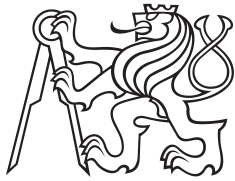


Bachelor Thesis



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Virtual Commissioning of Robotic Flexible Line with Conveyor System

Jan Šťastný

**Supervisor: Ing. Tomáš Jochman
Field of study: Cybernetics and Robotics
May 2024**

I. Personal and study details

Student's name: **Š astný Jan**

Personal ID number: **507209**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Virtual Commissioning of Robotic Flexible Line with Conveyor System

Bachelor's thesis title in Czech:

Virtuální zprovozn ní robotické flexibilní linky s dopravníkovým systémem

Guidelines:

1. Analyze the current hardware and software configuration of the workplace. Obtain the necessary information to design a new control system and virtual commissioning process.
2. Design the shuttle-oriented control logic of the conveyor system consisting of different modules (merge, split, arena) and propose an appropriate virtual commissioning process for the whole system.
3. Use simulation software to visualize and control the robotic flexible line. Implement the conveyor system control via simulated PLC. Create robotic pick and place operations that interact with the conveyor system. Extend simulation possibilities for virtual commissioning with the virtual robotic controller. Each component of the production line, i.e. robots, shuttles, will have an independent OPC UA interface.
4. Extend the PLC code for an above-laying Manufacturing Execution System to be able to control the virtual and physical production line.
5. Test the operation of the robotic flexible line with the conveyor system. Highlight the advantages and uses of your contribution over the original solution.

Bibliography / sources:

- [1] Petr Novák, Ji í Vysko il, Bernhard Wally. The Digital Twin as a Core Component for Industry 4.0 Smart Production Planning, IFAC-PapersOnLine, Volume 53, Issue 2, 2020, Pages 10803-10809, ISSN 2405-8963, <https://doi.org/10.1016/j.ifacol.2020.12.2865>.
- [2] Tomáš Staruch. Multi-agent Systems for Production Planning, Prague, Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering, 2023.
- [3] Vojt ch Outlý. ízení dopravníku v systému automatického rozvrhování výroby, Praha, eské vysoké u ení technické v Praze, Fakulta strojní, 2019.
- [4] SmartRouter Configuration. Montractec, 2018.
- [5] Montrac Data Acquisition and Control: Protokol Specification. 2nd. Montratec, 2018

Name and workplace of bachelor's thesis supervisor:

Ing. Tomáš Jochman Testbed CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **16.01.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Tomáš Jochman
Supervisor's signature

prof. Dr. Ing. Jan Kybic
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I want to thank my parents for their continuous financial and emotional support, my colleagues at Testbed for providing great working conditions and my classmate friends for having someone to share our difficulties with.

Děkuji svým rodičům za jejich podporu, kolegům v Testbedu za zajištění příjemného pracovního prostředí a kamarádům ze školy za vzájemné sdílení našich studijních problémů.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 24.5.2024

Jan Štastný

Abstract

This thesis focuses on the virtual commissioning of a robotic flexible production line integrated with a montratec conveyor system. The study makes use of the digital twin technology to simulate and optimize the production process, aiming to reduce downtime and improve system performance. Various simulation tools, including Process Simulate, TIA Portal, PLCSim Advanced, KUKA WorkVisual, and KUKA OfficeLite on Microsoft Hyper-V, are utilized to create an accurate digital representation of the production line. The thesis details the design and implementation of virtual commissioning, shuttle-oriented control logic for the conveyor system, and the integration of this control logic with robotic operations. The behavior is tested and validated through a series of experiments that demonstrate the efficacy of virtual commissioning in replicating real-world behavior and identifying areas for improvement.

Keywords: virtual commissioning; digital twin; production line simulation; montratec; robotic flexible line

Supervisor: Ing. Tomáš Jochman

Abstrakt

Tato práce se zaměřuje na virtuální zprovoznění flexibilní výrobní linky s roboty a dopravníkovým systémem montratec. Využívá technologii digitálních dvojčat k simulaci a optimalizaci výrobního procesu, s cílem snížit prostoje a zlepšit účinnost. Různé simulační nástroje, včetně Process Simulate, TIA Portal, PLCSim Advanced, KUKA WorkVisual a KUKA OfficeLite s Microsoft Hyper-V jsou použity k vytvoření přesné digitální reprezentace výrobní linky. Práce popisuje návrh a implementaci virtuálního zprovoznění, logiku řízení orientovanou na vozíky pro dopravníkový systém a integraci této logiky řízení s robotickými operacemi. Chování je testováno a ověřováno pomocí série experimentů, které ukazují účinnost virtuálního zprovoznění při replikaci reálného chování a identifikaci oblastí pro zlepšení.

Klíčová slova: virtuální zprovoznění; digitální dvojče; simulace výrobní linky; montratec; flexibilní robotická linka

Contents

1 Introduction	1		
2 Related work	3		
3 Workplace configuration	5		
3.1 Montrac conveyor system	7		
3.1.1 Routers	11		
3.1.2 Module configuration	11		
3.1.3 Logic and routing	13		
3.2 Robots	14		
3.3 Control system	14		
3.4 Communication	15		
3.5 Safety	15		
4 Shuttle-oriented control	17		
5 Virtual Commissioning	21		
5.1 Software tools used	23		
5.1.1 Process Simulate	23		
5.1.2 TIA Portal	23		
5.1.3 PLCSim Advanced	23		
5.1.4 KUKA WorkVisual	23		
5.1.5 KUKA OfficeLite on Microsoft Hyper-V	24		
5.2 Implementation	24		
5.2.1 Creating a Simulation Study	24		
5.2.2 Adding kinematics to the virtual line	25		
5.2.3 Modeling shuttles as Autonomous Guided Vehicles . . .	27		
5.2.4 Adding kinematics to robots and grippers	30		
5.2.5 Simulating conveyor components	31		
5.2.6 Connecting Process Simulate with a virtual PLC	33		
5.2.7 PLC code implementation	36		
5.2.8 Basic robotic operations	37		
5.2.9 Implementing VRCs	41		
5.2.10 Extending Process Simulate	43		
5.2.11 Communication	45		
5.3 Overview	46		
6 Experiments	49		
6.1 Testing conveyor line	49		
6.1.1 Table lookup	49		
6.1.2 Crossing and station objects .	49		
6.1.3 Operation	49		
6.2 Testing robotic workplaces	50		
6.2.1 Operations in standard mode	50		
6.2.2 Operations in line simulation with Virtual Robot Controllers . .	50		
6.3 Flexible line operation	50		
7 Conclusion	53		
A Bibliography	55		

Figures

<p>3.1 Photo of the montrac line 5</p> <p>3.2 Top-down diagram of the montrac line 6</p> <p>3.3 Montrac modules 1 9</p> <p>3.4 Montrac modules 2 10</p> <p>3.5 Configuration example – overview 11</p> <p>3.6 Configuration example – Route 1 12</p> <p>3.7 Configuration example – Route 2 13</p> <p>3.8 Configuration example – Route 3 13</p> <p>4.1 Montrac data structures 17</p> <p>4.2 Example of a router Function Block 18</p> <p>4.3 Example of a shuttle UDT 18</p> <p>4.4 Example of a module Function Block 19</p> <p>5.1 Diagram of proposed Virtual Commissioning 22</p> <p>5.2 Base study 24</p> <p>5.3 Setup of route 1 as HOME pose for crossing MA141 25</p> <p>5.4 Setup of route 2 as additional pose for crossing MA141 26</p> <p>5.5 Route control actions for crossing MA141 26</p> <p>5.6 Joint value sensors for crossing MA141 26</p> <p>5.7 Calculation of current route for crossing MA141 27</p> <p>5.8 Model of the shuttle AGV with gripping area (light blue) 27</p> <p>5.9 Extended interface of a shuttle . 28</p> <p>5.10 AGV targets 29</p> <p>5.11 Sensor body (blue) on a shuttle 29</p>	<p>5.12 Gripper links definition 30</p> <p>5.13 Gripper poses 31</p> <p>5.14 Diagram of physical crossing behavior 32</p> <p>5.15 Diagram of physical station behavior 33</p> <p>5.16 An example section of the PLC tag table 34</p> <p>5.17 An example section of the PS tags 34</p> <p>5.18 Example section of the generated TIA workbook 34</p> <p>5.19 Example section of the generated PS workbook 35</p> <p>5.20 Diagram of PLC setup phase . . 36</p> <p>5.21 Diagram of the main PLC loop 37</p> <p>5.22 Example testing battery – reality 38</p> <p>5.23 Example testing battery – simulation 38</p> <p>5.24 Robot setup example 39</p> <p>5.25 Example robot operation – pick from battery 40</p> <p>5.26 Diagram of a robot program cycle 42</p> <p>5.27 Example of the robot program selector 43</p> <p>5.28 Montrac extension ribbon bar . 43</p> <p>5.29 Camera snapshot example 44</p> <p>5.30 Gripper binding setup 45</p> <p>5.31 Gripper mounting signals 45</p> <p>5.32 Diagram of communication channels 46</p> <p>5.33 Process Simulate study image . 48</p> <p>6.1 Detection result 51</p>
---	---

6.2 Real robot picking a detected module	51
---	----

Tables

3.1 Overview of components by identifier	7
3.2 Overview of crossings by type ...	8
3.3 Effect of track stones on shuttle speed	14



Chapter 1

Introduction

With the increasing degree of automation and the corresponding design complexity of modern factories, implementing solutions without thorough testing is not optimal. While programs themselves can be simulated with mock data easily, the effect of real-world interactions such as object collisions and other physical constraints are missing. In order to prepare for the smoother construction of a complex assembly line, covering these points is essential. This can be achieved by creating a digital twin or Virtual Commissioning (VC) of the assembly line, specified such that most, if not all, impactful interactions are covered in the simulation.

Virtual Commissioning is the process of design, testing, and validation of software without the need for additional production downtime by using a digital twin. The results of this are significantly lower deployment costs, on-site time requirements, and the ability to try new processes and teach operators without the risk of damaging equipment. [1, 2, 3, 4, 5]

A digital twin can refer to various kinds of software simulations of real systems. Such simulations are usually much more complex than just Virtual Commissioning, which can be a part of them. These digital twins can reflect the state of the line in real time and be interconnected such that they can also affect said line with actuators or other control resources, creating a closed-loop system.

This work aims to create a VC of an existing line and use it for the design of a new shuttle-oriented control logic. This VC focuses on correctly simulating the robots, conveyor system, material flow, Programmable Logic Controller (PLC) program, and robot programs. It must also be able to connect to the existing Manufacturing Execution System (MES) via Open Platform Communications – Unified Architecture (OPC-UA) interface and be controlled in the same way the real line is.

The need for these changes on this specific line stems from the fact that before writing this thesis, the line used outdated components, which needed to be replaced, and the control system was no longer maintainable because it could

not be accessed properly.

The montrac conveyor system used in the real line did not have a digital counterpart yet, so the behavior had to be implemented from scratch in simulation software. The software used in this thesis includes the following:

- Process Simulate (PS)
- Totally Integrated Automation (TIA) Portal
- PLCSim Advanced
- KUKA WorkVisual
- KUKA OfficeLite
- Microsoft Hyper-V



Chapter 2

Related work

Virtual Commissioning (VC) has emerged as a pivotal methodology in the design, integration, and validation of complex manufacturing systems. VC allows for the pre-implementation testing of control systems by connecting them to simulation models that replicate the behavior of real plants. This approach mitigates risks and identifies potential issues before physical deployment, enhancing the efficiency and reliability of manufacturing operations. The concept of VC is integral to the implementation of Digital Twins (DTs), which are virtual representations of physical assets synchronized in real-time. This section reviews relevant literature on Digital Twin frameworks, methodologies, and Virtual Commissioning. [6, 7, 8]

Digital Twin frameworks provide the foundation for integrating DTs into manufacturing systems. Kritzinger et al. [9] define Digital Twins as virtual models that utilize real-time data to forecast and optimize production systems across different lifecycle phases. Various frameworks have been proposed to facilitate DT integration. Lu et al. [10] present a framework comprising an information model, communication mechanisms, and data processing modules. Tao et al. [11] introduce a five-dimensional DT framework including physical entities, virtual models, services, data, and connections. These frameworks share common elements such as the presence of a physical and virtual space, bilateral communication, and additional intelligence layers to support decision-making.

The design of DTs involves multiple stages, each enhancing the digital representation's capabilities. Kritzinger et al. [9] outline a methodology that progresses from Digital Models (DMs) to Digital Shadows (DSs), and finally to Digital Twins. This approach emphasizes the gradual enhancement of digital representations and their communication capabilities. However, traditional methodologies often overlook the intermediate steps necessary for developing the intelligence layer required for decision-making. Barbieri et al. [6] propose a stepwise approach using Virtual Commissioning (VC) to design, integrate, and verify DT architectures, providing a structured pathway for DT development.

Virtual Commissioning is a crucial tool for integrating and validating DT architectures within manufacturing systems. VC involves using virtual models to simulate and verify the control software of manufacturing systems, thus reducing deployment time and mitigating potential issues. Scheifele et al. [7] discuss a real-time co-simulation platform for VC, highlighting the integration of powerful simulation solutions based on integration interfaces and real-time co-simulation architectures. This platform utilizes the available computing power in real-time simulations through partitioning, parallelization, synchronization, and data exchange mechanisms.

Practical applications of Digital Twins and Virtual Commissioning have been explored in various manufacturing domains. For instance, Shen et al. [12] apply VC to tune control parameters of Computer Numerical Control (CNC) machine tools, while Burghardt et al. [13] use an immersive robotics environment integrating VC and Virtual Reality for automatic programming of industrial robots. Barbieri et al. [6] validate their methodology through a case study involving the integration of a DT into a flow shop for implementing a scheduling reactive to machine breakdowns.

Two studies explored the use of the Montrac line, a single-rail automated conveyor system, for the development of production control systems. In one study, the Montrac line was utilized to develop and test a system for automated control and monitoring. This study used the physical line directly and did not perform Virtual Commissioning. [14]

Another study applied a multi-agent system (MAS) to the Montrac line to enhance production planning and control. The MAS approach allowed for flexible and adaptive scheduling, accommodating dynamic changes in production requirements. This study partially used VC to test the developed algorithms, but the line simulation environment was not explored in detail and the focus was put on the real line. [15]

The approach in this thesis is to create a Virtual Commissioning environment based on the real modular montrac line, which will then be used to virtually develop and test a control system capable of handling the real line. This system logic will be designed such that possible changes in the line layout will be easy to implement. To achieve this, individual line components will be modeled as almost isolated objects that define the line layout via references. This introduces a sense of multi-agency in the system as their logic can be processed separately in each cycle.

Chapter 3

Workplace configuration

The line that was modeled already existed in Testbed and consisted of four robot workplaces and a monorail conveyor system used to transport resources between them.

Robot workplaces used KUKA Cybertech KR 8 R1620, and KUKA KR 10 Agilus 2. The monorail conveyor was a modular system called montrac produced by montratec GmbH. It consisted of plain rail segments, crossings, stations, shuttles, routers, sensors, and small metal block attachments that adjusted the speed of shuttles passing over them. The conveyor system was controlled by a network agent and the rest of the workplace with a Siemens SIMATIC S1500 PLC.

A photo of the assembly line can be seen in 3.1 and a top-down view diagram in 3.2. Components were specified by their identifier and an overview can be seen in 3.1. Details of how each component type works will be discussed in upcoming sections.



