming and configuring the UR5e robot for seamless integration with the remainder of the system. This procedure comprises utilizing URCaps, configuring the TCP(s), and utilizing the plane feature to use the resulting planes in the robot program.

For most of the following steps, the robot is switched on and in manual mode, which is password protected. The selection for switching between manual and automatic mode is located at the top right of the robot menu (see Figure 5.8). It is also ensured that no person other than the instructor is directly in the robot's working area and that all necessary safety instructions are observed [Uni13]. Even though it is a cobot, where serious injuries are very unlikely, people can still be injured in unfavorable cases.

Robot Configuration

Before the actual programming of the robot can begin, some configurations must be carried out.

1. **Configuring TCP for the gripper:**
   In the following lines the necessary steps to set the TCP for the gripper are described. An example of the teach pendant view is shown in Figure 5.8.

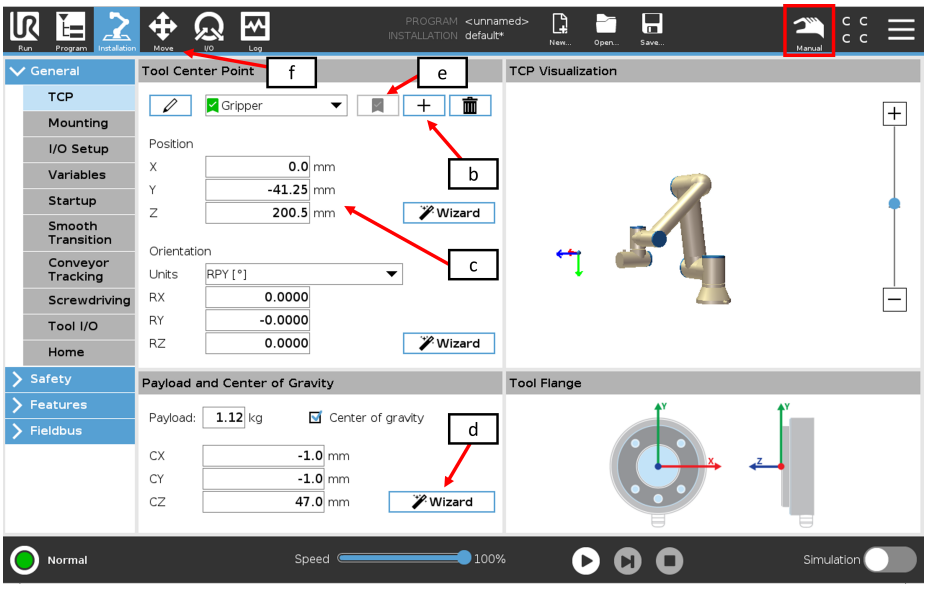


Figure 5.8.: Example of TCP setting for the gripper, own illustration

a) The gripper is attached to the robot. It is properly aligned and securely fastened.

b) In the Installation tab, in general, a new TCP is added and suitable name is entered.

c) The measured distances from the CAD drawing of the gripper are entered for the position of the TCP (X = 0 mm, Y = -41.25 mm, Z = 200.5 mm).

d) The wizard for determining the payload and the centre of gravity is carried out for the current configuration.

e) The new TCP is set as the default one.

f) In the Move tab, the newly set TCP is selected and some movements around its axes are executed. In this way it is possible to check that no gross errors have been made in the configuration.

2. **Configuring TCP for the base tool:**
In the following lines the necessary steps to set the TCP for the base tool using a wizard are described. An example of the teach pendant view is shown in Figure 5.9.

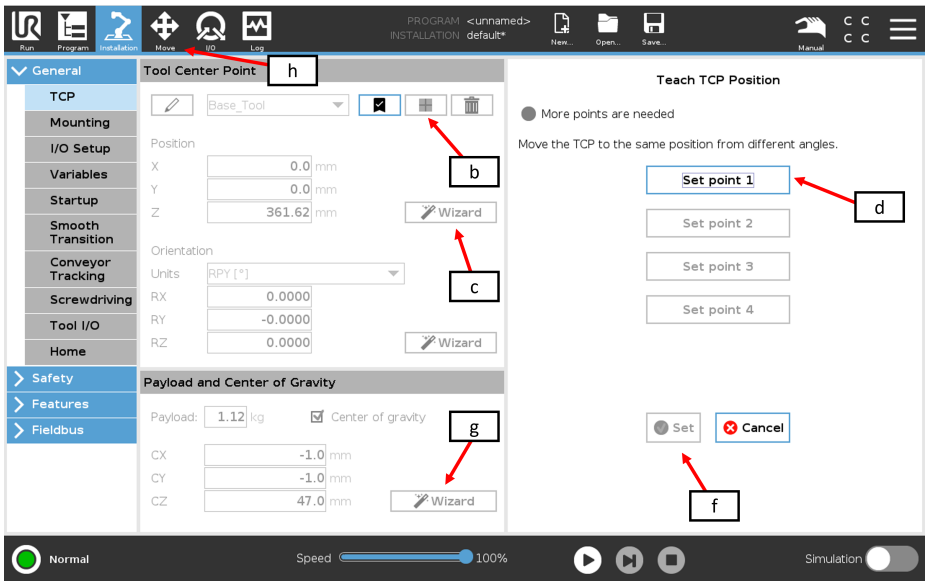


Figure 5.9.: Example of TCP setting for the base tool using the wizard tool, own illustration

a) The base tool is attached to the robots end effector (gripper).

b) A new TCP is added and a suitable name is entered.

c) The Wizard tool is used to configure the TCP of the Base tool.

d) The robot arm is moved so that the tip of the base tool is in contact with another tip, that is fixed in position.

e) The base tools TCP is moved to the fixed tip and the position is teached by pressing the "Teach" button. This step is repeated for four points in order to obtain an accurate measurement of the tool's tip position.

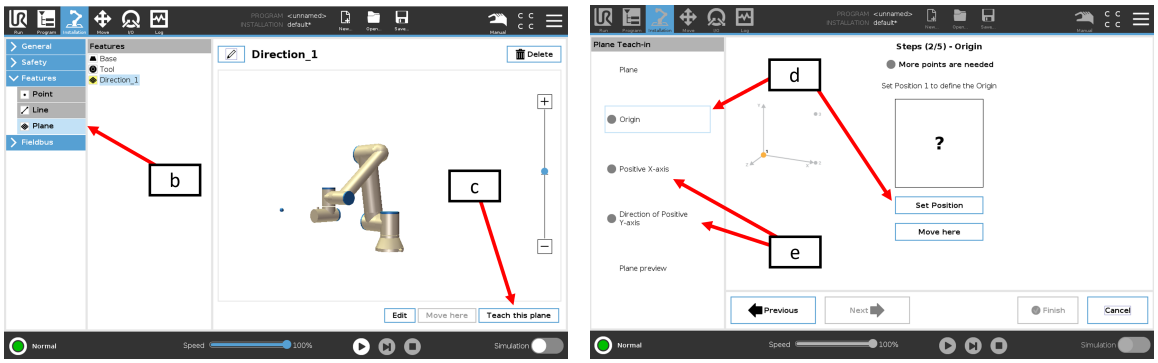
f) Once all positions are taught and the configuration is set, the robot arm automatically determines the position of the TCP.

g) The wizard for determining the payload and the centre of gravity is carried out for the current configuration.

h) By moving the robot around the new TCP, the configuration is validated.

By completing these steps, the TCP for the base tool is set, and it can be used in the next step to set the planes needed for the robot program.

3. **Configuring Plane:**
   In the following lines the necessary steps to configure a plane using the base tool and a wizard are described. An example of the teach pendant view is shown in Figure 5.10a.



(a) Adding and configuring a new plane feature using the wizard tool

(b) Teaching the origin and axes directions for the plane configuration

Figure 5.10.: Example of plane configuration using the wizard tool, own illustration

a) The base tool is attached to the gripper and its previously configured TCP is set as the active TCP.

b) In the installation tab, the features tab is selected. With the Add button, a new feature of the type plane is added and specified with a suitable name.

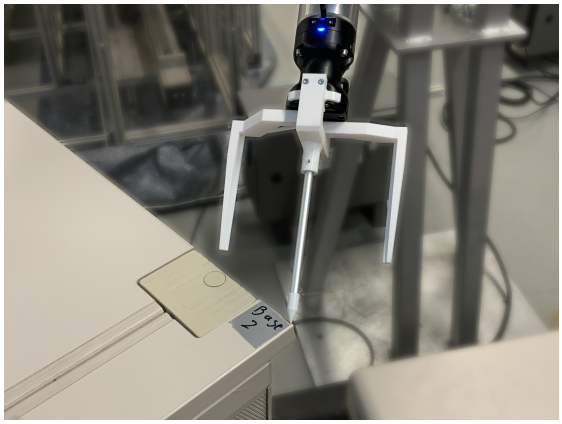
c) The plane is then configured with the help of a wizard.

d) In the first step, the robot arm is moved to the desired location of the new plane. This position will serve as its origin.