Figure 3.1: Photo of the montrac line

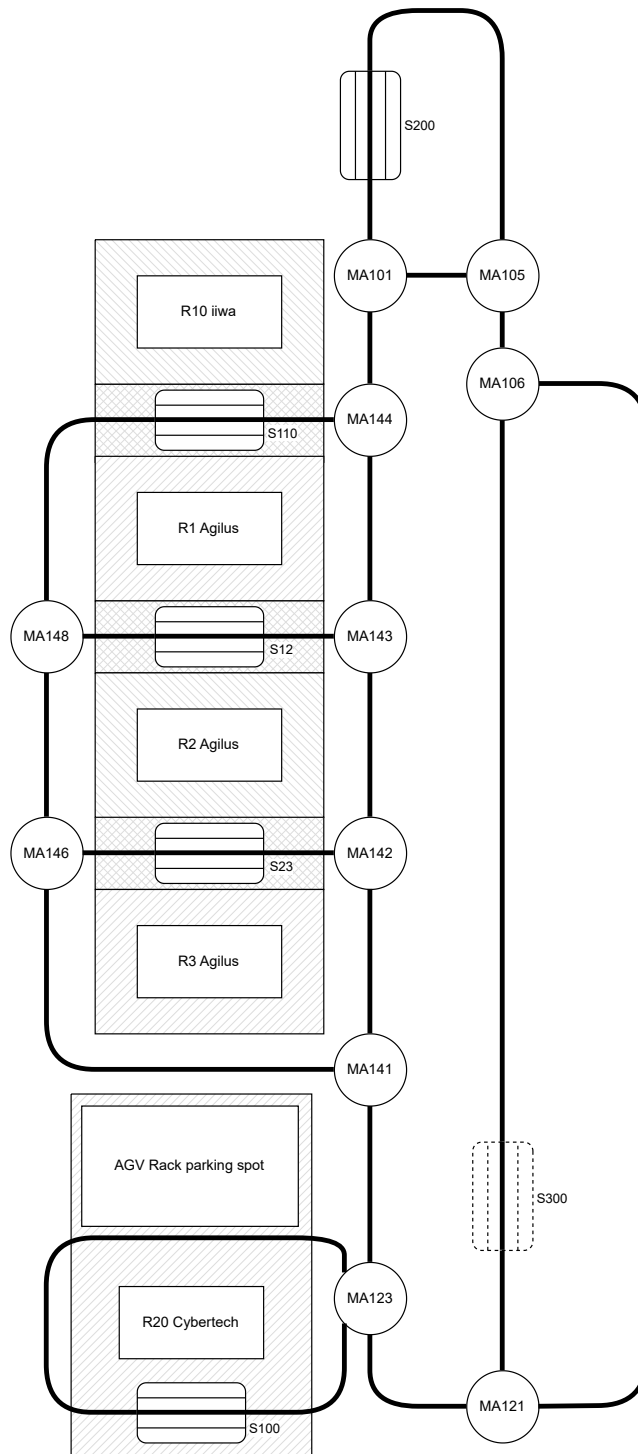


Figure 3.2: Top-down diagram of the montrack line

Identifier	Description
R1, R2, R3	Robots KUKA Agilus for pick and place operations
R20	Robot KUKA Cybertech for transferring materials from Autonomous Guided Vehicle (AGV) shelf to shuttles
S110	Stop station between R1 and R10
S12	Stop station between R1 and R2
S23	Stop station between R2 and R3
S100	Stop station at R20
S200	Stop station for manual pickup
S300	Virtual stop station for camera image capture
MA123	Crossing toward S100
MA148, MA146, MA141	Crossings joining S23, S12 and S110 with main loop
MA142, MA143, MA144	Crossings toward S23, S12 and S110 respectively
MA101	Crossing toward S200
MA105	Crossing joining S200 with main loop
MA106	Crossing toward S300
MA121	Crossing joining S300 with main loop

Table 3.1: Overview of components by identifier

3.1 Montrac conveyor system

Montrac used a modular system that could be customized to specific needs. Some modules used Infrared Modules (IRMs) to transfer information back and forth with shuttles. The line building blocks consisted of:

- Straight or slightly curved rail segments
- Curved rail segment – Required IRM to prevent shuttles from crashing.
- Routers – Allowed control and configuration of connected modules, remote management via User Datagram Protocol (UDP) or a web server.
- Physical station – Required IRM, allowed locking shuttles, and could be configured to act as the following station types:
 - Stop Station – Basic configuration, stops the arriving shuttle if it has an entry in routing tables and releases it otherwise. Allows setting a new target for the current shuttle.
 - StopWait Station – Allows setting a new target for a specific shuttle. If this shuttle is not currently in the station, the station sends the occupying shuttle away and checks the next one.
 - Edit Station – Allows for preloading of target assignments via processing table and automatically assigns the new addresses to

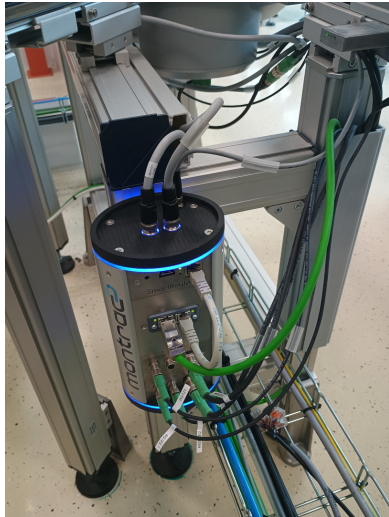
arriving shuttles. If a target is assigned, the shuttle stays in the station and departs otherwise.

- EditBusy Station – Similar to Edit station but allows selection from multiple tables.
- Virtual station – Required IRM and did not have physical construction.
- Signoff sensor – Used to detect shuttles leaving an outlink (module output).
- Crossings – Required one or more IRMs, one placed on each inlink (module input), and had multiple variations:
 - Collect L/R – Merged multiple inlinks into one outlink.
 - Divide L/R – Split one inlink into multiple outlinks.
 - Arena – Connected multiple inlinks into multiple outlinks.
 - Lift – Transferred shuttles between two z-levels of the track.

Photos of chosen modules can be seen in 3.3 and 3.4. An overview of used crossings sorted by type is shown in 3.2.

Type	Crossings
Arena	MA123
Divide Right	MA101
Divide Left	MA106, MA142, MA143, MA144
Collect Right	MA105, MA121
Collect Left	MA141, MA146, MA148

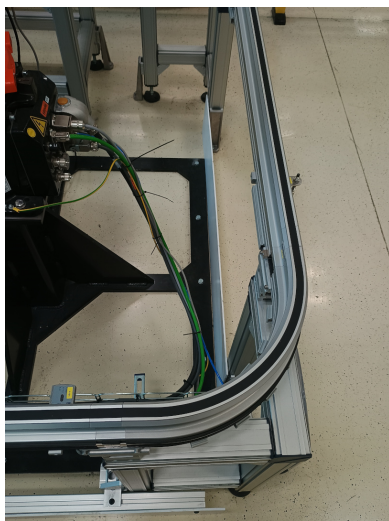
Table 3.2: Overview of crossings by type



(a) : Router



(b) : IRM Module



(c) : Curve



(d) : Physical station

Figure 3.3: Montrac modules 1

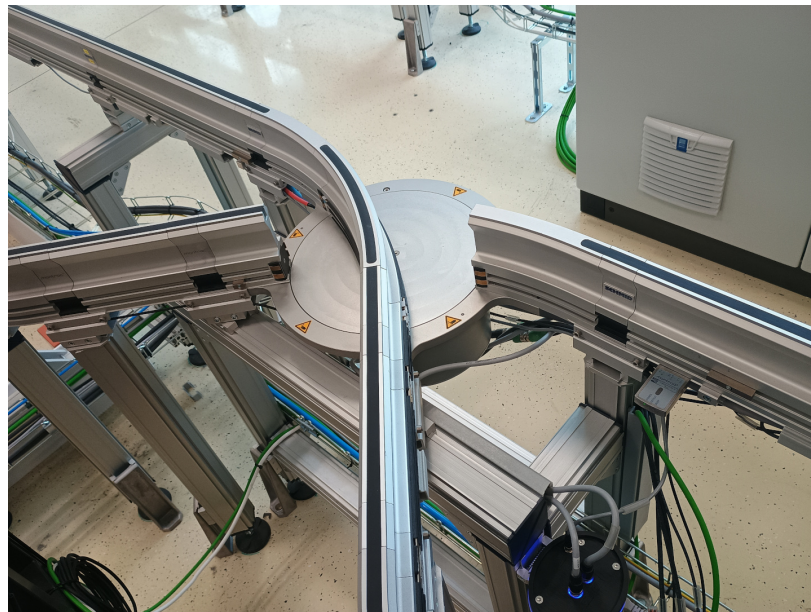
3. Workplace configuration



(a) : Crossing Collect



(b) : Crossing Divide



(c) : Crossing Arena

Figure 3.4: Montrac modules 2

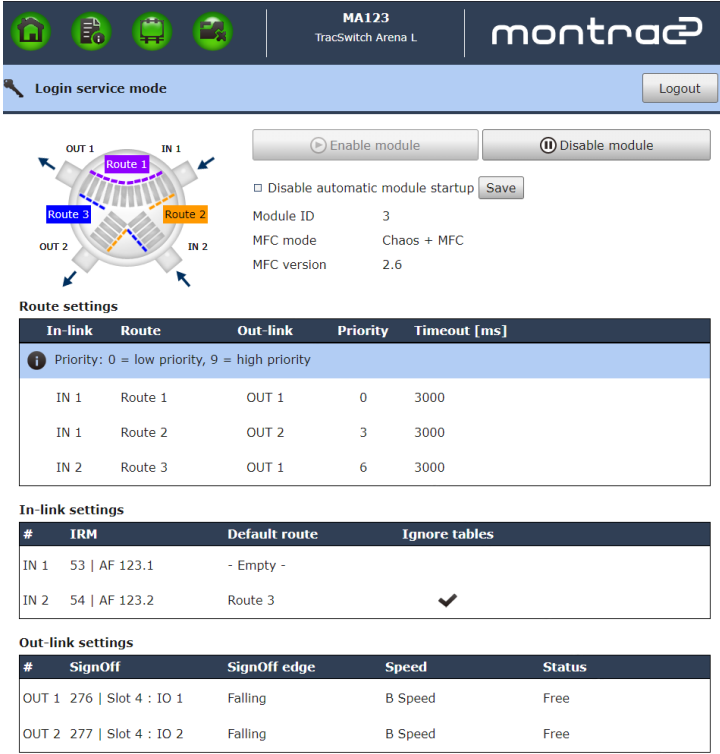
3.1.1 Routers

Before the control logic rework, new routers were installed. The new routers will be used in the rest of this thesis, but a short rundown of the old routers is in order. Those routers used UDP-based Montrac Data Acquisition and Control (MDAC) protocol for management such as configuration and sending commands to be performed. The routers were connected with each other via Controller Area Network (CAN) bus, which also connected modules to their routers. Each router also had a web server for easier configuration.

The new routers used PROFINET for connecting to the PLC and daisy-chain interconnection with other routers. This allowed for communication with each router from the PLC via library blocks provided by montratec and thus easier access to the router's data. Modules and signoff sensors were still connected to their routers using CAN bus.

3.1.2 Module configuration

As mentioned above, routers could be configured using their integrated web server. This was done following the montratec manuals [16]. The configuration needed was extensive and only an example can be seen in 3.5, 3.6, 3.7, and 3.8. The other modules followed the same principle.



Route settings

In-link	Route	Out-link	Priority	Timeout [ms]
i Priority: 0 = low priority, 9 = high priority				
IN 1	Route 1	OUT 1	0	3000
IN 1	Route 2	OUT 2	3	3000
IN 2	Route 3	OUT 1	6	3000

In-link settings

#	IRM	Default route	Ignore tables
IN 1	53 AF 123.1	- Empty -	
IN 2	54 AF 123.2	Route 3	✓

Out-link settings

#	SignOff	SignOff edge	Speed	Status
OUT 1	276 Slot 4 : IO 1	Falling	B Speed	Free
OUT 2	277 Slot 4 : IO 2	Falling	B Speed	Free

Figure 3.5: Configuration example – overview

Configuration available in 3.5 consisted of route settings, inlink settings, outlink settings, and MFC mode. Routes were defined by their name and the inlink and outlink they connected. Their configuration consisted of priority and timeout. Priority determined the order of routing in cases where multiple inlinks were occupied at the same time. In such cases, the shuttle waiting for a route with higher priority would be handled first. When a shuttle wanted to take a specific route but that route was blocked, it would wait for the duration of that route's timeout. If the route was still blocked, the shuttle would take the default route for its inlink or continue waiting for a route release if the default route could not be taken.

Inlinks showed which IRM they were connected to and could have their default route and table handling behavior set. Besides being taken after the timeout passes, the default route is always taken if the “Ignore Tables” toggle is set.

Outlinks show their corresponding signoff sensor and current occupancy status. They could also have their signoff edge behavior and departure speed configured. Signoff edge behavior determined whether the sensor would fire on a rising or falling edge.

Each route of a crossing had a control table and a chaos table. These were used for defining routing behavior, which is further described in a later section. Control tables could have up to 32 entries that matched the exact shuttle address or exact target address and would be discarded after successful routing. Chaos tables could have up to 16 entries that matched a range of shuttle addresses or a range of target addresses and would not get discarded. Both tables could have entries added at runtime, which was particularly useful for control tables considering their discarding nature. Figures 3.6, 3.7, and 3.8 show example configurations of chaos tables. Control tables had no entries for this crossing, but their configuration was almost identical to that of chaos tables.

The screenshot shows a configuration window for 'Route 1'. At the top, there are two buttons: 'chaos 1' and 'control 1'. Below them is a 'Selection' dropdown menu currently set to 'Route 1'. The main area contains a table with the following structure:

#	Shuttle ID: Start-End	Target Address: Start-End
1	0.0.0.0 - 0.0.0.0	100.0.0.2 - 100.0.0.255

Below the table is an 'Add new value' button. A message at the top of the table area states: 'Table doesn't refresh automatically. To refresh select choosen table again.' The table also has a dropdown menu for 'IDs as bytes'.

Figure 3.6: Configuration example – Route 1

chaos 1 control 1

Selection **Route 2**

Table doesn't refresh automatically. To refresh select chosen table again.

IDs as bytes

#	Shuttle ID: Start-End	Target Address: Start-End
1	0.0.0.0 - 0.0.0.0	100.0.0.1 - 100.0.0.1

Add new value

Figure 3.7: Configuration example – Route 2

chaos 1 control 1

Selection **Route 3** This table is currently not in use (ignore table flag is set)

Table doesn't refresh automatically. To refresh select chosen table again.

IDs as bytes

#	Shuttle ID: Start-End	Target Address: Start-End
1	0.0.0.0 - 0.0.0.0	100.0.0.2 - 100.0.0.255

Add new value

Figure 3.8: Configuration example – Route 3

3.1.3 Logic and routing

When a shuttle was idle in a station (it had arrived at its destination) a new target could be set. Shuttles would depart from stations after setting the corresponding start bit. Every time a shuttle entered an inlink of a crossing on its way, it would stop on the IRM and wait for the crossing's router to handle routing.

Routing consisted of three steps: checking routing tables, waiting for route change, and moving across the crossing. Table lookup was performed by first checking the control table for a shuttle address match, then checking the chaos table for a shuttle address match, then checking the control table for a target address match and finally checking the chaos table for a target address match. If a match was found at any point, the route for that inlink would be considered resolved. If no match was found and a default route was specified, the default route would be chosen, otherwise, the shuttle would wait (for a new entry in the tables). When all inlinks had their routes resolved, the route with the highest priority would be chosen to take place first. A routing was considered finished when the corresponding signoff sensor sent a signal that a shuttle had passed over it and thus left the crossing.