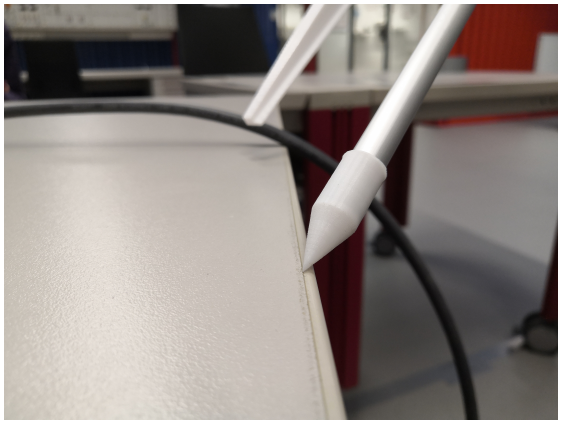
e) The positive X and Y direction are set in the next two steps using the teach pendant. These points are chosen as far away from the origin as possible. This increases the accuracy of the resulting plane.

f) The newly configured plane can be validated by positioning the robot at the origin of the plane and moving it in the X or Y direction. To do this, the plane must be selected as the active feature in the move tab. If the tip of the base tool now follows the edge of the real geometry, it has been configured correctly.

By completing these steps, the robot arm can establish a new plane using the TCP of the base tool and move with precision and accuracy according to the new reference plane. This feature is particularly useful when the robot needs to move and operate in a different part of the workspace or when the workspace itself is reconfigured. A new plane is configured for each of the three tables set up for testing purposes, as shown in Figure 5.11. In a future step, when the basic structure of the digital factory is in place, a level can then be configured for each side of this hexagonal frame.



(a) Using the base tool for setting the origin of the plane



(b) Using the base tool to establish the positive X and Y directions of the plane

Figure 5.11.: Example of setting a plane using the base tools, own illustration

Robot Program

In the robot program, the logic of the robot's operation is defined using code blocks. The fundamental structure of a robot program in URP is explained in Chapter 2.1.3, along with an example. Appendix C.1 shows the specific robot code developed for this project. The program's core consists of the main program, which invokes subprograms. Decisions made within the program, such as the positions from which parts are picked up and where they are placed, are determined using variables. The values of these variables correspond to their representations on the robot control's OPC UA server.

Figure 5.12 shows a flowchart of the main robot program. This part of the program contains a large part of the logic. The first step in the program, "Before Start", is only executed when the program is started for the first time. The gripper is activated, "Base_var" is set to "Base_1_const", the "Homee" subprogram is called (so the robot moves to its home position), and the OPC UA variables are read from the server for the first time. "Base_var" is always the active plane for all robot movements. When the "Base_var" is set to "Base_1_const", this means that the robot's current plane is set to "Base_1_const", so to one of the planes previously configured.

Everything that follows after this initialization step is the actual main program. This main program is (automatically) in a loop that is executed endlessly. First, in this loop, the robot moves to its home position. It should be noted that this home position depends on the current plane. After the start, this plane is "Base_1_const", but in later runs, it can be any of the up to six planes previously configured. Because of this, it is also essential that the type of movement used in the "Homee" subprogram is "Move Joints". If another movement type is used and the planes are on opposite sites of the robot, the robot crashes into itself.

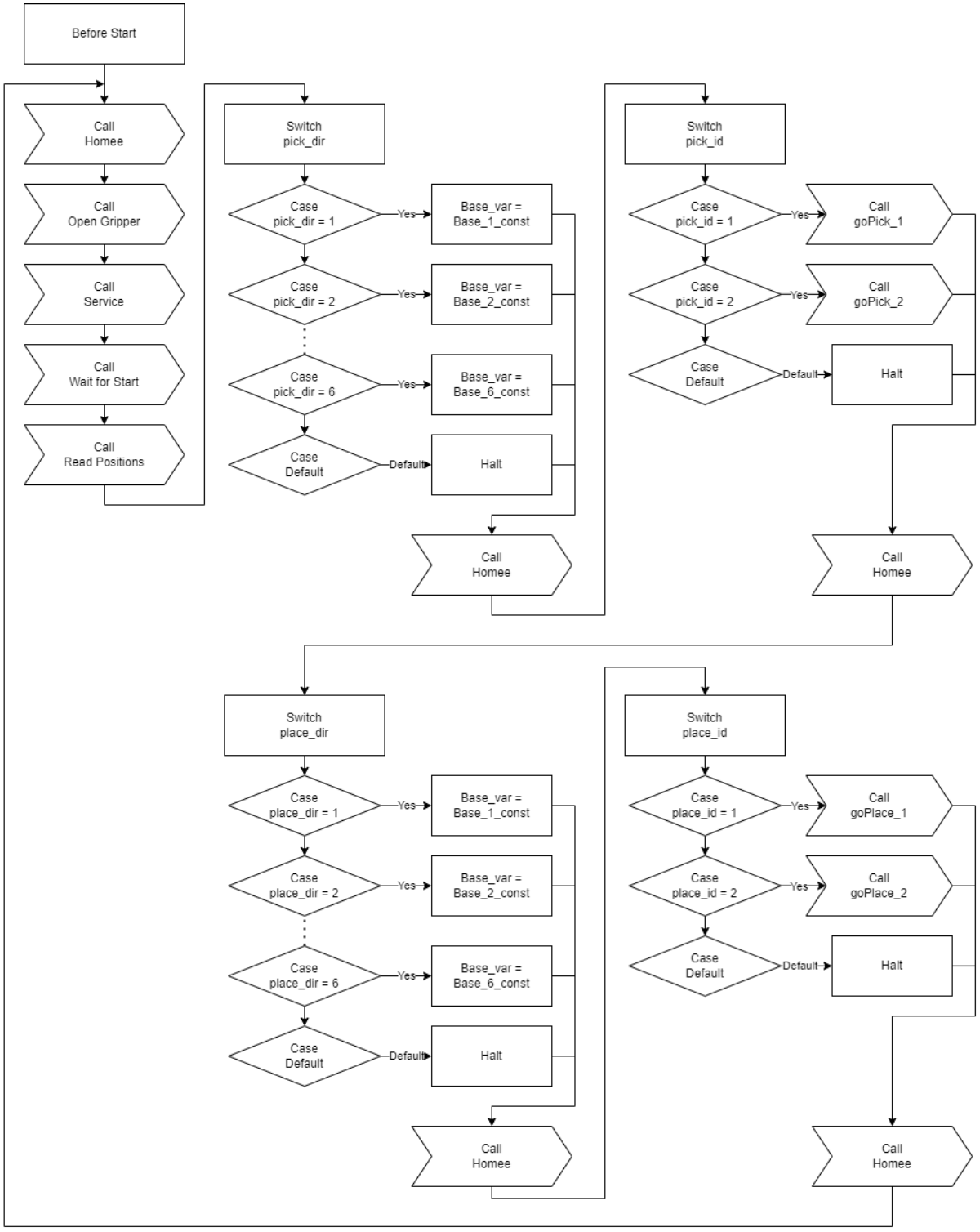


Figure 5.12.: Flowchart of the main robot program, own illustration

In the next step, the gripper is opened by calling "openGripper", and then the "Service" subprogram is called. In this subprogram, the "service" variable is checked, and if this variable is not zero (but 1,2,3,4,5,6), the robot goes to one of the specified positions and "Halts" there. When this happens, maintenance operations can be done. In this state, it is only possible to move on with the program if the operator activates the robot again with the help of the teach pendant. In this way, it is avoided that an external system or person interferes with this state by addressing the OPC UA variables and endangering the operating person.

After the service variable has been queried, the sub-program "Wait for start" is called. Here, the variable "isBusy" is first set to false. Then the "start" variable is continuously queried from the OPC UA server and awaited until it becomes true. This is the starting point for the program. Namely, the robot remains in the loop if an external system or person does not set the variable to true. However, if the variable becomes true, the robot breaks out of the loop, sets "start" directly back to false and the variable "isBusy" to true. Directly switching the "start" variable prevents the robot from starting in the next run without an external system having requested this start. Furthermore, by setting the "isBusy" variable to true, external systems can recognise that the robot is currently in a program run, i.e. that it is busy.

After the "start" variable has been set to true, the robot goes into the "Read Position" subprogram and queries the variables for the direction ("pick_dir", "place_dir") and the id ("pick_id", "place_id") of the pick and place positions he has to serve.

In the next part, which takes up most of the space in the flowchart 5.12, the active plane "Base_var" is first set to one of the up to six planes based on the variable "pick_dir" in a switch case branch. The robot is then moved to the specified home position by calling the "Homee" sub-program. From there, it picks the workpiece from a given position based on the "pick_id" variable.