When shuttles departed from their inlinks, their speed was set to the speed configured on the corresponding outlink. Aside from that, shuttles could have their speed adjusted by passing over certain placements of metal blocks (called stones) attached to the track as described in 3.3. The tall stone would take up both rail slots and the long stone one rail slot. Shuttles would also decrease their speed as they approached objects, down to a full stop when

within 13 cm of them to avoid collisions.

Upper slot	Lower Slot	Resulting speed
Long	Empty	Full stop
Empty	Long	One higher unless maximum speed One lower otherwise
Long	Long	Same as Long in top slot
Tall	Tall	Medium speed

Table 3.3: Effect of track stones on shuttle speed

3.2 Robots

Each robot had its designated workspace as shown in 3.2 by dashed rectangles around them. Workplaces of robots R1, R2, R3, and R10 overlapped on neighboring stations. This is shown as a crossed area. Each robot had a tool changer attached to it and up to 8 tools placed on a holder in its workplace. This allowed for different types of operations to be performed by every robot without the need for a manual change of the mounted tool. The tool-changing system used was a pneumatic solution SWK 007 from SCHUNK. All robots also had to be connected to their controllers to supply them with power, provide low-level control, host an OPC-UA server, allow for remote programming and access over the network via various protocols such as PROFINET and EtherCAT, and facilitate several other functions, most of which were not used. As typical for most modern robot controllers, they also had a teach pendant which was used for on-site programming and control of robots without the need of a computer.

The robotic programs used at the time were based on OPC-UA and allowed for the starting of different operations by setting a desired program number and then switching data ready.

3.3 Control system

The previous implementation of the montrac control system used an external server for sending UDP packets to routers and the source code could not be recovered. Overall behavior could be described as MES sending the server a message saying which shuttle should go to which station. The server would then take care of sending the MDAC messages to routers. The upgrade to PROFINET-capable routers allowed for the merging of the control functionality into the PLC together with robot control and the rest of the functions used, such as power consumption monitoring and OPC-UA access.

■ 3.4 Communication

Different parts of the line used different communication media and protocols. The most important part was the OPC-UA communication between the MES and PLC. A MES schedules the execution of operations in a plant to increase the efficiency of production while observing the processes and devices to obtain real-time data. This means that MES has to know the status of shuttles, robots, and other resources. [17, 18, 19]

Altogether, communication in the line consists of:

- Customer to Enterprise Resource Planning (ERP) – Internet
- ERP to MES – Internet or intranet
- MES to PLC – OPC-UA
- PLC to Routers – MDAC over UDP, PROFINET after upgrade
- Routers to Modules – CAN

■ 3.5 Safety

The safety of workers and other people around the line was ensured by Light Detection And Ranging (LIDAR) scanners or light curtains in open areas and metal fences with safety doors in others. The controlling PLC had dedicated safety blocks for handling inputs from these sensors to prevent overwriting by unauthorized personnel and to ensure safe logic evaluation.

Chapter 4

Shuttle-oriented control

The base station-oriented control of the montrac line was impractical for the use cases of Testbed so a new shuttle-oriented control had to be designed. The digital image of the montrac line was divided into router objects, which along with their diagnostic and control information contained module objects. These module objects also contained their own diagnostic data and control bits. These data blocks were updated to reflect the status of the real line cyclically via PROFINET communication blocks provided by montratec. Visual representation of this structure can be seen in 4.1.

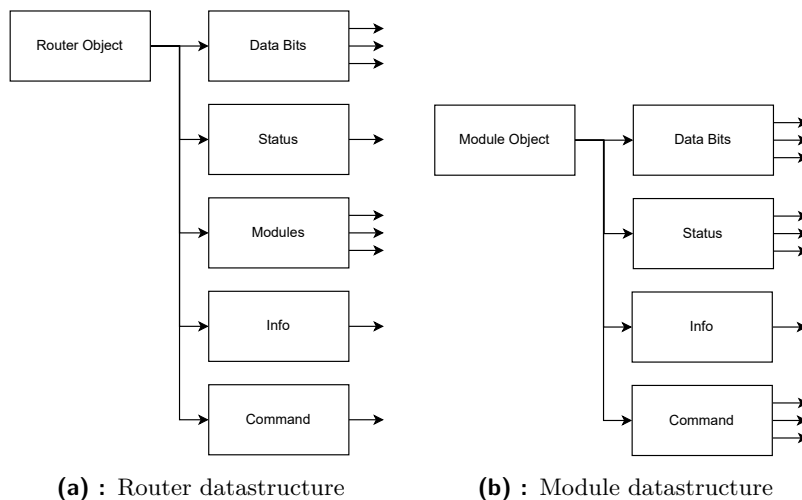


Figure 4.1: Montrac data structures

In each PLC cycle, all modules were iterated through after being updated. Shuttle data was updated during this process with information about where the shuttle was last seen and whether the shuttle was in a station or not. After updating shuttle data, the control logic was handled for each station. If there was a shuttle present and data ready edge was detected, the validity of input data was checked before sending the shuttle. This check consisted of detecting whether only the lock state was changed, whether the target station name was valid, and other miscellaneous checks regarding the system status.

The router objects were handled as instances of the “SRPN Router” Function Blocks (FBs) and individual modules as instances of the “SRPN Module SingleIRM” FBs. These library blocks were connected to the line Data Block (DB), where all router, module, and shuttle data was stored as well as to the hardware configuration via the tilde (~) notation. An example router FB can be seen in 4.2 and module FB in 4.4. The shuttles were modeled as a simple User Data Type (UDT) with inputs and outputs to fit the structure of their MES counterparts for easier integration. An example can be seen in 4.3.

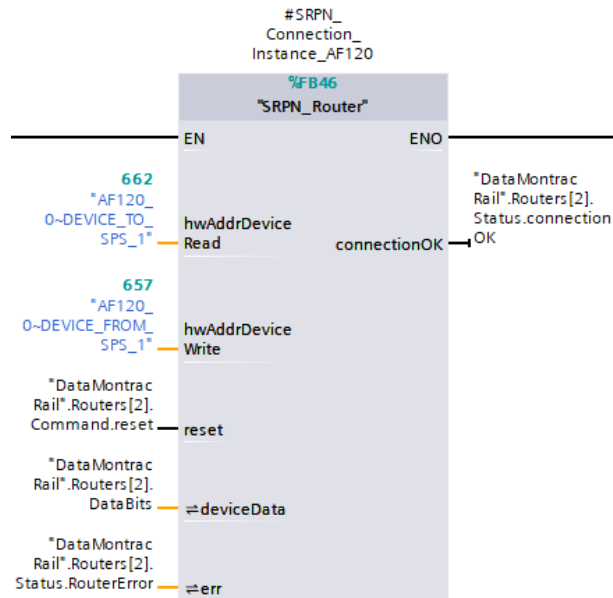


Figure 4.2: Example of a router Function Block

▼ TransportShuttle[1]	*TYPE_Montrac_Shuttle*
▼ TransportShuttleInput	*TYPE_Montrac_ShuttleInput*
DataReady	Bool
LockCommand	Bool
StationIdDestination	String
RoundTrip	Bool
▼ TransportShuttleOutput	*TYPE_Montrac_ShuttleOutput*
ShuttleInStation	Bool
StationIdCurrent	String
LockStatus	Bool
TargetStation	String
TargetWillLock	Bool
OperationStarted	Bool
OperationFinished	Bool
ResultNumber	Lint
ResultString	String
LastSeenAt	String

Figure 4.3: Example of a shuttle UDT

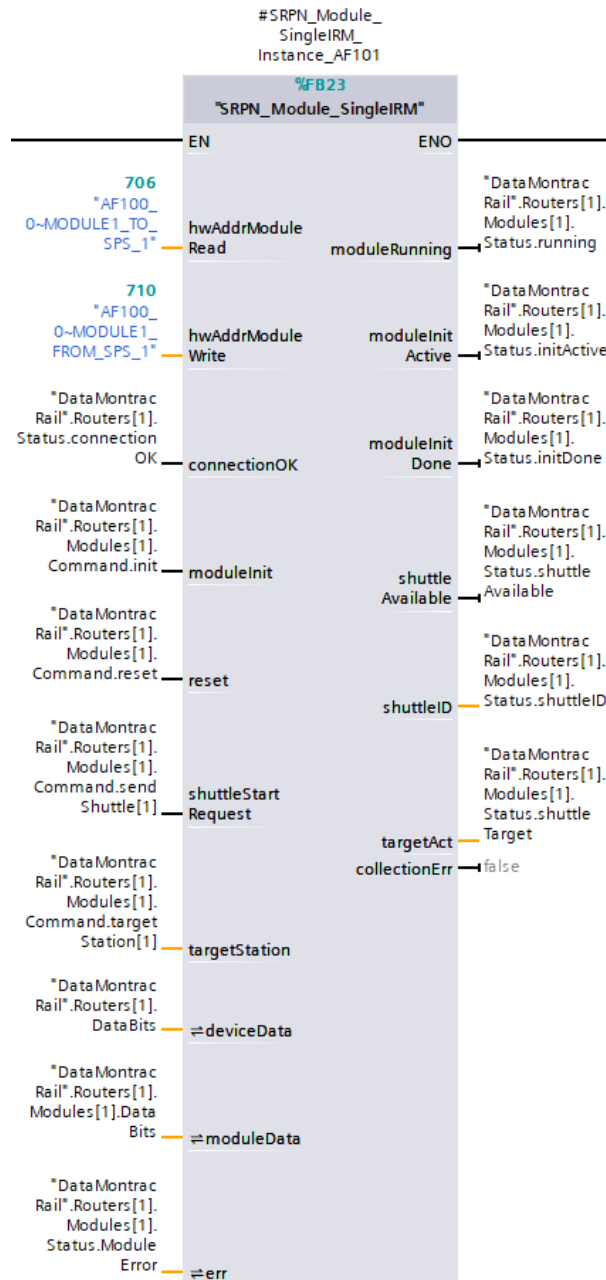


Figure 4.4: Example of a module Function Block



Chapter 5

Virtual Commissioning

As mentioned in the introduction, Virtual Commissioning of a line allows for the design, testing, and validation of software without the need for additional production downtime. This lowers deployment costs and on-site time requirements while also allowing for testing new processes and teaching operators without risking damaging the equipment.

Virtual Commissioning refers to both visually and functionally representative software simulation of the real line. Achieving this requires creating 3D models of all objects, adding kinematics to moving objects, simulating hardware behavior, control logic and Human-Machine Interfaces (HMIs), material flow, virtual robotic controllers, AGVs, and more. For this specific project, the virtual commissioning process was proposed as seen in 5.1 and had to encompass the following:

- Creating a 3D representation of the line (provided by Testbed)
- Creating kinematics of crossings
- Simulating montrac routing, station, and shuttle behavior
- Integrating control logic
- Simulating robots
- Simulating production material flow for an example process
- Integrating production requests

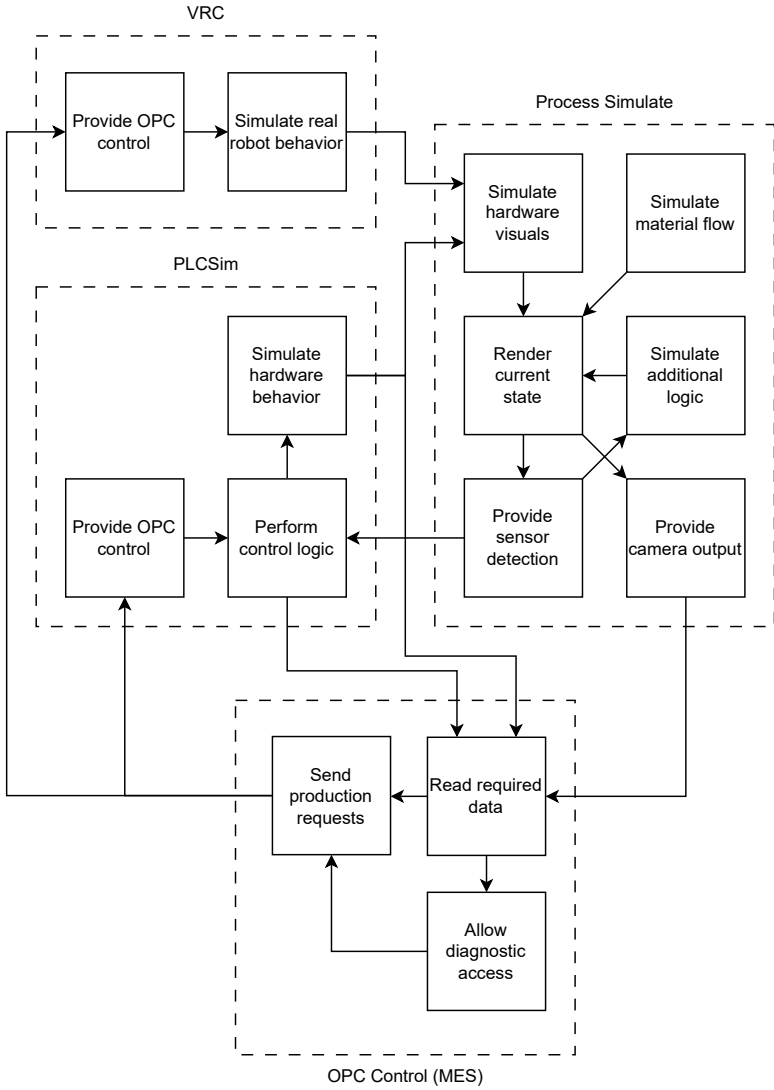


Figure 5.1: Diagram of proposed Virtual Commissioning

■ 5.1 Software tools used

■ 5.1.1 Process Simulate

Process Simulate is a simulation software for industrial automation and robotics shop floors by Siemens. It allows for simulating physical interactions between objects, processes such as welding, connecting with simulated PLCs, simulating material flow, viewing processes in virtual reality, and more. PS projects are called studies and have two modes of operation – standard and line simulation. Standard mode allows for easy simulation with predefined operation within PS and line simulation mode connects the study to a PLC and allows for event-based simulation.

The main benefits of Process Simulate include the capability to import custom 3D models in the JT format as components, adding Logic Blocks (LBs) and Structured Control Language (SCL) scripts to any resources present in the study, connecting to multiple Virtual Robot Controllers (VRCs) and PLCs at the same time and implementing custom features not found in Process Simulate itself via Microsoft .NET extensions.

■ 5.1.2 TIA Portal

Totally Integrated Automation (TIA) Portal is a software development suite that integrates the design of PLC code, hardware configuration, HMIs, and other features in one place. It supports coding in multiple languages like Ladder and SCL, real-time debugging, and much more. Additional information can be found in [20].

■ 5.1.3 PLCSim Advanced

PLCSim Advanced is a virtual PLC simulator by Siemens. It supports simulating multiple PLCs of S7-1500, S7-1500R/H, ET 200SP, or ET 200pro families with either local or Transmission Control Protocol/Internet Protocol (TCP/IP) access. Additionally, it allows for virtual time scaling, which is a very useful function for either precise debugging (lower scaling) or quick automated testing (higher scaling).