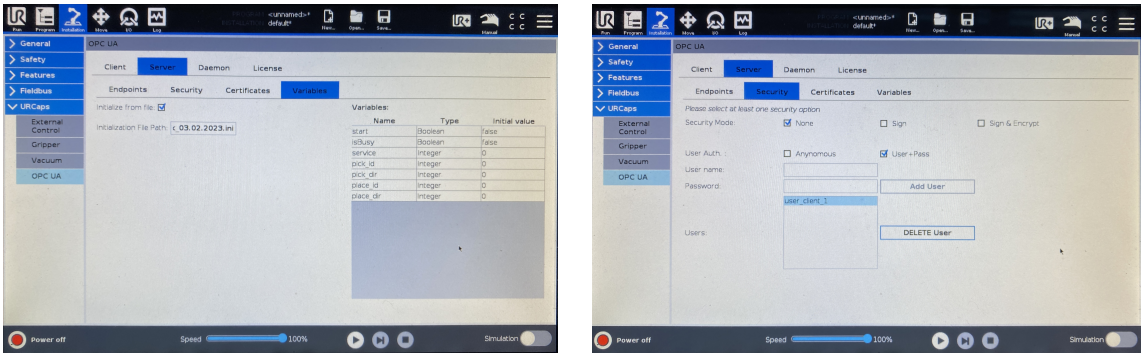
The subprogram called up for this purpose, e.g. "goPick_2", together with its counterpart (here "goPlace_2"), is the only part of the program that has to be adapted or added for the commissioning of a new module. Therefore, the number of possible modules is unlimited, and the effort for commissioning on the robot side is very manageable. To teach this subprogram, the module is connected to the later frame of the digital factory (Hexagon) in the direction of the "Base_1_const", and then the pick and place movement is taught. For the demonstration in this project, the workpiece is picked or placed from one of the tables, but the procedure itself is the same.

After the workpiece has been picked as described here, it is placed on another module (or table) in the following step based on the variables "place_dir" and "place_id". After this step, the robot returns to its current home position, and the main program starts again from the beginning.

OPC UA Robot (Robot Control)

For communication with the robot using the OPC UA protocol, only a server is required in this project for the robot (no client). This server's configuration is straightforward since an OPC UA URcap from Rockerfarm is used. First, the license is stored on a USB stick and then inserted into the robot's teach panel. The license is then copied to the robot control's hard drive, and the URcap can be used [Roc20].

Before the server is activated, some settings must be configured. First, an initialization file must be created on the computer in which the required variables are specified. To specify one variable, its name, its type, and starting value are written in a text file, each separated by a semicolon (e.g., "service;int;0"). The individual variables are then written in a separate line in the text document. The text document is saved and provided with the extension ".ini". With the help of a USB stick, the configuration file can be stored in the robot control. The URcap can initialize the variables based on the configuration file (Figure 5.13a). The variables can also be set using the teach pendant as an alternative to this procedure.



(a) Example of OPC UA Server's variable configuration

(b) Example of OPC UA Server's username & password configuration

Figure 5.13.: Example of OPC UA server configuration, own illustration

After the initialization of the variables, user authentication is configured. The safety configuration is done by adding a user with a name and password (Figure 5.13b). The name and password must be specified if a client wants to communicate with the server. In this way, a basic protection is ensured.

The server can now be started under the endpoints tab. In the robot program, the OPC UA variables must only be aligned with their robot counterparts before each use. For this purpose, the OPC UA URcap offers a read and a write function.

5.2.2. OPC UA

In the following chapter OPC UA server and client are programmed respectively the generated code is described.

OPC UA Client (PC)

In this project, the client represents the connection between the server running on a PC/server and the server running in the robot control. To accomplish this task, the client provides Python functions that ensure communication with the OPC UA server of the robot control. The generated Python code can be seen in C.2.

The code shown in Figure 5.15 summarizes the structure of the overall generated code. First, modules required for the following code are imported. Here, these are "asyncio" for the use of the async functions, which in turn is the basis for "asyncua" (opcua-asyncio), which is used to build the OPC UA Client (Chapter 2.2). The module "logging" can be used to monitor the running client better, but it is not essential for the client in the version shown here.

```python
import asyncio
import logging

from asyncua import Client, ua

# Url for connection to UR5 with username and password (both have to be replaced with actual access data)
url_ur5 = "opc.tcp://name:password@192.168.157.233:4840"


# Client functions used to read and write variables from/to OPC UA Server in robot control

# Read single variable of nodeID, return this value
async def read_var(nodeID):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        value = await node.read_value()
        return value


# Write single value to variable with nodeID
async def write_service(nodeID, value):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        await node.write_value(value, ua.VariantType.Int32)

```

Figure 5.14.: Code snippet of the OPC UA client, own illustration

First, after importing the modules, the URL of the robot is initialized together with the name of the client and the corresponding password. The name and password must match the user configured in the robot control; otherwise, the communication will be blocked by the OPC UA server of the robot control.

A series of functions follow, with their structure always being roughly the same. In the first function "read_var" the "nodeID" of the variable to be read is passed, the corresponding variable is queried and then returned with the "return" statement. In

the intermediate steps a connection to the OPC UA server of the robot control is established first, for which the previously defined URL is used. Then first the "node" of the variable is queried using the "nodeID" and finally the value of the node is read.

The second function looks very similar, but a value is passed or written and not read here. For this reason, a "value" is passed in addition to the nodeID. This value is then written to the variable associated with the nodeID. It should be noted that the type of the transfer value must be specified. The different variable types are available under "ua.VariantType". Here, for example, "Int" does not work as a transfer type, "Int32" must be used instead.

All other functions that follow either read values or write values to the OPC UA server of the robot control. The only difference are the types of the variables and the number of them.

The "main" function of the this client has no special function in the current implementation. However, if needed, it could be used for subscription handling, for example.

OPC UA Server (PC)

The OPC UA Server (PC) is responsible for managing the communication between the server of the robot control and other clients in the factory. For communication with other servers, it uses the functions provided by the OPC UA client discussed before. The generated Python code for the server can be seen in C.3.

First, as with the client, various modules are imported. Besides "asyncio", "asyncua" and "logging", "queue" is imported, which will be used for the creation of a queue. In addition to these modules, the functions previously defined in the client class are imported. This way, they can be easily called from the server.

In the next step, the node IDs of the variables needed in the robot control server are initialized. One method to easily find out these node IDs is to start the server of the robot and then connect to it using UAExpert. The variable names and the general structure of the server can be examined in this way. The node IDs are initialized as strings. Additionally, at the beginning of the code, a queue is initialized with "queue.Queue()" and a specified length. The usage of this will be explained in the upcoming "pick_and_place" function.

Three functions follow. The first function is used to set the service variable and thus to call a service subprogram in the robot code. In the function, only the nodeID of the service variable is passed and an integer value that determines which service program is ultimately called in the robot. The next two functions deal with the pick and place function of the robot, the central element of the whole project. The two functions can be seen in 5.15.

```python

# Queueing pick and place requests, return False when queue already full, True when it's not
@uamethod
async def pick_and_place(nodeID, pick_id, pick_dir, place_id, place_dir):
    if pap_queue.full():
        return False
    else:
        pap_queue.put([pick_id, pick_dir, place_id, place_dir])
        return True


# Start certain robot process
async def pap_action(pick_id, pick_dir, place_id, place_dir):
    # Set id of module and its direction, start Process
    # Pick
    await asyncio.create_task(write_pos(nodeID_pick_id, nodeID_pick_dir, pick_id, pick_dir))
    await asyncio.create_task(read_pos(nodeID_pick_id, nodeID_pick_dir))
    # Place
    await asyncio.create_task(write_pos(nodeID_place_id, nodeID_place_dir, place_id, place_dir))
    await asyncio.create_task(read_pos(nodeID_place_id, nodeID_place_dir))
    # Start
    await asyncio.create_task(write_start(nodeID_start, True))

```

Figure 5.15.: Pick and place functions of OPC UA Server (PC), own illustration

The function called pick_and_place(), is used for scheduling pick and place requests. The queue used for this task has been initialized before. The function takes four arguments: "pick_id", "pick_dir", "place_id", and "place_dir". These arguments represent the ID and direction of the modules used for picking and placing. When called, the method adds the pick and place request to a First-In-First-Out (FIFO) queue. If the queue is full, the method returns False; otherwise, it returns True.

To ensure that the queue does not become too long, the maximum length of the queue is set to a predefined value using the "pap_queue_length" variable, in this case to three. If the queue is full, new pick and place requests will not be added to the queue until older requests have been completed and removed from the queue.

The next function "pap_action()" practically just calls some "write_pos()" and "read_pos()" functions from the client class. First the ID, direction and the corresponding nodeIDs are passed for picking and then for placing. After writing, the just set variables are read from the server of the robot control. This process is only used for monitoring, but is not relevant for the actual function. In the end, the "start" variable is written, which starts the pick and place program in the robot.

The first step in the "main" function of the server is the creation of the OPC UA server instance using the "Server()" class. The server URL and the corresponding name are then set for the server instance. An address space is created with "server.nodes" and defines the namespace, which is required for the address space.

After the general setup for the server is completed, a number of arguments are initialized for the subsequent definition of the server functions/methods. The initialization is done with the help of "ua.Argument()". For each of the arguments a number of attributes are specified, such as the name and the data type.

In the following initialization of the OPC UA methods the previously defined "idx", the name of the method, the associated Python function and input and output variables are specified. These functions can then be called by other clients once the server is running. Figure 5.16 shows what the call looks like when performed using UaExpert.

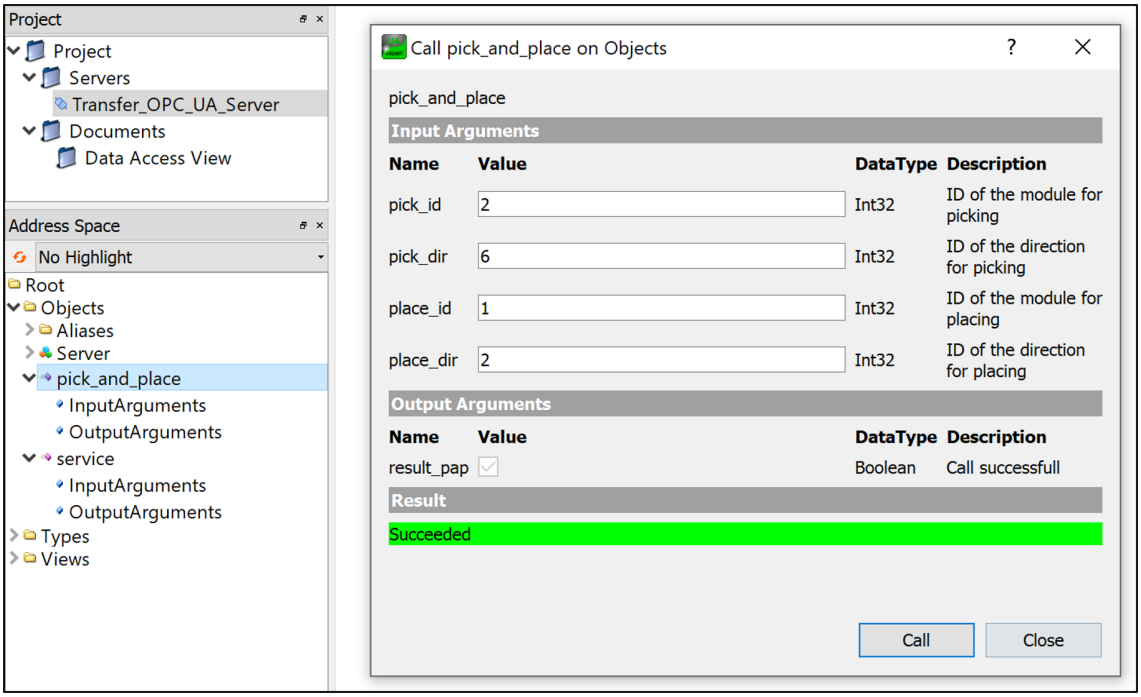


Figure 5.16.: Call of pick_and_place function using UaExpert, own illustrtion

The server is started and the following part of the code is in a loop, which is only interrupted by a 0.5 sec sleep. The code can be seen in Figure 5.17. In this loop, the server continuously checks the queue for new pick and place requests. When a request is detected and the robot is not busy, the server sends a pick and place instruction to the robot controls server, using the "pap_action()" described before. The arguments used there, are the ones stored in the queue, originally defined by a client as shown in Figure (5.16.

In addition, the current status of the robot and the length of the queue for the pick and place function are output to the console. This is only for monitoring purposes, but does not affect the actual function.

```python

    # Running Server
    logger.info("Starting Server!")
    async with server:
        while True:
            # Read/update variables
            robot_busy = await read_var(nodeID_isBusy)

            # Send pick and place instruction if one in queue
            if pap_queue.qsize() > 0 and not robot_busy:
                instruction = pap_queue.get()
                await asyncio.create_task(pap_action(instruction[0], instruction[1], instruction[2], instruction[3]))

            # Basic server functions/helper functions
            print("Robot busy = " + str(robot_busy))
            print("Queue size = " + str(pap_queue.qsize()))
            await asyncio.sleep(0.5)

```

Figure 5.17.: Pick and place functions of OPC UA Server (PC), own illustration

5.3. Manual

The following manual provides a simple instruction to start the assets proposed operation and some further hints. For procedures required for modifications of the system or commissioning of new modules, refer to the respective chapters in the previously described implementation.

Operating Instructions

Before operating, ensure that the system is connected to the power supply and to the network. All necessary components have to be fixed in position. For this operation, the gripper has to be mounted to the robot, the base tool should not.

1. **Positioning**: Position the workpiece to be transported at one of the designated tables.

2. **Power On**: Turn on the workpiece transfer unit's power supply.

3. **Connectivity**: Ensure that the robot control is connected to the network and that the OPC UA server is active. Start the OPC UA Server (PC) and use UaExpert to access it.

4. **Initiate Robot Program**: Run the robot program on the robot control by selecting the program using the teach pendant and pressing the run button. For this step, the robot can either be in automatic or manual mode.

5. **Select Target Position**: Use UaExpert to call the "pick_and_place" function with the provision of id and direction for the pick and place positions.

6. **Start Robot program Function**: After receiving the mentioned variable, the robot arm will move to the designated position to pick up the workpiece and transport it to the target position. The program will continue running and wait for further instructions.

7. **Maintenance**: The maintenance function is already included in the robot program. If maintenance is required, use UaExpert's interface to send a "service" signal to the workpiece Transfer Unit. The robot arm will then move to the designated maintenance position. Maintenance mode can be stopped by pressing the continue button in the teach pendant.

Emergency Stop

In the event of an emergency, press the emergency stop button on the control panel to immediately halt the workpiece transfer unit's operation.

Workpiece Size and Weight

The workpiece transfer unit is designed to accommodate workpieces with a specific size range and weight of up to 2 kg. If a workpiece exceeds these specifications, test the new conditions precisely before execution. Especially the gripper may not be able to handle the workpiece.

6. Validation

Several validation experiments on the workpiece transfer unit's subsystems and the complete system are carried out to guarantee the functionality and performance. The findings of these validation experiments are presented in the following chapter.

6.1. Validation of subsystems

The workpiece transfer unit comprises several subsystems, including the OPC UA server/client, robot arm, maintenance service function, and gripper. Each of these subsystems underwent validation tests to ensure their correct operation. These tests are carried out at the end of the project as well as throughout it, whenever a subsystem is implemented in ins function.

For the sake of clarity and to avoid duplicate entries, in the following mainly the requirements mentioned at the beginning will be listed. The subsystem validation is only listed if the content tested there is worth mentioning.

Validation of Service Function

The maintenance service function is tested to ensure that it operates correctly and meets the specified requirements. To validate the service function, the "service" variable is passed to the robot. Upon receiving this variable, the robot stops after performing the current pick and place function and moves to the maintenance position. This validation ensures that the service function is functioning correctly and the robot is able to respond appropriately to the service signal. The ongoing of the robot is in the tests just possible with it being activated using the teach pendant.

Validation of Gripper

The gripper is validated by testing its quality and functionality. It is discovered that the quality of the gripper is not satisfactory due to the fact that it is 3D printed using PLA. This highlights the need to manufacture a gripper with a higher mechanic rigidity for the final product.

6.2. Validation of requirement

It is essential to thoroughly validate each requirement to ensure that the asset satisfies the requirements specified in the project's specification. This section provides an overview of the requirements for the workpiece transfer unit and the methodologies used to validate each of them.

Validation of Functional Requirements

To validate the functional requirements of the workpiece transfer unit, several tests are performed. The results of these tests are summarized below:

- **Transfer function**: The workpiece transfer unit is able to successfully pick up and place workpieces in predetermined positions within the unit's limited number of positions. This requirement is validated through multiple tests and it is found to meet the specified performance requirements.
- **Transfer variability**: The tests that can be done so far confirm this variability of the system. Different pick and place instructions are passed to the system from multiple devices at once. But due to the digital factory's ongoing development, the transfer variability requirement can not be validated in its completion over a extended period. Once the digital factory is completed, the workpiece transfer device will be tested under various pick-and-place scenarios determined by an external system, such as a MES. This future validation aims to confirm the workpiece transfer unit's ability to adapt and respond to varying positional instructions provided by external systems, and successfully execute the required pick-and-place operations based on the positional information provided.
- **Workspace**: The workpiece transfer unit is able to move the workpiece within the predetermined workspace. This requirement is validated through several tests with different pick and place positions. However, it must be said that this workspace, as indicated in Chapter 5.1, does not include the optimal workspace to be achieved. To cover a significantly larger area, another robot, such as the UR10, would have to be used. The modules could then also be supplied directly with the robot, without the need for conveyor belts for every module.
- **Asset Dimension**: The asset is designed in such a way that the specified area is not exceeded and there is still space left to accommodate other components.
- **Structure of positioning area**: The workpiece transfer unit is created to facilitate workpieces transferred from a flat surface and a conveyor belt. Multiple tests validate the transfer from a flat surface. Still, the validation of the transfer from a conveyor belt is unresolved due to the ongoing construction of the digital factory and the manufacturing method of the gripper. With the current gripper, the needed accuracy can not be guaranteed. Nevertheless, the workpiece transfer unit's design accommodates the conveyor belt transfer, and its functionality can

be validated through future testing and the manufacturing of a mechanically rigid gripper (e.g. milled out of aluminum).