■ 5.1.4 KUKA WorkVisual

WorkVisual is an environment for managing, programming, and diagnosing KUKA robots. WorkVisual also supports non-KUKA kinematic systems to a certain degree and allows for communication via PROFINET, PROFIBUS, EtherCAT, EtherNet/IP, DeviceNet, and VARANBUS. [21]

5.1.5 KUKA OfficeLite on Microsoft Hyper-V

In order to use multiple KUKA.OfficeLite VRCs at the same time as required by the simulation, Microsoft Hyper-V hardware virtualization platform was used. This platform allows for virtualizing multiple operating systems, hard drives, switches, and other devices on one host system. The advantage of using KUKA.OfficeLite VRCs is that they behave the same way the real controllers do, so any programs created can be transferred straight to real robots after being tested virtually. [22, 23]

5.2 Implementation

5.2.1 Creating a Simulation Study

The base model for the assembly line was provided by Testbed, which made the process of creating the VC significantly shorter. The study contained the montrac conveyor system without kinematics, robot fixtures, robot tables, tool racks, and grippers without kinematics. The study can be seen in 5.2.

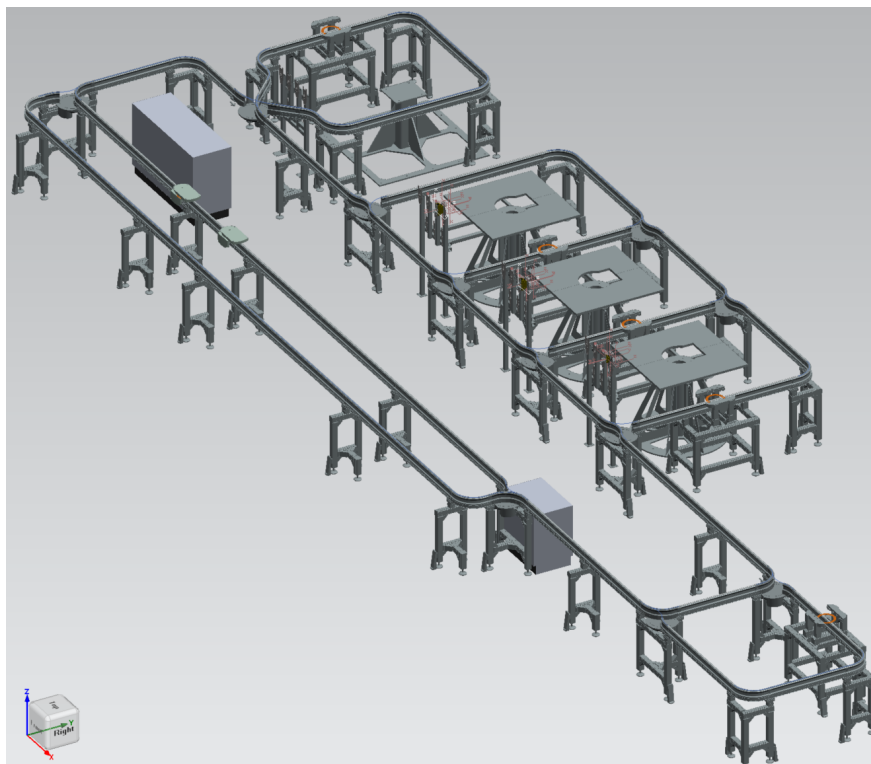


Figure 5.2: Base study

5.2.2 Adding kinematics to the virtual line

To visualize the process of crossings changing routes, kinematics had to be added to their models. This was done in PS via the kinematics control menu as seen in 5.3 and 5.4. To be able to change these routes from a PLC, a logic block was added to each crossing with input *selectRoute* and output *currentRoute*. The inner logic consists of move pose action, as seen in 5.5, and joint value sensor, as seen in 5.6, for each route possible. Since the crossing can only be in one position at a time and the joint angle distance between each position is sufficient for the joint value sensor to not falsely trigger, the actual route output can be easily calculated by multiplying each sensor output by a distinct value. This can be seen in 5.7. This approach allows for easy implementation of any routing setup possible.

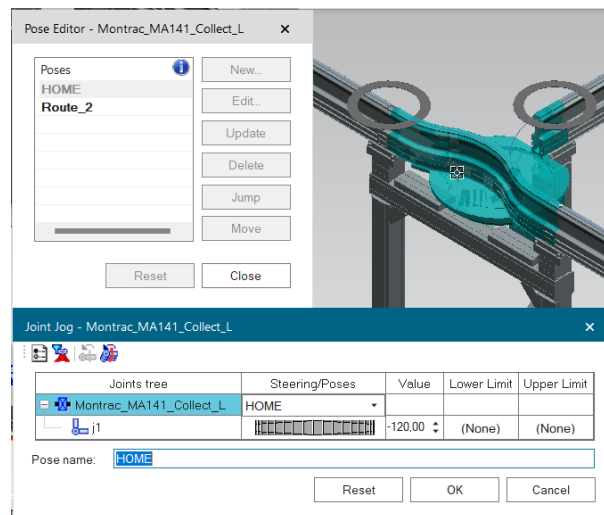


Figure 5.3: Setup of route 1 as HOME pose for crossing MA141

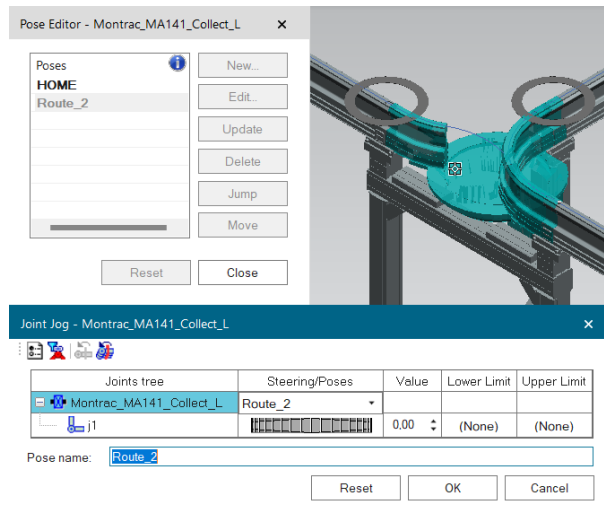


Figure 5.4: Setup of route 2 as additional pose for crossing MA141

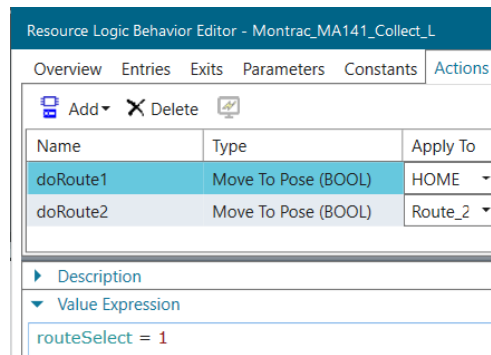


Figure 5.5: Route control actions for crossing MA141

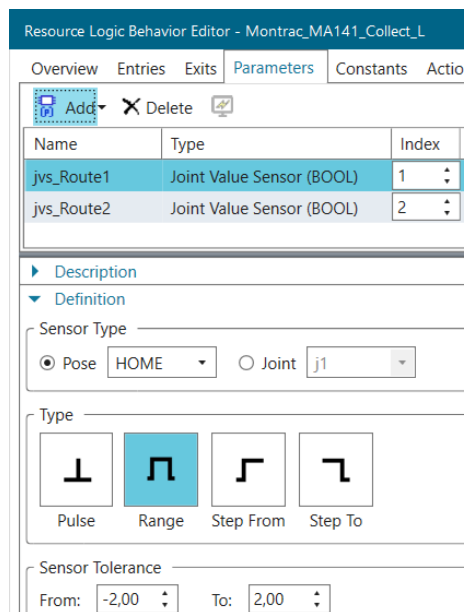


Figure 5.6: Joint value sensors for crossing MA141

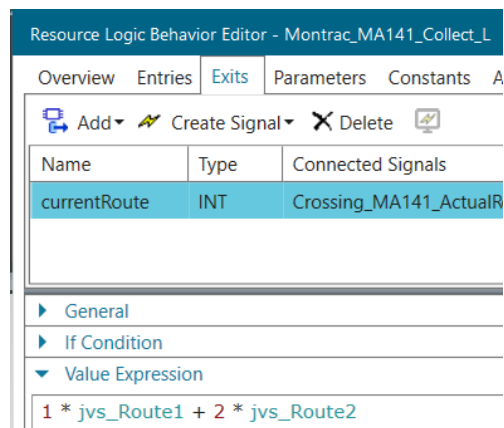


Figure 5.7: Calculation of current route for crossing MA141

5.2.3 Modeling shuttles as Autonomous Guided Vehicles

To create shuttles that can move on the conveyor rails, Autonomous Guided Vehicles (AGVs) were used. AGVs in PS can have custom 3D models as their visual representation and support kinematics and interactions with other objects, such as gripping in this case. The shuttle model with its gripping area, which is normally invisible during simulation, can be seen in 5.8.

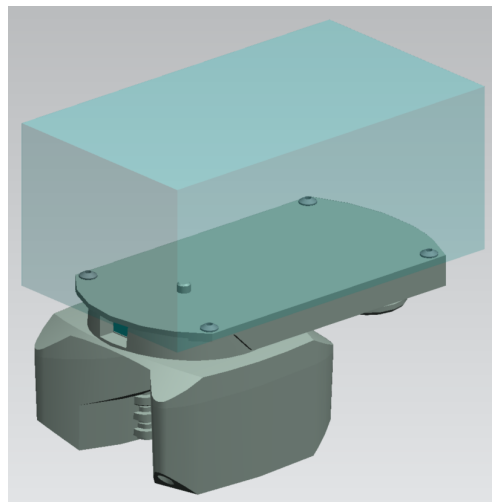


Figure 5.8: Model of the shuttle AGV with gripping area (light blue)

AGVs move between targets based on numeric identifiers. If an object is set as AGV carpet using the “Create AGV Carpet” button, AGVs only move on it when they can. This allows for limiting movement area as desired. The carpet in this study was modeled as wires laid on top of the montrac rail to make the shuttles move the same way they do on the real line. Two possible unintuitive behaviors of the AGV carpet were observed. First was that every time the Line Simulation mode was exited and re-entered, such as when quickly switching to Standard mode to change the part setup, the

carpet object was reset and had to be defined again. The second was that upon simulation start and whenever shuttles had their targets changed, they did not resume their routing on the carpet unless they were at a point where two individual wires connected and instead first returned to the closest such point. These had to be kept in mind while setting up the carpet and targets.

The control logic is represented by a Logic Block which gets automatically created when an object is designated as a AGV. The movement of AGVs is defined by their target, movement speed, motion planner type, path planner type, rotation method and control signals such as emergency stop and auto acquire. Each AGV also has an ID and provides several outputs, for example, the ID of the currently acquired target, actual speed, and whether the AGV is busy. The base AGV object had itsLB interface extended to allow for signal-based gripping of objects above it and additionally defined as a gripper to allow for this behavior. This extended interface can be seen in 5.9.

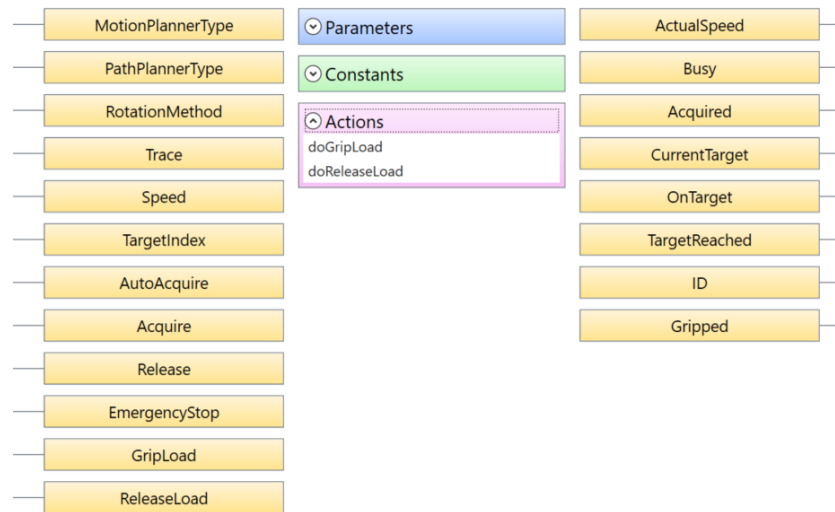
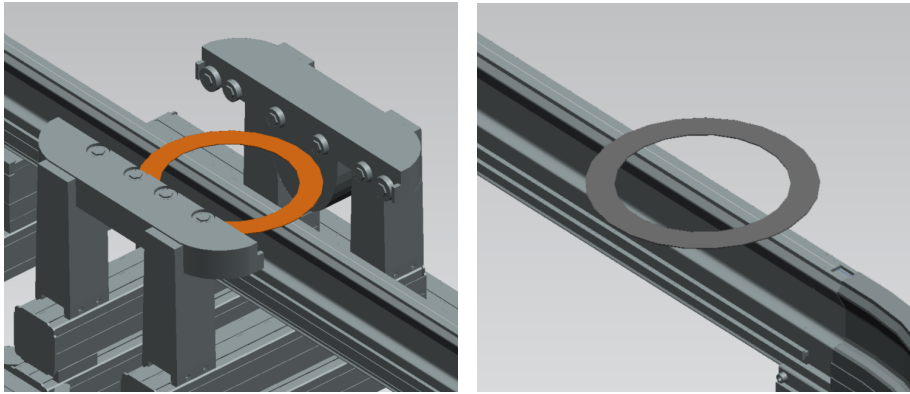


Figure 5.9: Extended interface of a shuttle

Each station is represented by one target and crossings have a target on each of their inlinks. These targets were color-coded for easier recognition at first glance. Stations used orange circular targets as seen in 5.10a and all other targets were grey circles, as seen in 5.10b. Since shuttles can only move in one direction on the real line and AGVs move between two targets via the shortest path available, additional targets to enforce the travel direction were added. For easier recognition at first glance, targets had a numbering system implemented. Stations have their target IDs represented with up to 3 digits the same way they are numbered. Crossing inlink targets take the 3-digit crossing identifier and add a digit in front of it based on the inlink ID. For example, inlink 1 of crossing MA123 has identifier 1123 and inlink 2 of crossing MA146 has identifier 2146. Targets added to enforce travel direction have 4-digit identifiers starting with the number 9. Having customized IDs also meant that each target had to be saved as a new component because multiple instances of a component share the same values for constants.

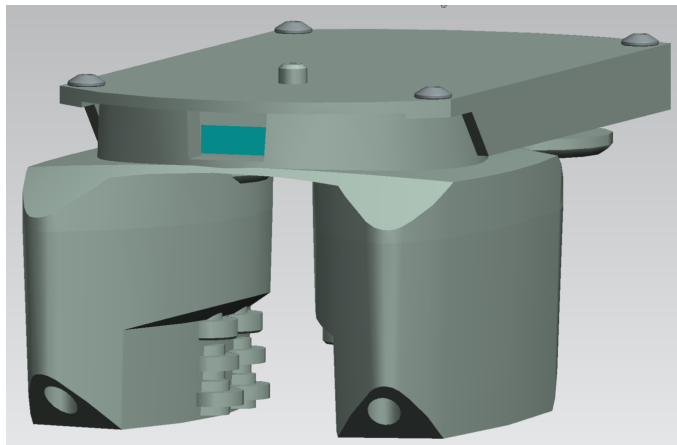


(a) : Example of a station target

(b) : Example of a generic target

Figure 5.10: AGV targets

The shuttle slowdown behavior when approaching an object was simplified only to the full stop when within 13 cm. This was implemented by attaching a proximity sensor to the front of each shuttle and appropriately configuring the sensing range and detection list. This sensor can be seen in 5.11. Attention should be paid to how proximity sensors work in PS because they sense in all directions and thus detect objects even to the sides and behind. This was not a major problem in this case because the rails were always far enough from each other but the stops on crossings had to be put far enough.

**Figure 5.11:** Sensor body (blue) on a shuttle

5.2.4 Adding kinematics to robots and grippers

When robots were added to the study, their 3D models were downloaded from the KUKA Download Center with already defined kinematics [24]. The grippers present in the study did not have kinematics yet and had to be implemented manually. This was done using the kinematics editor in PS by first defining individual parts of the gripper as separate links connected by prismatic joints as seen in 5.12 and then defining an “Open” and “Closed” poses as seen in 5.13.

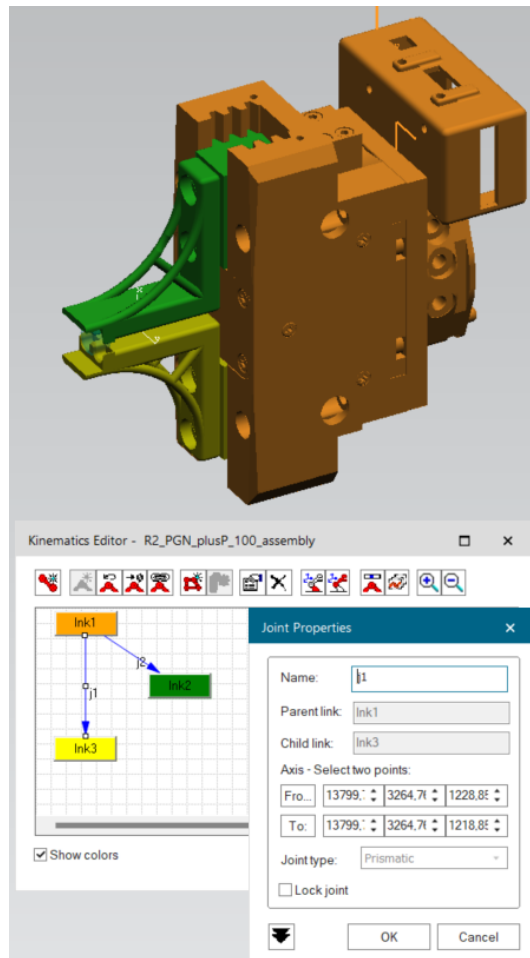


Figure 5.12: Gripper links definition

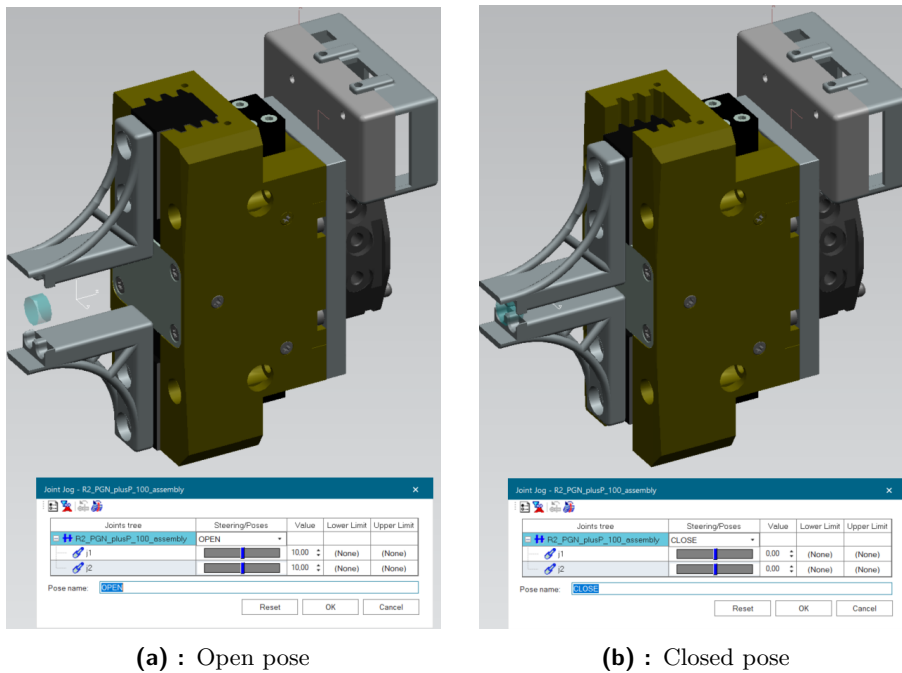


Figure 5.13: Gripper poses

5.2.5 Simulating conveyor components

In the process of simulating montrac hardware, modules were first generalized into crossings and stations. The behavior of these module types was then described with a state diagram to make implementation easier. The diagram for crossings can be seen in 5.14 and for stations in 5.15. Both diagrams were then converted into Finite-State Machines (FSMs) and implemented in the PLC. All modules also had to be parameterized to reflect the real setup.

When converting the crossing behavior into an FSM, the stage of route resolving was the most complex to implement. This was because routing tables had to be searched through correctly and handling results was also a multi-step process. Implementing table searching to be efficient was crucial as this process looped through and compared hundreds of entries and could cause a significant slowdown of the simulation when scaled to the whole line. This was done by tightly following the real hardware behavior described in 3.1.3, specifically considering routing resolved as soon as a match is found and stopping the search. Invalid table entries were also made to be identified before the data comparison started to avoid wasting computation resources.

Stations also used route resolving when deciding whether the shuttle that just arrived there should stop or be sent off again.

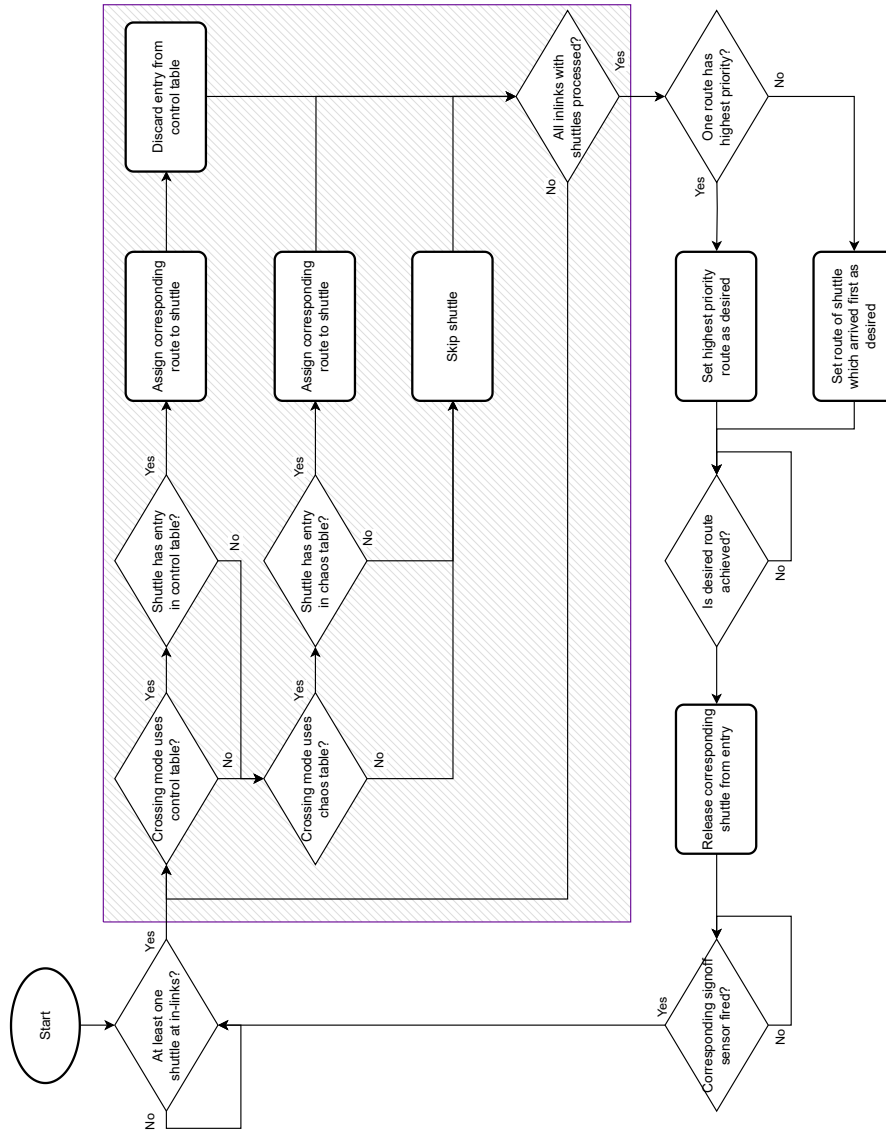


Figure 5.14: Diagram of physical crossing behavior

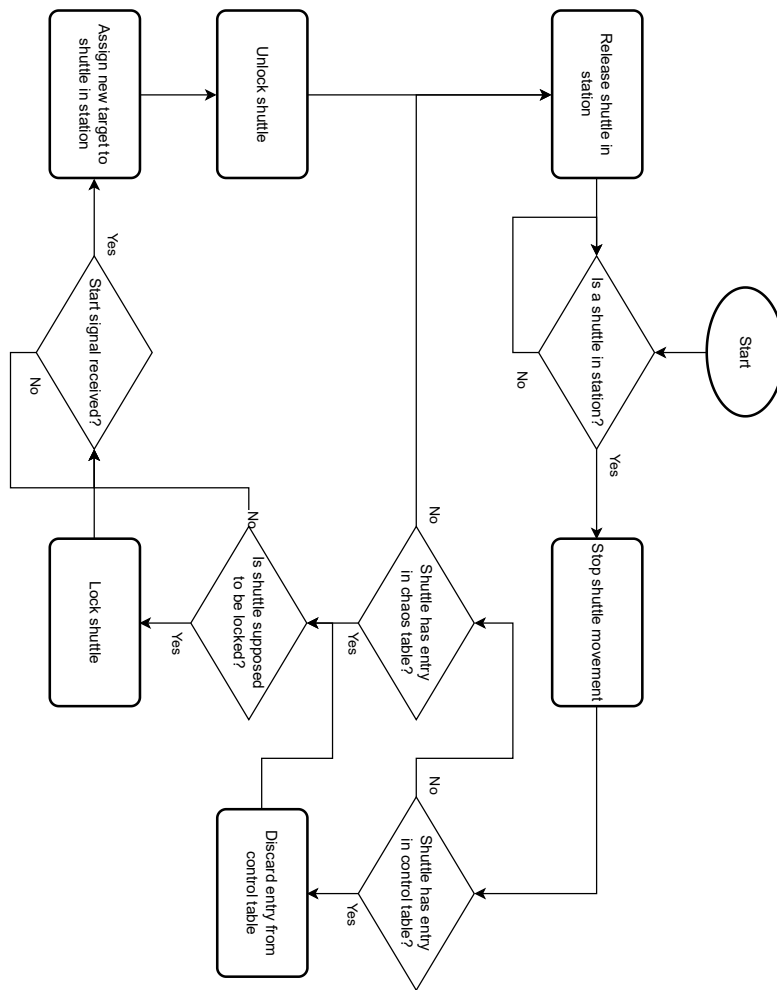


Figure 5.15: Diagram of physical station behavior

5.2.6 Connecting Process Simulate with a virtual PLC

PS study and the simulated PLC are both separate environments that had to be connected. This was done using the tag system and a self-made Python script for automatically generating both sides of this connection quickly.

In the PLC, tags are specified in tables and assigned addresses. An example can be seen in 5.16. These tags can then be used in the program and either read from or written to. It is common not to work directly with the tags but to rewrite their content to user DBs and work with those. Tag definitions can be imported from a specially formatted Excel workbook.

On the PS side, tags are connected to the inputs and outputs of resources. These tags can then be controlled from within PS or connected to a PLC. When connecting tags to a PLC a connection has to first be set up and then each tag has to be set to “PLC Connected”, assigned a connection and an address corresponding to the one in the PLC. An example of these tags can

Name	Tag table	Data type	Address	Retain
Shuttle1_Trace	PS_TO	Bool	%Q1000.0	<input type="checkbox"/>
Shuttle1_TargetIndex	PS_TO	Int	%QW1001	<input type="checkbox"/>
Shuttle1_Speed	PS_TO	Real	%QD1003	<input type="checkbox"/>
Shuttle1_Sensor	PS_FROM	Bool	%I1000.3	<input type="checkbox"/>
Shuttle1_RoundTrip	PS_FROM	Bool	%I1000.4	<input type="checkbox"/>
Shuttle1_PauseShuttle	PS_TO	Bool	%Q1000.2	<input type="checkbox"/>
Shuttle1_OnTarget	PS_FROM	Bool	%I1000.2	<input type="checkbox"/>
Shuttle1_ManualTarget	PS_FROM	DWord	%ID1003	<input type="checkbox"/>
Shuttle1_LockTarget	PS_FROM	Bool	%I1000.1	<input type="checkbox"/>
Shuttle1_DataReady	PS_FROM	Bool	%I1000.0	<input type="checkbox"/>
Shuttle1_AutoAcquire	PS_TO	Bool	%Q1000.1	<input type="checkbox"/>
Shuttle1_Acquired	PS_FROM	Int	%IW1001	<input type="checkbox"/>

Figure 5.16: An example section of the PLC tag table

be seen in 5.17. Tag definitions can also be imported from an Excel workbook with a different format.

Signal Name	Type	Robot Signal Name	Address	IEC Format	PLC Connection	External Connect	Resource
shuttle_montrac_1_Acquire	INT		No Address	Q	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_Acquired	INT		1001	I1001	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_ActualSpeed	REAL		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_AutoAcquire	BOOL		1000.1	Q1000.1	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_Busy	BOOL		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_CurrentTarget	INT		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_DataReady	BOOL		1000.0	I1000.0	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_GripLoad	BOOL		2000.3	Q2000.3	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_Gripped	BOOL		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_ID	INT		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_ManualTarget	DWORD		1003	I1003	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_MotionPlannerType	INT		No Address	Q	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_OnTarget	BOOL		1000.2	I1000.2	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_PathPlannerType	INT		No Address	Q	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_PauseShuttle	BOOL		1000.2	Q1000.2	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_Release	INT		No Address	Q	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_ReleaseLoad	BOOL		2000.4	Q2000.4	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_RotationMethod	INT		No Address	Q	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_RoundTrip	BOOL		1000.4	I1000.4	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_Sensor	BOOL		1000.3	I1000.3	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1_Sensor
shuttle_montrac_1_Speed	REAL		1003	Q1003	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_TargetIndex	INT		1001	Q1001	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1
shuttle_montrac_1_TargetReached	BOOL		No Address	I	<input type="checkbox"/>		shuttle_montrac_1
shuttle_montrac_1_Trace	BOOL		1000.0	Q1000.0	<input checked="" type="checkbox"/>	MontracTwin	shuttle_montrac_1

Figure 5.17: An example section of the PS tags

Setting up the tags and DB updates is simple but can be time-consuming with a large number of tags and mistakes can happen. A common mistake is using overlapping addresses for tags with larger data types which can lead to unexpected behavior. To avoid these issues, a Python script was written. This script takes shuttle count, crossing name list, station name list, and passthrough waypoint name list as inputs and automatically generates all 3 parts of the connection – the TIA portal tag Excel workbook (ex. 5.18), the PS Excel workbook (ex. 5.19), and TIA portal DB assignment SCL code.