- **Positioning accuracy**: The robot's datasheet specifies a positioning precision of 0.1mm. However, it should be noted that the accuracy is affected by all of the components involved, especially the manipulator's performance, which is identified as a system flaw. The current gripper may deform over time under operational pressure, resulting in a loss of precision. As discussed in the segment on hardware implementation, replacing the current gripper with an aluminium gripper is recommended. While the current end effector may be able to satisfy the system's requirements in the current configuration, it may not be able to guarantee accuracy over time or under changed conditions.
- **Workpiece dimension**: The workpiece transfer unit is designed to accommodate workpieces with a specific geometry, ensuring that the workpieces do not exceed the unit's physical limitations. This requirement is validated by doing all of the tests with the actual workpiece at hand. Additional to that, due to the design of the gripper, minor differences/changes in the geometry of the workpiece are possible.
- **Workpiece weight**: The workpiece transfer unit is able to accommodate workpieces weighing up to 2 kg. This requirement is validated through several tests with the workpieces.
- **Time for transfer function**: The system is able to transport workpieces between positions in an average of under 30 seconds, meeting the specified performance requirement. This requirement is validated through several tests with different placing positions and movement as it can be seen in Table 6.1.

  The presented table exhibits the outcomes of a series of experiments conducted on the workpiece transfer unit to assess its performance in transporting workpieces from one position to another. In the experiments various combinations of starting positions and pick and place positions are tested. M1, M2, and M6 represent different modules of the unit. D1, D2, and D6 depict different direction/locations of these modules in the unit. The time taken to complete each test is recorded in seconds. The average time consumed across all tests is 17.10 seconds.

Case	Start Direction	pick_id (Pick Module)	pick_dir (Pick Direction)	place_id (Place Module)	place_dir (Place Direction)	Time [s]
1	D1	M2	D2	M2	D6	16.40
2	D1	M2	D6	M2	D2	18.79
3	D1	M2	D6	M1	D1	16.30
4	D1	M1	D1	M1	D2	17.05
5	D1	M1	D2	M1	D6	19.81
6	D6	M1	D6	M2	D1	15.04
7	D1	M2	D1	M1	D1	12.54
8	D1	M1	D1	M2	D6	15.43
9	D1	M2	D6	M1	D2	18.42
10	D6	M1	D2	M1	D6	21.08
Average Time Consumed						17.10

Table 6.1.: *Results of Workpiece Transfer Unit Tests*

Validation of Documentation Requirements

The following points deal with the control of the documentation requirements. The basis for the fulfillment of these requirements is the present documentation.

- A comprehensive and clear definition of the asset for the workpiece transfer unit, including its components, workspace, and other features, is developed.
- The asset definition is documented in detail in the project specifications and made available to all team members involved in the development and testing of the workpiece transfer unit.
- The asset definition is used as a reference throughout the development process to ensure that all components and software developed are aligned with the defined asset and its specifications.
- The asset definition was updated and refined throughout the development process to incorporate any changes or updates to the workpiece transfer unit's design and functionality.

Validation of Connectivity Requirements

The asset's connectivity requirements are intended to assure its seamless integration with other nodes in the digital factory system. To ensure the connectivity requirements are met, the following validation experiments are conducted:

- **Digital interaction between nodes**: The digital interaction between the workpiece transfer unit and other nodes in the digital factory system are validated by conducting tests on the system's OPC UA server/client subsystem. The validation tests confirme the integrity and security of the data transmission between the OPC UA server/client and other devices, including the OPC UA Server&Client (PC) and OPC UA robot control.
- **Industry 4.0 protocol and interface**:
  The compliance of the workpiece transfer unit's protocol and interface with Industry 4.0 standards is based on the choice of OPC UA as a communication technology. OPC UA is widely considered as the leading communication technology for Industry 4.0 applications [Pla18].

Validation of Safety Requirements

The safety requirement for the workpiece transfer unit is carefully considered throughout the design and development process to prevent any potential harm to individuals in the manufacturing environment. Hence, several safety tests are conducted in a controlled environment to validate the safety measures of the workpiece transfer unit.

- Emergency stop test: The emergency stop button is pressed while the workpiece transfer unit is in operation, and the system immediately haltes its movement to prevent any potential harm.
- Drop and slip test: The gripper is tested with various workpiece weights and robot speeds to ensure that it securely holds the workpiece during transportation and preventing it from falling or slipping out.
- Cobot function: The robot is stopped in operation by the application of force.

To conclude this chapter, the subsystems and requirements of the workpiece transport unit were validated to ensure its functionality and performance. The unit was designed and developed with these requirements in mind, although specific validations could not be completed for example due to the ongoing construction of the digital factory. The validation results provide a solid foundation for future development and successful implementation in real-world scenarios.

7. Conclusion and Outlook

This chapter summarises the project's findings, including validating the asset's subsystems and requirements and identifying possible unit enhancements. A analysis of the system's strengths, weaknesses, and enhancement opportunities is provided, emphasising the unit's prospective impact and future development opportunities.

7.1. Conclusion

This project's objective was to design and implement a workpiece transfer unit for a digital factory. The development process consisted of multiple phases, including the conception of hardware and software components, implementation, testing, and validation. The developed hardware consists of the robot stand, gripper, UR5e robot, test setup, and base tool. In contrast, the software consists of OPC UA and Robot programming for communication and integration between system components.

The validation of the subsystems, including the OPC UA server/client, the robot arm, the maintenance service function, the gripper, and the functional, documentation, connectivity, and safety requirements, confirmed the system's compliance with the project specifications. The workpiece transfer unit was designed and developed with these specifications in mind, despite the fact that certain validations, such as transfer variability and conveyor belt transfer, could not be concluded due to the ongoing construction of the digital factory.

The implementation and testing phase demonstrated that the workpiece transfer unit is capable of performing effective pick-and-place operations between three tables, simulating the future module arrangement in the digital factory. In providing a system that is functional and can be refined in the future, the project was able to accomplish its objectives. The validation results provide a solid foundation for further refinement and optimization, paving the way for the unit's successful implementation in real-world scenarios.

7.2. Outlook

The asset is the basis for future development and improvement of the digital factory system. Optimizing the end effector design for improved performance and efficiency is one area of future work. Further investigation of the hexagonal frame for the modules and integration of the workpiece transfer unit with the digital factory system is also necessary. As soon as the basic frame of the digital factory in the form of the hexagon has been completed and a module including the connected conveyor belt is ready, the first test runs can take place and the system can then be expanded with further modules. These steps are probably the most important ones in the near future.

Another potential avenue for future work is integrating digitalization solutions to improve the system's overall performance and efficiency. These advancements can be achieved by incorporating these technologies into the software components, enabling a better connected and adaptable workpiece transfer unit in the digital factory environment.

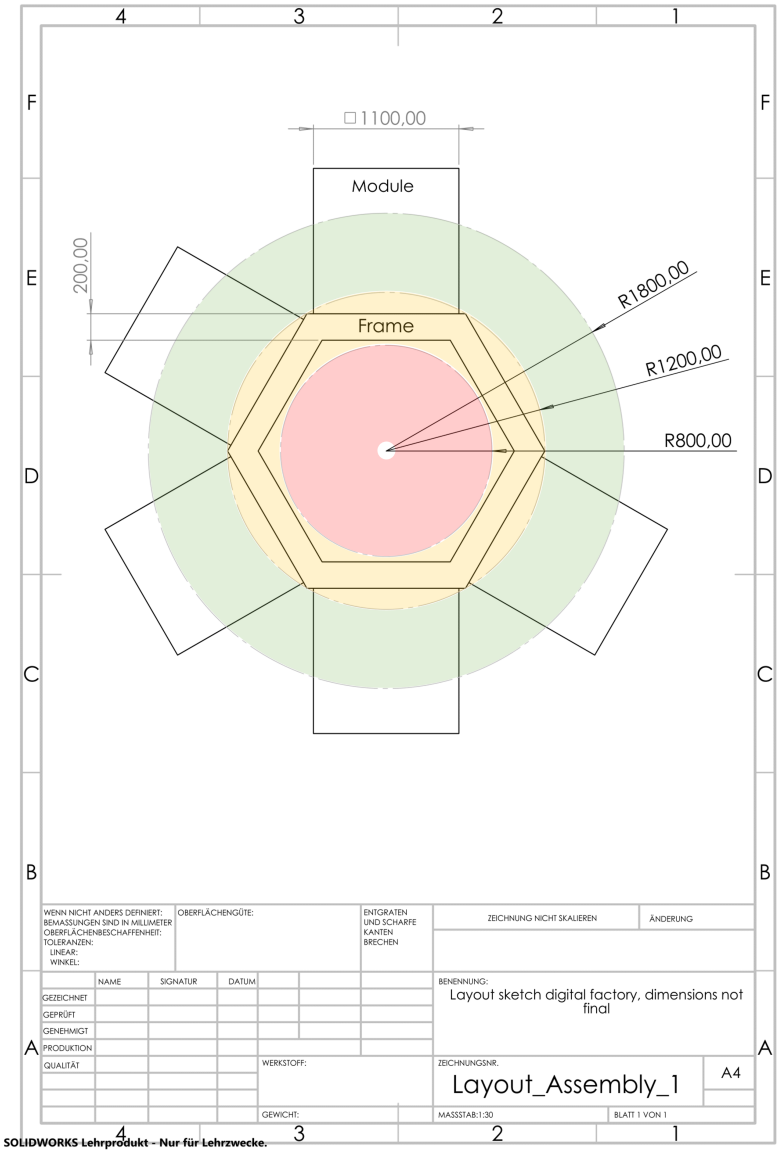
In order to verify the system's performance and functionality, testing and validation in a real-world environment can be conducted. This process entails integrating the system with actual industrial components, completing the remaining validations, such as transfer variability and conveyor belt transfer, and evaluating its performance in real-world scenarios.