Name	Path	Data Type	Logical Address	Comment	Hmi Visible	Hmi Accessible	Hmi Writeable	Typeobject ID	Version ID
Shuttle1_Trace	PS_TO	Bool	%Q1000.0		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_AutoAcquire	PS_TO	Bool	%Q1000.1		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_PauseShuttle	PS_TO	Bool	%Q1000.2		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_TargetIndex	PS_TO	Int	%QW1001		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_Speed	PS_TO	Real	%QD1003		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_DataReady	PS_FROM	Bool	%I1000.0		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_LockTarget	PS_FROM	Bool	%I1000.1		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_OnTarget	PS_FROM	Bool	%I1000.2		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_Sensor	PS_FROM	Bool	%I1000.3		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_RoundTrip	PS_FROM	Bool	%I1000.4		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_Acquired	PS_FROM	Int	%IW1001		PRAVDA	PRAVDA	PRAVDA		
Shuttle1_ManualTarget	PS_FROM	DWord	%ID1003		PRAVDA	PRAVDA	PRAVDA		

Figure 5.18: Example section of the generated TIA workbook

Signal Name	Electrical Name	Internal Name	Function	Type	Address	External Connection	PLC Connection	Comment
shuttle_montrac_1_Trace	shuttle_montrac_1_Trace			BOOL	Q1000.0	MontracTwin		PRAVDA
shuttle_montrac_1_AutoAcquire	shuttle_montrac_1_AutoAcquire			BOOL	Q1000.1	MontracTwin		PRAVDA
shuttle_montrac_1_PauseShuttle	shuttle_montrac_1_PauseShuttle			BOOL	Q1000.2	MontracTwin		PRAVDA
shuttle_montrac_1_TargetIndex	shuttle_montrac_1_TargetIndex			INT	Q1001	MontracTwin		PRAVDA
shuttle_montrac_1_Speed	shuttle_montrac_1_Speed			REAL	Q1003	MontracTwin		PRAVDA
shuttle_montrac_1_DataReady	shuttle_montrac_1_DataReady			BOOL	I1000.0	MontracTwin		PRAVDA
shuttle_montrac_1_LockTarget	shuttle_montrac_1_LockTarget			BOOL	I1000.1	MontracTwin		PRAVDA
shuttle_montrac_1_OnTarget	shuttle_montrac_1_OnTarget			BOOL	I1000.2	MontracTwin		PRAVDA
shuttle_montrac_1_Sensor	shuttle_montrac_1_Sensor			BOOL	I1000.3	MontracTwin		PRAVDA
shuttle_montrac_1_RoundTrip	shuttle_montrac_1_RoundTrip			BOOL	I1000.4	MontracTwin		PRAVDA
shuttle_montrac_1_Acquired	shuttle_montrac_1_Acquired			INT	I1001	MontracTwin		PRAVDA
shuttle_montrac_1_ManualTarget	shuttle_montrac_1_ManualTarget			DWORD	I1003	MontracTwin		PRAVDA

Figure 5.19: Example section of the generated PS workbook

The script allowed for generating multiple tag targets at the same time. In this case Process Simulate and TIA Portal. After defining tag templates and file headers, the individual tags could be generated in loops by using indexes or looping through lists. A pseudocode example for generating shuttle tags can be seen in 5.1.

```

1 # AddSw takes swType, typeStr, addrOffset, ioDest and ioFormat
2 typeBool = TagType()
3 typeBool.AddSw(SWType.TIA, "Bool", 0.1, "", r"%0}{1}{2:.1f}")
4 typeBool.AddSw(SWType.PS, "BOOL", 0.1, "", r"{0}{1}{2:.1f}")
5
6 typeInt = TagType()
7 typeInt.AddSw(SWType.TIA, "Int", 2, "W")
8 typeInt.AddSw(SWType.PS, "INT", 2, "", r"{0}{1}{2:d}")
9
10 typeReal = TagType()
11 typeReal.AddSw(SWType.TIA, "Real", 4, "D")
12 typeReal.AddSw(SWType.PS, "REAL", 4, "", r"{0}{1}{2:d}")
13
14 typeDword = TagType()
15 typeDword.AddSw(SWType.TIA, "DWord", 4, "D")
16 typeDword.AddSw(SWType.PS, "DWORD", 4, "", r"{0}{1}{2:d}")
17
18 addrI = 1000
19 addrQ = 1000
20
21 tags = [
22     [typeBool, "Trace", "Q"],
23     [typeBool, "AutoAcquire", "Q"],
24     [typeBool, "PauseShuttle", "Q"],
25     [typeInt, "TargetIndex", "Q"],
26     [typeReal, "Speed", "Q"],
27
28     [typeBool, "DataReady", "I"],
29     [typeBool, "LockTarget", "I"],
30     [typeBool, "OnTarget", "I"],
31     [typeBool, "Sensor", "I"],
32     [typeBool, "RoundTrip", "I"],
33     [typeInt, "Acquired", "I"],
34     [typeDword, "ManualTarget", "I"],
35 ]
36
37 for shuttleId in range(1, 7):
38     for tag in tags:
39         ... # Perform assignments as defined by user

```

Listing 5.1: Part of tag autogeneration pseudocode

5.2.7 PLC code implementation

As mentioned in sections 5.2.5 and 5.2.6, the PLC had to simulate the mon-trac hardware and exchange data with Process Simulate. Besides these functions, the PLC also had to communicate with MES via OPC-UA, provide utility functions that the hardware simulation built on top of, and facilitate unit testing. These utility functions were mostly simple lookups of objects based on various parameters, such as *GetShuttleIndexByAddr*, *GetStationAddrByName*, or *GetStationIndexByPSID*. An exception was the *SearchCrossingTables* function which was a composite function implementing the crossing table lookup behaviour for routing inbound shuttles. This used four sub-functions for searching the Chaos and Control tables for shuttle ID and target addresses respectively. This modularity allowed for the re-use of the table lookup functions in stations, where the behavior was much simpler than in crossings but used the same principles.

The code consisted of the setup part, as seen in 5.20, and the main loop, as seen in 5.21. The MES OPC-UA access was solved by creating an interface to the required parameters in Data Blocks.

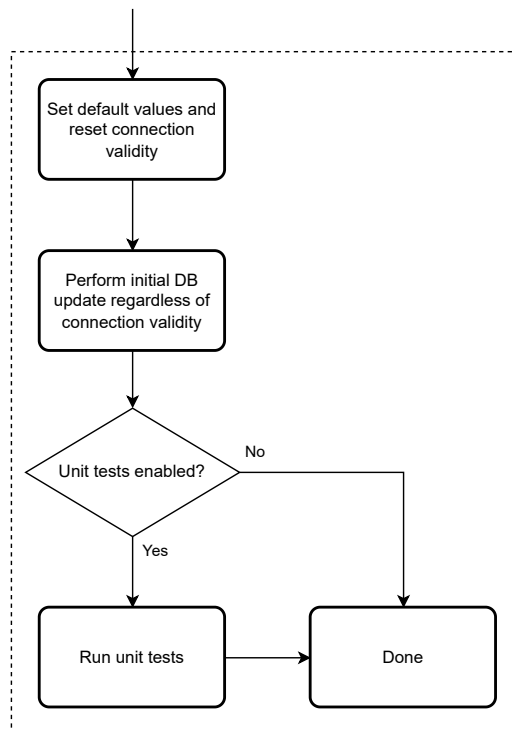


Figure 5.20: Diagram of PLC setup phase

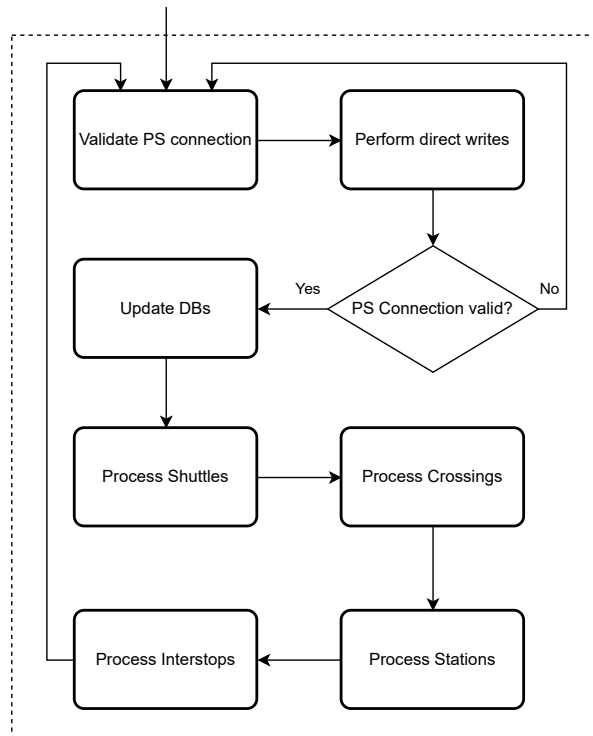


Figure 5.21: Diagram of the main PLC loop

■ 5.2.8 Basic robotic operations

The task to be simulated was dynamically taking apart a battery based on the color of slotted modules to simulate their state. A green module represented a re-usable part without damage and a red module represented a part that was somehow damaged and not fit for immediate use. An example of a real testing battery can be seen in 5.22 and the virtual counterpart (in a different configuration) in 5.23. After the battery was initially loaded onto a shuttle in the manually operated station S200, the battery lid was removed in station S23 using robot R3. Next, the battery was sent under the detection camera at S300, where a photo of the battery was taken and module recognition was performed. After that, the battery was sent to station S12 for module disassembly by robots R1 and R2, finishing the process.

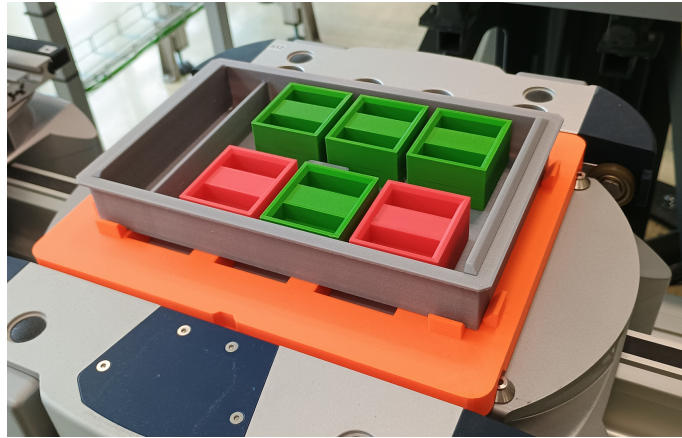


Figure 5.22: Example testing battery – reality

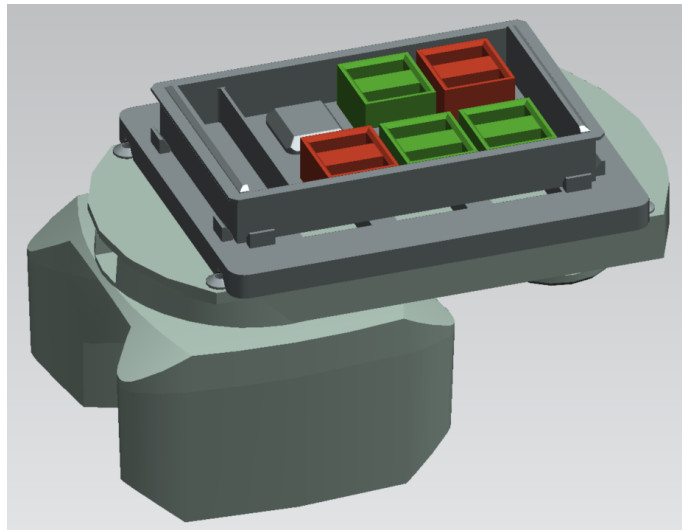


Figure 5.23: Example testing battery – simulation

The first step in creating robotic programs was testing basic robotic operations in PS using the standard simulation mode and internal robot controllers called Motion Planner (MOP). These operations were:

- Mounting a tool from the rack
- Unmounting a tool into the rack
- Picking a module from the battery
- Placing a module onto the tray

Every operation was divided into the following locations: Home, Approach, PreAct, Act, PostAct, Depart, and End. These locations were chosen in order to make the motion smooth and safe. All four operations had Offline Programming (OLP) commands assigned to the Act location. These commands are specific for each robot controller and most of the commands available

in PS are only for the purpose of simulating robot actions using the MOP controller. Each location had several parameters associated with it. The ones used in this thesis are as follows:

- Robot configuration – Consisted of “Status” and “Turn” to define in which configuration the robot should arrive at the location
- Tool Nr – What tool offset should be used when determining the location
- Reference base – In what base should the coordinates be saved (adds the ability to calibrate parts of trajectories)
- Motion type – Point to Point (PTP), Linear (LIN), and others. PTP is the most efficient and usually fastest motion type possible but does not guarantee what trajectory the robot will take. LIN is less efficient but guarantees the path to be a linear interpolation of the two locations and can have the speed specified in meters per second.
- Speed – What percentage of maximum speed or what exact speed should the robot move to this location at
- Zone – How far the robot can be from the exact location while still counting as reaching it, specified as a radius of a sphere around the location.

The Tool Nr and Reference Base options required defining tools and bases for the robot. This was done using the “Base and Tool Setup” in the “Robot Setup” menu as seen in 5.24.

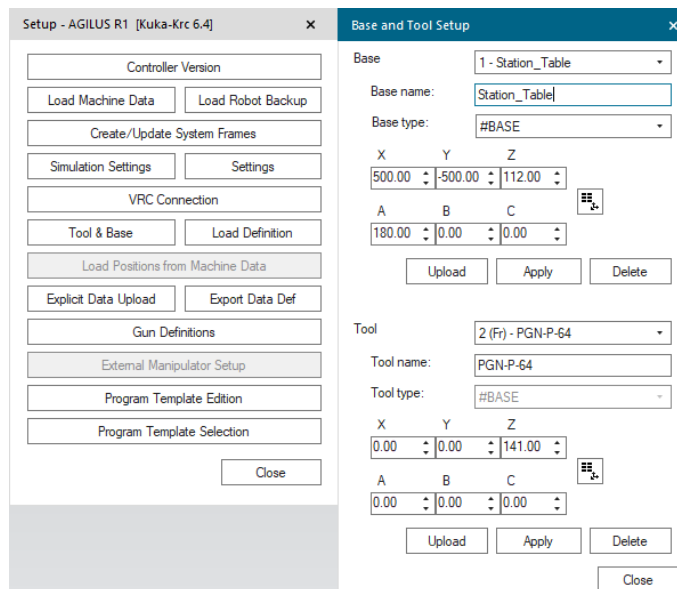


Figure 5.24: Robot setup example

An example operation flow can be seen in 5.25. This operation represents picking a module from the battery after a shuttle with the lid removed has

arrived at the station. The Start and End operations represent the home position of the robot. Approach and Depart are located 13 cm above the primary Act location for safe traversal of the robotic workspace. Having approached above the target, PreAct and PostAct locations allow for saving time by moving at a greater speed than what is safe for finally picking the target at the Act location.