In summary, the workpiece transfer unit offers a beneficial solution for developing the digital factory and lays the groundwork for further advancements and refinements in the future.

A. Specification

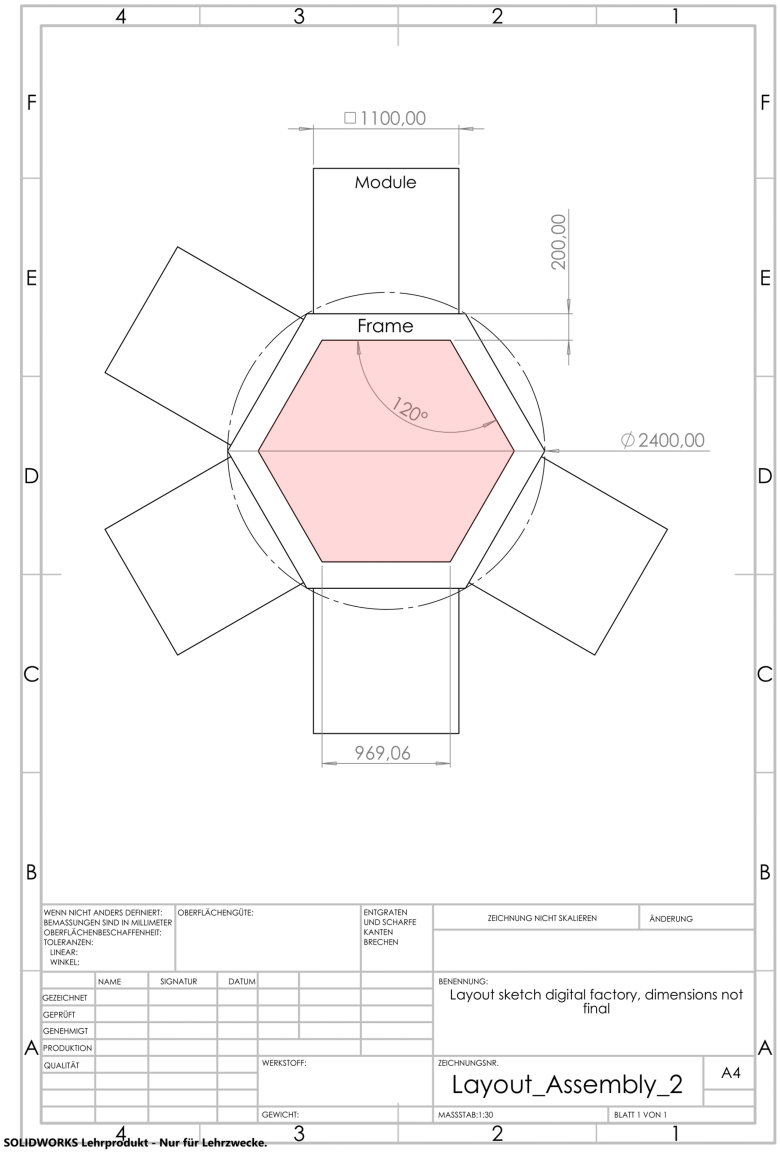
A.1. Workspace

Required workspace in relation to the expected layout of the digital factory. The red colored area describes the minimum requirement. The other two areas describe meaningful improvements. Own illustration.



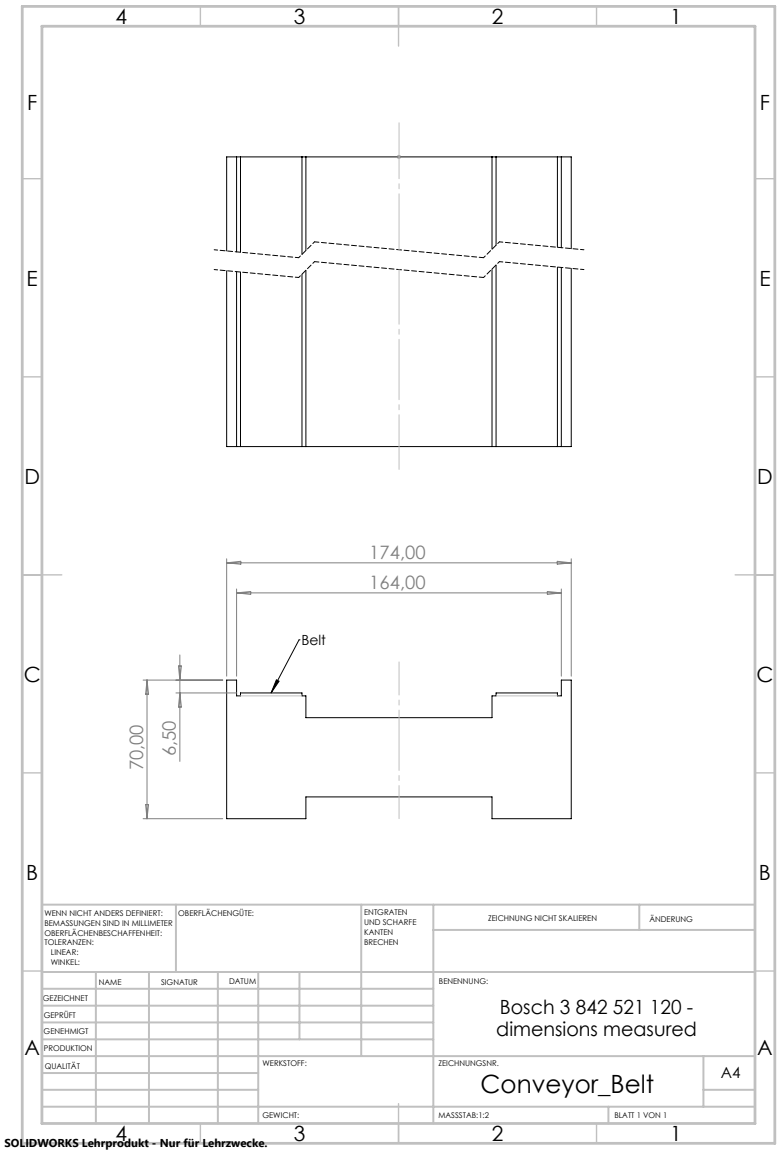
A.2. Asset Dimension

Maximum available area for the implementation of the asset in relation to the planned digital factory. Area marked red. Own Illustration.



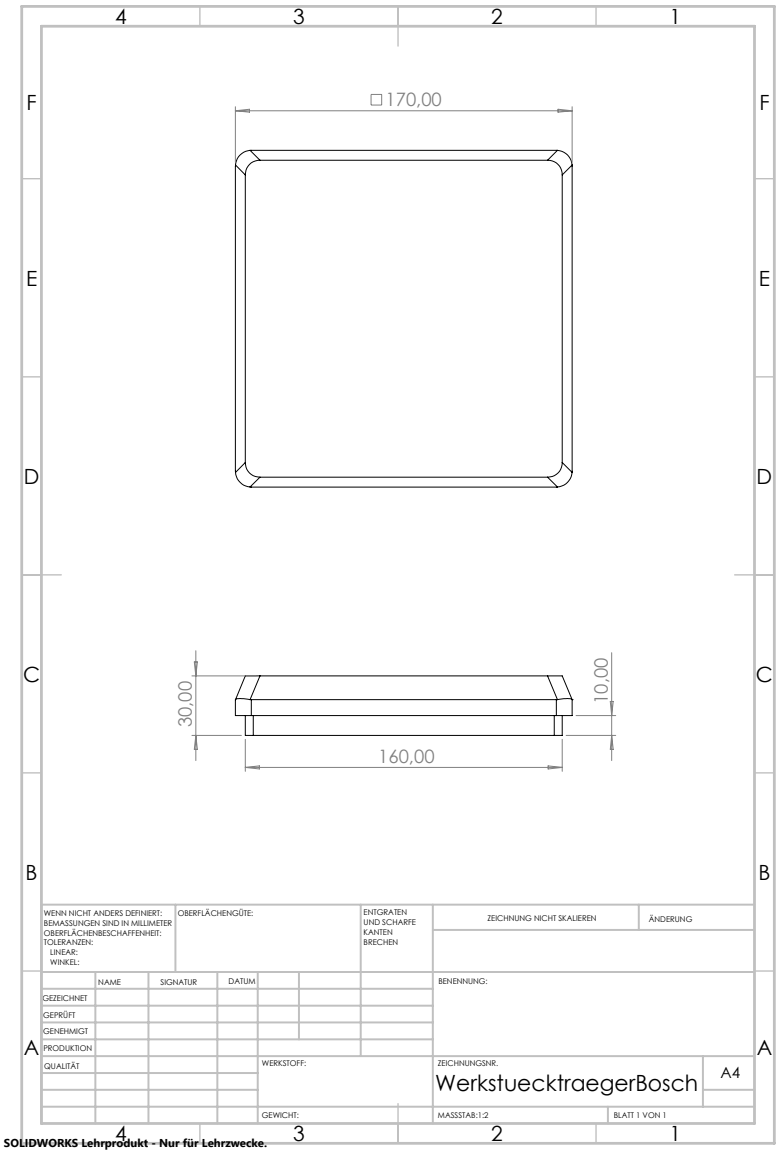
A.3. Conveyor Belt

Technical drawing of the conveyor belt used in the digital factory. Dimensions measured. Own illustration.



A.4. Workpiece

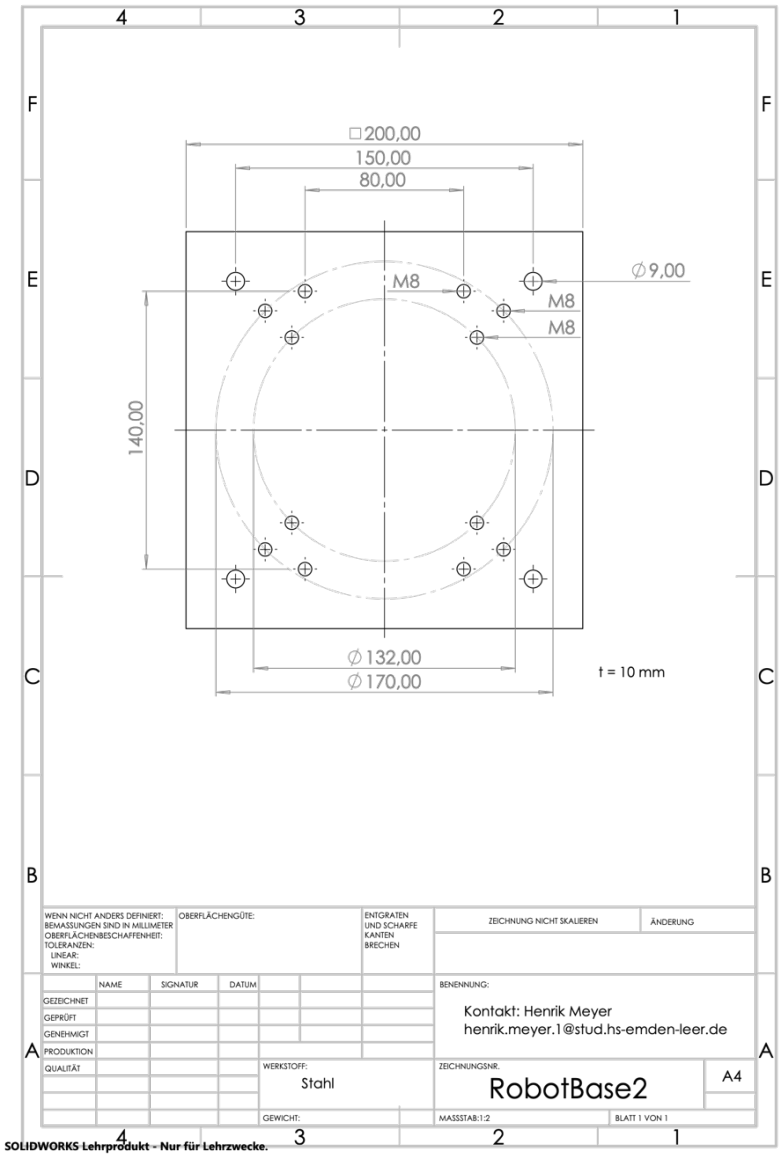
Technical drawing of the workpiece to be transported. Dimensions measured. Own illustration.



B. Hardware Implementation

B.1. Adapter Plate

Technical drawing of the adapter plate used for mounting the UR5e. Own illustration.



C. Software Implementation

C.1. UR5e Robot Program

```
BeforeStart
  Gripper Activate
  Base_varBase_1_const
  Call Homee
  Read OPC UA server: start
  Read OPC UA server: isBusy
  Read OPC UA server: service
  Read OPC UA server: pick_id
  Read OPC UA server: pick_dir
  Read OPC UA server: place_id
  Read OPC UA server: place_dir

Robot Program
  Call Homee
  Call openGripper
  Call Service
  Call wait_for_start
  Call read_pos
  Switch pick_dir
    Case 1
      Base_varBase_1_const
    Case 2
      Base_varBase_2_const
    Case 3
      Halt
    Case 4
      Halt
    Case 5
      Halt
    Case 6
      Base_varBase_6_const
    Default Case
      Halt
  Call Homee
  Switch pick_id
```

```
      Case 1
        Call goPick_1
      Case 2
        Call goPick_2
      Case 101
        'test'
      Default Case
        Halt
    Call Homee
    Switch place_dir
      Case 1
        Base_varBase_1_const
      Case 2
        Base_varBase_2_const
      Case 3
        Halt
      Case 4
        Halt
      Case 5
        Halt
      Case 6
        Base_varBase_6_const
      Default Case
        Halt
    Call Homee
    Switch place_id
      Case 1
        Call goPlace_1
      Case 2
        Call goPlace_2
      Case 101
        'test'
      Default Case
        Halt

  Service
    Read OPC UA server: service
    Switch service
      Case 0
        'no service needed'
      Case 1
        service0
        Write OPC UA server: service
        Base_varBase_1_const
```

```
      MoveJ
        Waypoint_1
      Halt
    Case 2
      service0
      Write OPC UA server: service
      Base_varBase_2_const
      MoveJ
        Waypoint_1
      Halt
    Case 3
      service0
      Write OPC UA server: service
      Halt
    Case 4
      service0
      Write OPC UA server: service
      Halt
    Case 5
      service0
      Write OPC UA server: service
      Halt
    Case 6
      service0
      Write OPC UA server: service
      Base_varBase_6_const
      MoveJ
        Waypoint_1
      Halt

Homee
  MoveJ
    posHome

openGripper
  Gripper Open (1)

wait_for_start
  isBusy False
  Write OPC UA server: isBusy
  Read OPC UA server: start
  Loop  True start
    Wait: 0.1
    Read OPC UA server: start
```

```
    start False
    Write OPC UA server: start
    isBusy True
    Write OPC UA server: isBusy

  read_pos
    pick_id0
    pick_dir0
    place_id0
    place_dir0
    Read OPC UA server: pick_id
    Read OPC UA server: pick_dir
    Read OPC UA server: place_id
    Read OPC UA server: place_dir

  goPick_1
    MoveP
      posTo_1_down
    MoveL
      pos_1_down
    Gripper Move45% (1)
    MoveL
      pos_1_up
    MoveL
      posTo_1_up

  goPlace_1
    MoveL
      posTo_1_up
    MoveL
      pos_1_up
    MoveL
      pos_1_down
    Gripper Open (1)
    MoveL
      posTo_1_down

  goPick_2
    MoveP
      posTo_2_down
    MoveL
      pos_2_down
    Gripper Move45% (1)
    MoveL
```

```
    pos_2_up
  MoveL
    posTo_2_up

goPlace_2
  MoveP
    posTo_2_up
  MoveL
    pos_2_up
  MoveL
    pos_2_down
  Gripper Open (1)
  MoveP
    posTo_2_down
```

C.2. OPC UA Client

```
import asyncio
import logging

from asyncua import Client, ua

# Url for connection to UR5 with username and password (both have to
                                    be replaced with actual access
                                   data)
url_ur5 = "opc.tcp://name:password@192.168.157.233:4840"


# Client functions used to read and write variables from/to OPC UA
                                    Server in robot control

# Read single variable of nodeID, return this value
async def read_var(nodeID):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        value = await node.read_value()
        return value


# Write single value to variable with nodeID
async def write_service(nodeID, value):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        await node.write_value(value, ua.VariantType.Int32)


# Write single value to variable with nodeID
async def write_start(nodeID, value):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        await node.write_value(value, ua.VariantType.Boolean)


# Write two values, here position represented by current id and dir
                                    value
async def write_pos(nodeID_id, nodeID_dir, id, dir):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID_id)
        await node.write_value(id, ua.VariantType.Int32)

        node = client.get_node(nodeID_dir)
        await node.write_value(dir, ua.VariantType.Int32)


# Read two values, here position represented by current id and dir
                                    value, return these values
async def read_pos(nodeID_id, nodeID_dir):
```

```
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID_id)
        value_1 = await node.read_value()

        node = client.get_node(nodeID_dir)
        value_2 = await node.read_value()

        return [value_1, value_2]


async def main():
    logger = logging.getLogger(__name__)

    # Running Client
    logger.info("Starting Client!")
    async with Client(url=url_ur5) as client:
        while True:
            print("Counter Client")
            await asyncio.sleep(0.5)


if __name__ == "__main__":
    asyncio.run(main())
    # logging.basicConfig(level=logging.DEBUG)
    # asyncio.run(main(), debug=True)
```