Paths & Locations	Config	Acc	Tool Nr	Base Nr	Motion	Speed	Zone	OLP Commands
PICK_MODULE_S12								
Start	S 2 T 43	100 % (def)	2 (Fr) - PGN-P-64	1 - Station_Table	PTP	100 %	C_DIS 5mm	
Approach		100 % (def)	2 (Fr) - PGN-P-64	5 - Battery_S12	PTP	100 %	C_DIS 5mm	
PreAct		100 % (def)	2 (Fr) - PGN-P-64	5 - Battery_S12	LIN	0.2 m/s	FINE	
Act		100 % (def)	2 (Fr) - PGN-P-64	5 - Battery_S12	LIN	0.05 m/s	FINE	WAIT SEC 2
PostAct		100 % (def)	2 (Fr) - PGN-P-64	5 - Battery_S12	LIN	0.05 m/s	FINE	
Depart		100 % (def)	2 (Fr) - PGN-P-64	5 - Battery_S12	LIN	0.2 m/s	C_DIS 5mm	
End	S 2 T 43	100 % (def)	2 (Fr) - PGN-P-64	1 - Station_Table	PTP	100 %	C_DIS 5mm	

Figure 5.25: Example robot operation – pick from battery

The Accuracy and Tool Nr options were explicitly set to be identical in all locations. The Config option was only set explicitly on the Start and End locations, being implicitly defined for the rest. This was possible because the simulation correctly processed robot motion with joint configurations in mind, making accidental re-configuration during the operation observable when testing. The Base Nr option was set to Station Table for the Start and End operations and Battery_S12 for the rest. This allowed for separate calibration of the home position and battery position bases in reality.

Motion, Speed, and Zone options were the most important for the correct execution of the planned path, and extra attention was paid to their design. When moving in a space where there could be objects and robot movement had to be deterministic, the LIN motion was used. In free space, the faster but non-deterministic PTP motion was used. Specifically in this operation, the robot moved to Start and Approach using PTP, then through PreAct to Lin and back through PostAct to Depart using LIN and finally back home to End using PTP again. The distance traversed using PTP and speed during the whole operation were maximized without compromising safety to make the final process more time-efficient. The Zone option also contributed to the time saved by essentially allowing the robot to “eyeball” the movement and only get within 5 mm of the Start, Approach, Depart, and End locations before continuing to the next. This option can cause unintended behavior, especially in cases where there are multiple co-linear locations, and should be tested before implementing it.

After picking a module, the robots would continue with the Place operation and put modules on the trays in their workspaces.

■ 5.2.9 Implementing VRCs

The first step to implement Virtual Robot Controllers (VRCs) was preparing the environment. This meant setting up three virtual machines to run the controllers, adding sensors to facilitate the behavior of simulation OLP commands, and creating interfaces to control the VRCs. The virtual machines running virtual robotic controllers were set up in Hyper-V by doing the installs and generic configuration on one instance and then cloning it two times before proceeding with station-specific configuration.

With the transition from MOP controllers to VRCs, simulation OLP commands no longer worked. This meant that gripping, releasing, mounting, and unmounting had to be done programmatically via signals. Gripping and releasing functionality was already present in PS using Logic Blocks added to the grippers themselves. The mounting and unmounting functionality was not available during Line Operation runtime and had to be implemented externally as further discussed in section 5.2.10. The triggers for mounting and unmounting tools were added as direct write to robotic signals in the KUKA Robot Language (KRL) scripts, whereas the gripper triggers were implemented via proximity sensors in PS. These sensors for gripping and releasing were located in the gripper heads and placement trays respectively.

These mounting signals could be seen as an interface from VRCs to PS. In order to control the robots and thus create an interface from the outside, the VRC OPC-UA servers were used. The robot operations developed earlier were exported from PS and downloaded to the VRCs as individual programs using KUKA WorkVisual. To allow the starting of these programs via OPC-UA in a controlled manner, a new script was written in KRL. This script waited for a rising edge in *ProgramReady* and then started the corresponding program based on the provided *ProgramNumber*, waited for it to finish and unless an error had occurred went back to waiting for *ProgramReady* again. This behavior can be seen in diagram 5.26. The .dat files of the main script and programs that required runtime updates were exposed to the OPC-UA interface by setting them as “Public” in their headers.

The tool mounting and unmounting programs did not require runtime updates in this case since each robot only had one tool available. Although even if they had more than one tool, using separate programs could still be favorable as each tool-robot set had a dedicated mounting signal. This would result in the need for dynamically changing which output signals are written to and such overhead would not outweigh the simplicity of adding new programs for each tool. In contrast, the programs for picking and placing battery modules were implemented as parametrized instances because there are many possible locations for modules, and managing tens of programs would not be feasible.

After downloading a program to the robot, the process of integrating it with the main script was the following. The program was added to a new program selector branch with a (unique) program number as seen in 5.27.

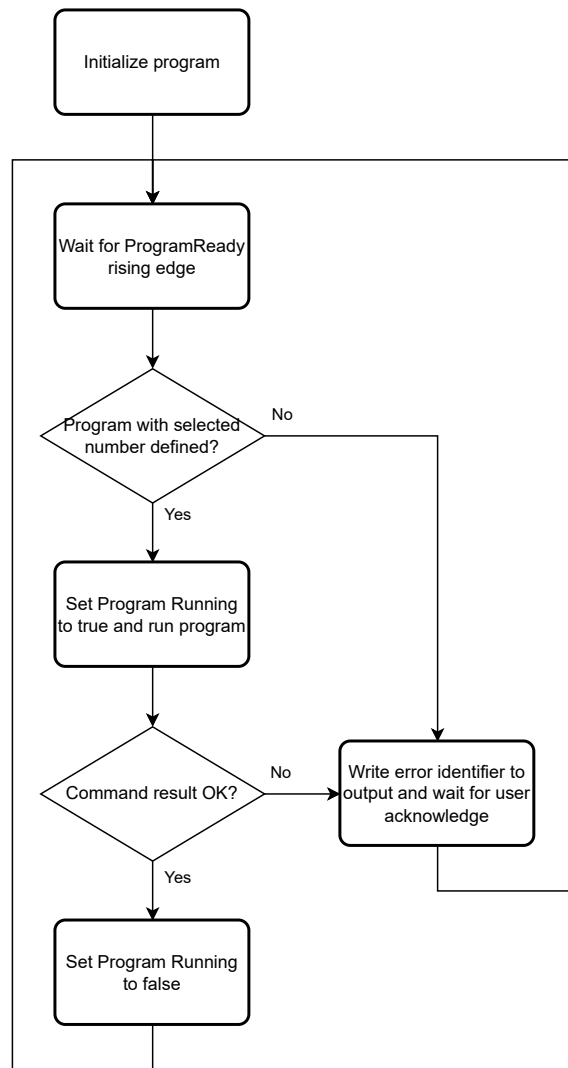


Figure 5.26: Diagram of a robot program cycle

The program's .dat file was set to public as required and if that included OPC-UA access to any position variables, those had to be changed from the *E6POS* type to the *POS* type because there was a bug in the KUKA OPC-UA implementation at the time. This bug caused the *A* parameter of *E6POS* type to not be writable from OPC-UA and default to zero. Additional commands for robot signals or other OLP commands were added to the .src file as necessary. For the programs used in these robots, only the tool grip and release robot output signals were toggled in the corresponding places.


```

LOOP
  respCommandResult = 0

  WaitRE(reqCommandReady)
  SWITCH reqCommandNumber
    CASE 1 ; Go home
      SetCmdStart()

      PTP_POS_HOME

      WAIT FOR $ASYNC_STATE == #IDLE
      SetCmdDone()

    CASE 2 ; Mount tool 1
      SetCmdStart()

      MOUNT_TOOL_1()

      WAIT FOR $ASYNC_STATE == #IDLE
      SetCmdDone()

    CASE 3 ; Unmount tool 1
      SetCmdStart()

      UNMOUNT_TOOL_1()

      WAIT FOR $ASYNC_STATE == #IDLE
      SetCmdDone()

    CASE 21 ; Pick from S12 shuttle
      SetCmdStart()

      PICK_MODULE_S12()

      WAIT FOR $ASYNC_STATE == #IDLE
      SetCmdDone()

```

Figure 5.27: Example of the robot program selector

5.2.10 Extending Process Simulate

At the time of creating the Virtual Commissioning, PS was missing some functions that were required for correct simulation. Namely, these were creating snapshots (saving images) from cameras automatically during runtime and mounting and unmounting tools during line operation mode with virtual robotic controllers. To get this functionality, the PS support for Microsoft .NET extensions was made use of. An extension was created using the provided Tecnomatix.dll library and required registration to PS via the CommandReg executable provided [25, 26]. This extension included three functions that could be called with buttons from the control ribbon as shown in 5.28.

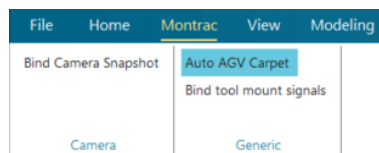


Figure 5.28: Montrac extension ribbon bar

The first function allowed a camera to be bound to a signal and when that signal's integer value changed from zero to a positive number, a snapshot from that camera would be saved next to the project file. Attaching a camera

to a signal using this function opens a new graphics viewer for that camera if it does not exist yet. Whenever a new snapshot is taken, it uses the current configuration of the camera and focuses the graphics viewer unless the “Tabbed” viewers mode is selected. An example snapshot can be seen in 5.29.

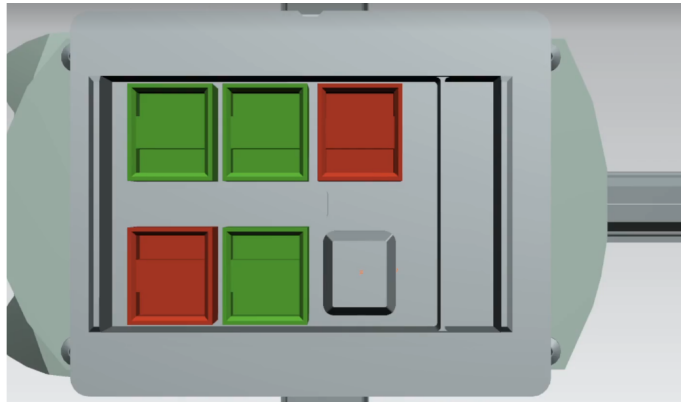


Figure 5.29: Camera snapshot example

The second function required a selection of one robot, one frame, and one gripper. In turn, it would create two robotic signals, one for mounting and the other for unmounting said gripper at any time, even during line simulation runtime. The selected gripper also had to be defined in the relevant robot’s toolbox under the name of its model component. If the operation could not be finished for any reason, the user would get an error message explaining what went wrong. An example of binding gripper “R1_PGN_plusP_64_assembly” to robot R1 at frame “DATUM_CSYS(1)” can be seen in 5.30 and the resulting signals in 5.31.

The last function would save time by automatically assigning the first object called “AGV_CARPET” as the AGV carpet when switching to line simulation mode, which would have to be done manually otherwise.

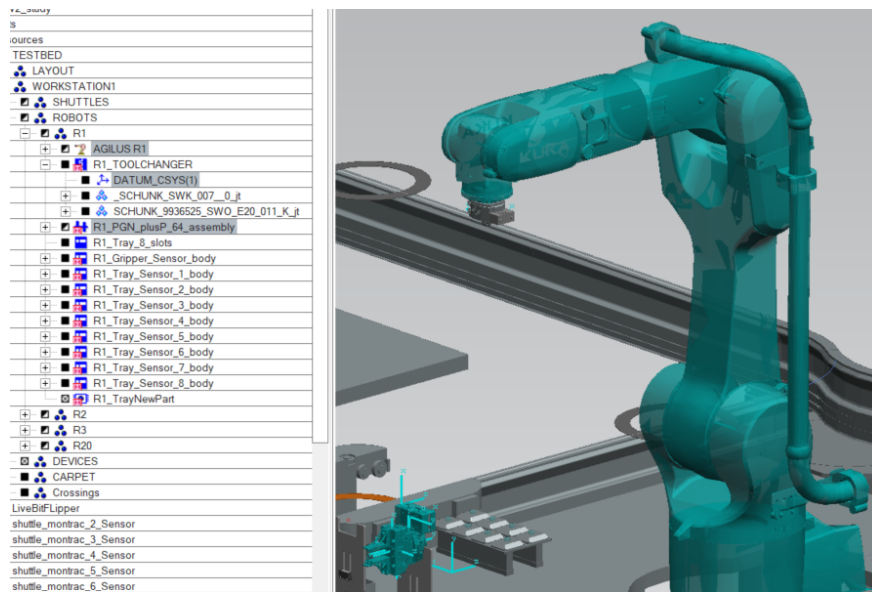


Figure 5.30: Gripper binding setup



 AGILUS R1_Mount_R1_PGN_plusP_64_assembly	BOOL
 AGILUS R1_Unmount_R1_PGN_plusP_64_assembly	BOOL

Figure 5.31: Gripper mounting signals

5.2.11 Communication

To allow control from MES and other possible clients, control via OPC-UA was implemented alongside control from PS. Only one of these input methods can be used at once but outputs are always sent to the OPC-UA interface. The interface was created using Siome and gives access to shuttles and robots. To simulate MES without the need for it to be deployed a Python script that mimics the way MES interacts with montrac was written. This script also allowed for automated testing of shuttles, robotic operations, and a full sample production process. This means that integrating the real MES is trivial by simply connecting it to the same interface. The script was based on a OPC-UA shuttle sender randomizer by Petr Douda.

The diagram 5.32 was created to help visualize the communication happening.

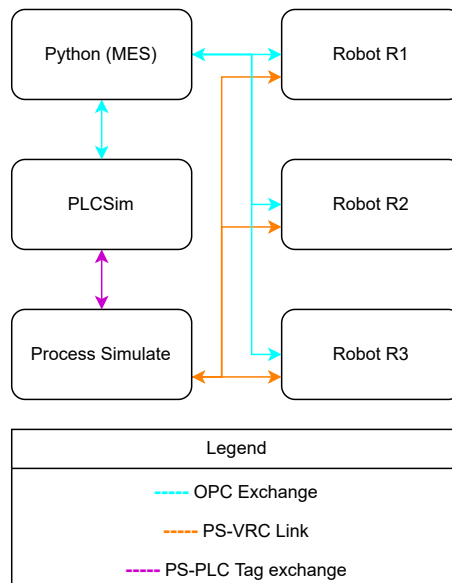


Figure 5.32: Diagram of communication channels

5.3 Overview

Considering the number of individual components and interconnected behavior present, this section provides a reference overview for them.

The final PS study can be seen in 5.33 and consisted of the following objects:

- Montrac conveyor system:
 - Rail segments
 - Crossings with kinematics, which connected a specific route based on the request on their logic block input and outputted the current route.
 - Stop targets, which represented either a station, crossing inlink, or an inter-stop for fine-tuning shuttle paths.
 - AGV-based shuttles, which allowed for the gripping of objects above them in order to transport them and were controlled via Logic Blocks. Each shuttle also had a proximity sensor to prevent collisions.
 - Wire-based AGV carpet which shuttles moved on
 - Support structure
- Robot R1, R2 and R3, each with:
 - Mounting fixture
 - Tool changer
 - Tool rack

- Gripper with a pickup sensor and a logic block, which handled gripping and releasing objects based on the inputs
- Workspace table
- Module storage trays for R1 and R2, which contained sensors on each position and a logic block that triggered its output when a new part was placed in any position
- Robot R20 with:
 - Mounting fixture
 - Tool changer
 - Tool rack
- Camera above station S300, which took pictures of shuttles in the station and was used for module detection
- Logic block “Bit Flipper” to allow detection of running simulation by the PLC
- Switchboards

In addition to the objects, the ability to automatically take snapshots with cameras and the ability to mount or unmount tools based on robotic signals was present from the extension developed in 5.2.10.

The PLC implementation details can be seen in section 5.2.7. The code consisted of several utility functions and the following main modules:

- Process Simulate connection setup
- Unit tests
- Direct writes between OPC-UA and PS
- Updating Data Blocks
- Processing shuttles (control logic)
- Processing crossings (FSM)
- Processing stations (FSM)
- Processing interstops

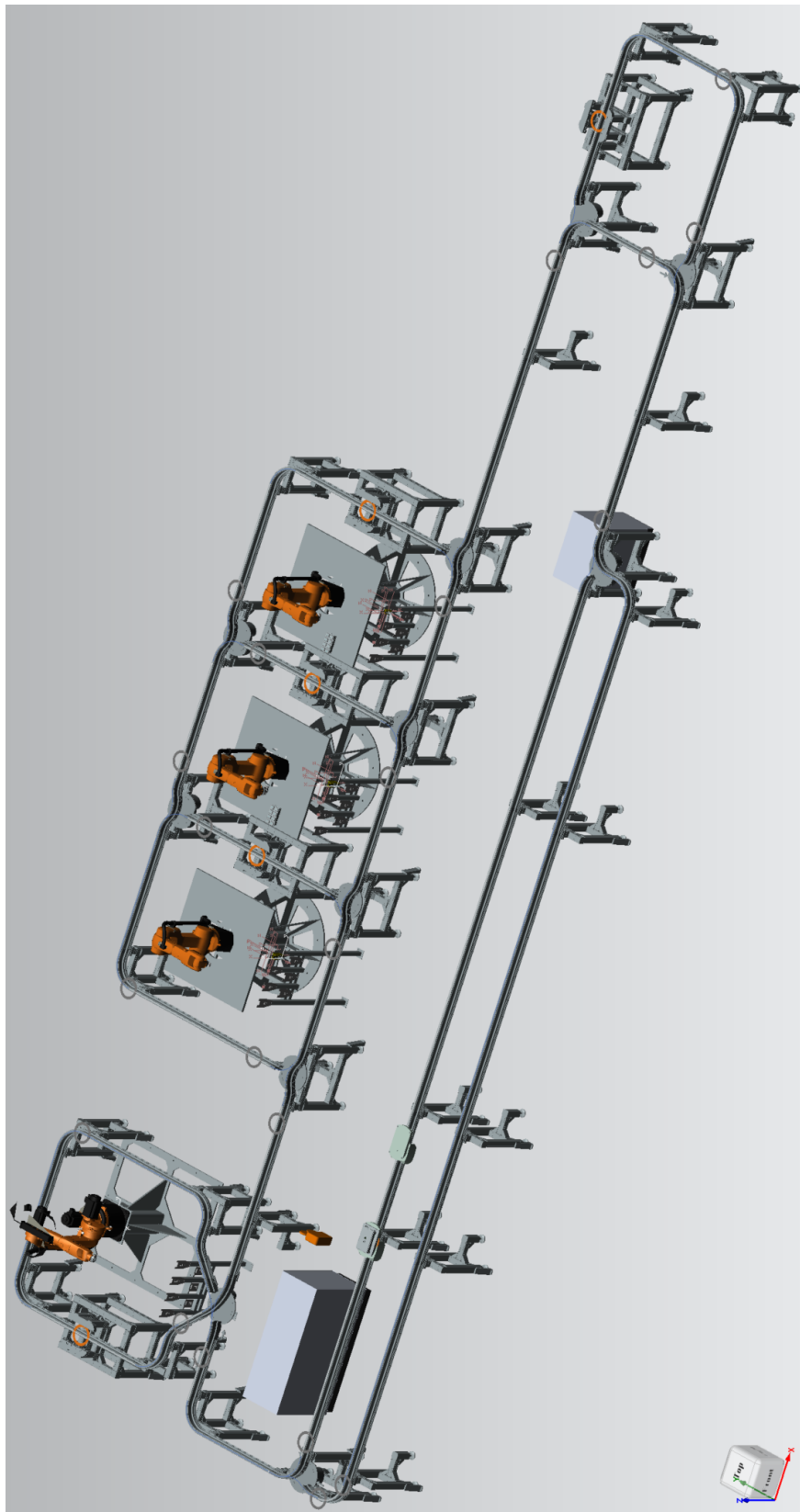


Figure 5.33: Process Simulate study image

Chapter 6

Experiments

6.1 Testing conveyor line

6.1.1 Table lookup

Before testing the table lookup itself, the function for discarding entries in control tables was tested first. This consisted of testing both valid and invalid indices of an empty table and tables with various data arrangements to cover most possible scenarios. Next, table lookup was tested with only the control table having entries, only the chaos table having entries and both tables having entries. These tests were done purely in TIA Portal.

6.1.2 Crossing and station objects

Having tested the table lookup functionality, the crossings and stations could be tested. Each type of crossing was tested in simulation with shuttles being routed based on their routing tables. Both visual and functional correctness were verified.

6.1.3 Operation

After testing stations and individual types of crossings, the full conveyor system was tested using the Python script from 5.2.11 extended to send shuttles to random stations and verify that they arrived at the correct station on time. This script allowed for semi-automatic testing where no manual input was required unless an error had happened.

6.2 Testing robotic workplaces

6.2.1 Operations in standard mode

The behavior of both individual mount, pick, place, and unmount operations as well as their compound operation was tested. Attention was especially paid to safe picking and placing, meaning proper speed when close to the objects, correct gripping positions, and not changing the robot configuration during the motion.

6.2.2 Operations in line simulation with Virtual Robot Controllers

The VRC line simulation was tested to make sure that each robot performed its assigned operations correctly. This included correct tool and base definitions, correct behavior when starting and performing programs, and proper OPC-UA access.

6.3 Flexible line operation

Lastly, the whole line was tested on a sample production cycle consisting of loading a battery module, removing the lid, detecting two module types, and picking the modules from the battery based on their type. This process was performed using the MES substitution script which should behave the same way a real MES would. This test was recorded as proof of correct behavior and the video can also be found in the attachments. A link to the whole project on GitHub with access limited to Testbed members is also in the attachments.

The battery module detection part of the MES substitution script was based on a version by Martin Jílek, thresholding the image in Hue-Saturation-Value (HSV)-spectrum and detecting continuous shapes using OpenCV2 [27]. The modules were then either identified as “OK” or “NOK” based on their color and their positions were discretized to represent which battery slots they were in. For example, the snapshotted module configuration in 5.29 would be detected as visualised in 6.1, returning the following values:

```
1 modulesOk = [(1,0), (1,1), (2,1)]  
2 modulesNok = [(2,0), (0,1)]
```

A real robot picking a module detected as “OK” can be seen in 6.2.

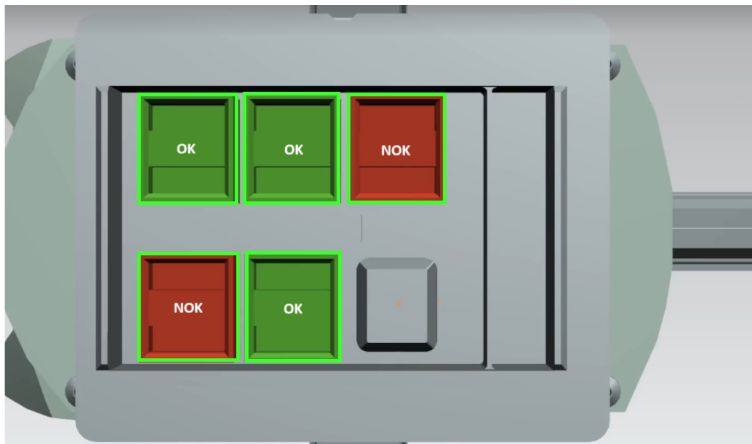


Figure 6.1: Detection result

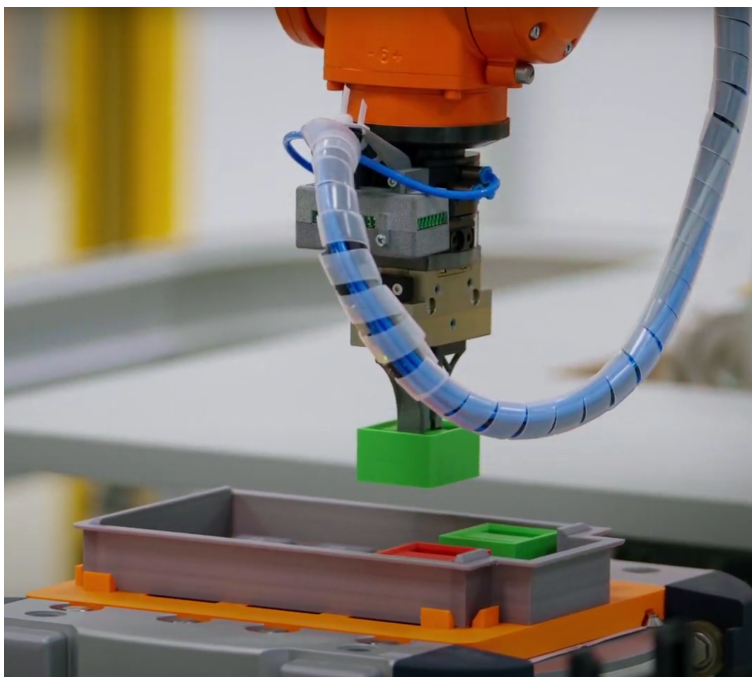


Figure 6.2: Real robot picking a detected module



Chapter 7

Conclusion

The Virtual Commissioning accurately replicated the physical behavior of the production line, including the movement and interaction of shuttles and robots. This fidelity is crucial for validating control logic and ensuring that the virtual model can be relied upon for system design and testing. The integration of VRCs allowed for control of robotic operations within the simulated environment. This integration ensured that the robots performed their tasks like picking and placing modules correctly. The VC also demonstrated the flexibility of the production line to adapt to changes in production requirements. The ability to simulate different configurations and operational conditions without disrupting the actual production line is a significant advantage. The virtual environment facilitated testing of the PLC code, robotic programs, and overall system integration. Issues that might have gone unnoticed in a physical setup were identified and addressed in the simulation, ensuring a smoother deployment process.

While the VC achieved its primary objectives, several areas for potential improvement were identified. The VRC gripping mechanism could be changed from sensor-based to robot-signal-based gripping, enhancing the reliability of robotic operations. Automating the generation of robot signal operations during the export process would also be possible, potentially using free-text OLP to streamline this transition. The user interface could be improved for configuring and monitoring the VC, making the system more accessible to operators and engineers. This includes better visualization tools and more intuitive controls for managing the simulation. A big step towards creating a full Digital Twin would be incorporating real-time data from the physical production line into the VC, enhancing the simulations.

The successful implementation of a Virtual Commissioning process for a robotic flexible line with a montrac conveyor system demonstrates the significant benefits of digital twin technology in manufacturing. The VC provided a platform for testing and optimizing the production line, especially the MES. By addressing the identified areas for improvement, future work can further enhance the capabilities of Virtual Commissioning and simplify the process.

Appendix A

Bibliography

- [1] Siemens, “What is robotics virtual commissioning?.” <https://www.plm.automation.siemens.com/global/en/our-story/glossary/what-is-robotics-virtual-commissioning/102446> [Accessed: 12/2/2024].
- [2] Siemens, “Robotics programming and simulation.” <https://plm.sw.siemens.com/en-US/tecnomatix/robotics-programming-simulation/> [Accessed: 12/2/2024].
- [3] Siemens, “Plan your factory installation with virtual commissioning tools before building the machine.” <https://resources.sw.siemens.com/en-US/white-paper-virtual-commissioning-uses-digital-twins-to-validate> [Accessed: 12/2/24].
- [4] Siemens, “Virtual commissioning with siemens solutions reduces launch time by three weeks.” <https://resources.sw.siemens.com/en-US/case-study-idc> [Accessed: 12/2/24].
- [5] Siemens, “Reducing robot programming time by 25 percent.” <https://resources.sw.siemens.com/en-US/case-study-ethos-automation> [Accessed: 12/2/2024].
- [6] G. Barbieri, A. Bertuzzi, A. Capriotti, L. Ragazzini, D. Gutierrez, E. Negri, and L. Fumagalli, “A virtual commissioning based methodology to integrate digital twins into manufacturing systems.” <https://doi.org/10.1007/s11740-021-01037-3> [Accessed: 20/5/2024].
- [7] C. Scheifele, A. Verl, and O. Riedel, “Real-time co-simulation for the virtual commissioning of production systems.” <https://linkinghub.elsevier.com/retrieve/pii/S2212827119302215> [Accessed: 20/5/2024].
- [8] N. Striffler and T. Voigt, “Concepts and trends of virtual commissioning – a comprehensive review.” <https://linkinghub.elsevier.com/retrieve/pii/S0278612523002145> [Accessed: 20/5/2024].

- [9] K. W, K. M, T. G, H. J, and S. W, “Digital twin in manufacturing: a categorical literature review and classification.” IFAC Pap 51(11):1016–1022.
- [10] L. Y, L. C, W. KIK, H. H, and X. X, “Digital twin driven smart manufacturing: connotation, reference model, applications and research issues.” Robot Comput Integr Manuf 61:101837.
- [11] T. F, Z. M, L. Y, and N. A, “Digital twin-driven prognostics and health management for complex equipment.” CIRP Ann 67(1):169–172.
- [12] S. W, H. T, Y. Y, H. J, T. F, and N. A, “Digital twin based virtual commissioning for computerized numerical control machine tools. digital twin driven smart design.” Elsevier, Amsterdam, pp 289–307.
- [13] B. A, S. D, G. P, K. K, P. P, and C. R, “Programming of industrial robots using virtual reality and digital twins.” Appl Sci 10(2):486.
- [14] V. Outlý, “Řízení dopravníku v systému automatického rozvrhování výroby.” <https://dspace.cvut.cz/handle/10467/85066> [Accessed:20/5/2024].
- [15] T. Staruch, “Multi-agent systems for production planning.” <https://dspace.cvut.cz/handle/10467/107203> [Accessed: 20/5/2024].
- [16] montrac®, “Downloads.” <https://www.montratec.de/en/downloads/> [Accessed: 1/10/2023] Restricted Access.
- [17] Oracle, “What’s the difference between an mes and erp system?.” <https://www.netsuite.com/portal/resource/articles/erp/mes-erp-differences.shtml> [Accessed: 14/3/2024].
- [18] SAP, “What is a manufacturing execution system (mes)?.” <https://www.sap.com/products/scm/execution-mes/what-is-mes.html> [Accessed: 14/3/2024].
- [19] Siemens, “What is a manufacturing execution system?.” <https://www.plm.automation.siemens.com/global/en/our-story/glossary/manufacturing-execution-systems-mes/38072> [Accessed: 14/3/2024].
- [20] Siemens, “Totally integrated automation portal – always ready for tomorrow.” <https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html> [Accessed: 13/5/2024].
- [21] K. AG., “Kuka.workvisual.” https://www.kuka.com/en-us/products/robotics-systems/software/system-software/kuka_systemsoftware/kuka-work-visual [Accessed: 26/4/24].
- [22] K. AG., “Kuka.officelite.” https://www.kuka.com/en-us/products/robotics-systems/software/simulation-planning-optimization/kuka_officelite [Accessed: 26/4/24].

- [23] scooley, “Introduction to hyper-v on windows 10.” <https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/> [Accessed: 26/4/24].
- [24] KUKA, “Download center.” <https://www.kuka.com/en-us/services/downloads> [Accessed: 20/5/2024].
- [25] J. Dryk, “An extension of process simulate for optimization of robotic cells.” <https://dspace.cvut.cz/handle/10467/73950> [Accessed: 1/4/24].
- [26] Siemens, “Tecnomatix.net api.” <https://docs.sw.siemens.com/en-US/doc/288782031/PL20230416673824280.TecnomatixNET-API-Tecnomatix2307..NET-API-doc> [Accessed: 1/4/24].
- [27] OpenCV, “Opencv documentation.” <https://docs.opencv.org/4.x/> [Accessed: 18/5/2024].