C.3. OPC UA Server

```
import asyncio
import logging
import queue

from asyncua import Server, ua
from asyncua.common.methods import uamethod

# Import of Client functions. This Server is connected to Server of
                                    robot control only via the Client
from Client import read_var, read_pos, write_start, write_service,
                                    write_pos

# Node IDs for communication with the OPC UA server of the robot
                                    control
nodeID_start = "ns=2;s=start"
nodeID_isBusy = "ns=2;s=isBusy"
nodeID_service_id = "ns=2;s=service"

nodeID_pick_id = "ns=2;s=pick_id"
nodeID_pick_dir = "ns=2;s=pick_dir"

nodeID_place_id = "ns=2;s=place_id"
nodeID_place_dir = "ns=2;s=place_dir"

# Fifo queue for storing pick and place requests, pap_quqeue_length
                                    defines the max amount of requests
                                     stored
pap_queue_length = 3
pap_queue = queue.Queue(pap_queue_length)


# Functions basically just write variables that are used in the
                                    robot program to start certain
                                    processes
# and/or read variables to monitor its state. Connection realized
                                    using Client functions.

# Put robot into one of its service positions/programs, return True
                                    when program runs through
@uamethod
async def service(nodeID, service_id):
    await asyncio.create_task(write_service(nodeID_service_id,
                                        service_id))
    return True


# Queueing pick and place requests, return False when queue already
                                    full, True when it's not
@uamethod
```

```
async def pick_and_place(nodeID, pick_id, pick_dir, place_id,
                                    place_dir):
    if pap_queue.full():
        return False
    else:
        pap_queue.put([pick_id, pick_dir, place_id, place_dir])
        return True


# Start certain robot process
async def pap_action(pick_id, pick_dir, place_id, place_dir):
    # Set id of module and its direction, start Process
    # Pick
    await asyncio.create_task(write_pos(nodeID_pick_id,
                                        nodeID_pick_dir, pick_id,
                                        pick_dir))
    await asyncio.create_task(read_pos(nodeID_pick_id,
                                        nodeID_pick_dir))
    # Place
    await asyncio.create_task(write_pos(nodeID_place_id,
                                        nodeID_place_dir, place_id,
                                        place_dir))
    await asyncio.create_task(read_pos(nodeID_place_id,
                                        nodeID_place_dir))
    # Start
    await asyncio.create_task(write_start(nodeID_start, True))


async def main():
    logger = logging.getLogger(__name__)

    # Setup server
    server = Server()
    await server.init()
    server.set_endpoint("opc.tcp://0.0.0.0:4840")
    server.set_server_name("Digital Factory Transfer")

    # Setup namespace
    uri = "https://github.com/heMeyer/UR5_OPCUA_1.git"
    idx = await server.register_namespace(uri)

    # Create root node for upcoming functions, variables...
    objects = server.nodes.objects

    # Prepare arguments for methods
    # Pick and place
    pick_id = ua.Argument()  # Implementation as a argument
    pick_id.Name = "pick_id"  # Display name
    pick_id.DataType = ua.NodeId(ua.ObjectIds.Int32)  # Data type
    pick_id.ValueRank = -1  # Amount of array dimensions (-1 equals
                                        scalar value)
    pick_id.ArrayDimensions = []  # amount of values in each array
                                        dimension
```

```
    pick_id.Description = ua.LocalizedText("ID of the module for
                                    picking")  # Display
                                    explanation
    pick_dir = ua.Argument()
    pick_dir.Name = "pick_dir"
    pick_dir.DataType = ua.NodeId(ua.ObjectIds.Int32)
    pick_dir.ValueRank = -1
    pick_dir.ArrayDimensions = []
    pick_dir.Description = ua.LocalizedText("Direction of the
                                    direction for picking")

    place_id = ua.Argument()
    place_id.Name = "place_id"
    place_id.DataType = ua.NodeId(ua.ObjectIds.Int32)
    place_id.ValueRank = -1
    place_id.ArrayDimensions = []
    place_id.Description = ua.LocalizedText("ID of the module for
                                    placing")
    place_dir = ua.Argument()
    place_dir.Name = "place_dir"
    place_dir.DataType = ua.NodeId(ua.ObjectIds.Int32)
    place_dir.ValueRank = -1
    place_dir.ArrayDimensions = []
    place_dir.Description = ua.LocalizedText("Direction of the
                                    direction for placing")

    result_pap = ua.Argument()
    result_pap.Name = "result_pap"
    result_pap.DataType = ua.NodeId(ua.ObjectIds.Boolean)
    result_pap.ValueRank = -1
    result_pap.ArrayDimensions = []
    result_pap.Description = ua.LocalizedText("Call successfull")

    # Service positions
    service_id = ua.Argument()
    service_id.Name = "service_id"
    service_id.DataType = ua.DataType = ua.NodeId(ua.ObjectIds.Int32
                                    )
    service_id.ValueRank = -1
    service_id.ArrayDimensions = []
    service_id.Description = ua.LocalizedText("Service Positions: 0
                                    = none, 1-6 = Maintanance
                                    Positions")

    result_s = ua.Argument()
    result_s.Name = "result_s"
    result_s.DataType = ua.NodeId(ua.ObjectIds.Boolean)
    result_s.ValueRank = -1
    result_s.ArrayDimensions = []
    result_s.Description = ua.LocalizedText("Call successfull")

    # Populating address space
```

```
    await objects.add_method(idx, "pick_and_place", pick_and_place,
                                        [pick_id, pick_dir, place_id,
                                        place_dir],
                                [result_pap])
    await objects.add_method(idx, "service", service, [service_id],
                                        [result_s])

    # Running Server
    logger.info("Starting Server!")
    async with server:
        while True:
            # Read/update variables
            robot_busy = await read_var(nodeID_isBusy)

            # Send pick and place instruction if one in queue
            if pap_queue.qsize() > 0 and not robot_busy:
                instruction = pap_queue.get()
                await asyncio.create_task(pap_action(instruction[0],
                                                     instruction[1],
                                                    instruction[2],
                                                    instruction[3]))

            # Basic server functions/helper functions
            print("Robot busy = " + str(robot_busy))
            print("Queue size = " + str(pap_queue.qsize()))
            await asyncio.sleep(0.5)


if __name__ == "__main__":
    asyncio.run(main())
    # logging.basicConfig(level=logging.DEBUG)
    # asyncio.run(main(), debug=True)
```

Bibliography

[aas23] aasx-package-explorer, *Aasx package explorer*, 2023. [Online]. Available: https://github.com/admin-shell-io/aasx-package-explorer.

[Col01] A. W. Colombo, *Digitalization of icps: The digitalization transformation*, HS Emden/Leer, 2023-03-01.

[Col21] A. W. Colombo, *Engineering industrial cyber-physical system: Life cycle engineering of industrial cyber-physical systems (icps)*, 2022-09-21.

[GHIU17] M. Graube, S. Hensel, C. Iatrou, and L. Urbas, *Information Models in OPC UA and their Advantages and Disadvantages*. Limassol, Cyprus: IEEE, 2017, ISBN: 9781509065059. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=8233358.

[HS 22] HS Emden Leer, *Projects for the digital factory: Winter semester 2022/23*, 2022.

[Ind] Industrie 4.0, *Din spec 91345:2016-04, referenzarchitekturmodell industrie 4.0 (rami4.0)*, Berlin. DOI: 10.31030/2436156.

[MLD09] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN: 978-3-540-68898-3. DOI: 10.1007/978-3-540-68899-0.

[opc23] opcua-asyncio, *Opcua-asyncio*, 2023. [Online]. Available: https://github.com/FreeOpcUa/opcua-asyncio.

[Pla18] Plattform Industrie 4.0, *Details of the administration shell: From idea to implementation*, Geschäftsstelle, Bülowstraße 78, D-10783, Berlin, 2018. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/vws-in-detail-presentation.pdf?__blob=publicationFile&v=12.

[Rob18] Roboitq, *Hand-e: Robotiq hand-e for e-series universal robots*, 2018. [Online]. Available: https://assets.robotiq.com/website-assets/support_documents/document/Hand-E_Instruction_Manual_e-Series_PDF_20190122.pdf.

[Roc20] Rocketfarm OPC UA, *User manual: Urcap*, 2020.

[Sch16] K. Schweichhart, *Reference architectural model industrie 4.0 (rami 4.0): An introduction*, 2016. [Online]. Available: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf.

[Uni13] Universal Robots, *Technische spezifikationen ur5*, Universal Robots, Ed., 2013.

[Uni18a] Universal Robots, *Universal robots e-series user manual: Ur5e: Orginal instrcution (en)*, 2018.

[Uni18b] Universal Robots, *Ur5 working area*, Universal Robots, Ed., 2018. [Online]. Available: `https://www.universal-robots.com/download/mechanical-e-series/ur5e/robot-working-area-pdf-ur5e-e-series/`.

[Uni21] Universal Robot Support, *Urcap - basics: The new platform for ur accessories and peripherals*, 2022-12-21. [Online]. Available: `https://www.universal-robots.com/articles/ur/urplus-resources/urcap-basics/` (visited on 03/19/2023).