NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

JORGE MIGUEL SOBRAL DOS SANTOS

BSc in Electrical and Computer Engineering Sciences

AUGMENTED REALITY FOR VIRTUAL COMMISSIONING WITHIN MODULAR PRODUCTION SYSTEMS

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

AUGMENTED REALITY FOR VIRTUAL COMMISSIONING WITHIN MODULAR PRODUCTION SYSTEMS

JORGE MIGUEL SOBRAL DOS SANTOS

BSc in Electrical and Computer Engineering Sciences

Adviser: André Dionísio Bettencourt da Silva Parreira Rocha
Assistant Professor, NOVA School of Science and Technology

Examination Committee

Chair: Luís Filipe dos Santos Gomes
Associated Professor with Habilitation, NOVA School of Science and Technology

Rapporteur: João Almeida das Rosas
Assistant Professor, NOVA School of Science and Technology

Adviser: André Dionísio Bettencourt da Silva Parreira Rocha
Assistant Professor, NOVA School of Science and Technology

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2023

Augmented Reality for Virtual Commissioning within Modular Production Systems

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.5) [1].

To my friends, my family, and my future self.

Acknowledgements

I want to start this document by thanking everyone who allowed and helped me to achieve my academic goals.

I want to thank my adviser, Professor André Dionísio Bettencourt da Silva Parreira Rocha, for allowing me to explore such an interesting project for my dissertation and for helping me through its development.

Additionally, I want to thank every teacher who has contributed to this big learning experience, especially the ones from my department, the *Department of Electrical and Computer Engineering*.

The *NOVA School of Science and Technology* also deserves to be acknowledged for receiving me and allowing me to study in this impressively welcoming universitarian campus.

A special thank you has to be given to all my family, who always motivated me throughout this challenging course. I'm especially grateful for all the support given to me by my sister, my parents, and my cousins throughout this academic journey.

I have to thank my close friends for believing that I was able to push through all the hard work involved with this long challenge.

Finally, I thank my close group of colleagues that was formed through this journey and whom I have the pleasure of calling friends since almost the very beginning of this course. Thanks to them, I was able to surpass every new academic challenge that appeared in front of me, because these challenges would also appear in front of them and we would mutually help each other to surpass them. Additionally, these colleagues were also the friends with whom I would spend time to forget about my academic problems. I am really deeply grateful for all your help and companionship *MTBS*.

”

“It is not enough to have a good mind, the main thing is to use it well.”

— René Descartes
(Philosopher, scientist and mathematician)

Abstract

The testing of the control software of a manufacturing system, also known as commissioning, takes up to 25% of the total project's time. This time can be significantly reduced through Virtual Comissioning (VC), which also reduces risks and costs associated with the traditional commissioning approaches. Despite this, VC hasn't yet been used to commission modules that are to be added to a Modular Production Systems (MPS).

The use of MPSs has been growing since these systems allow shorter product life cycles, more product customization, and system flexibility.

In addition to this, the addition of modules to a system is better done when it is visualized before these are bought to prevent unnecessary associated delays and costs.

This document proposes a solution that allows a manufacturer to test the control software of a module that they want to add to a MPS through VC while using Augmented Reality (AR) to provide an accurate visualization of the module's appearance, operation, and interaction with the remaining system.

The solution was tested and it was possible to conclude that it solves the proposed problem, but that it has to be tested for industrial systems and adapted for different types of systems.

Keywords: Industry 4.0, Multi-Agent Systems, Modular Production Systems, Simulation, Virtual Commissioning, Augmented Reality

Resumo

O teste do software de controlo de sistemas de manufatura pode demorar até 25% do tempo total de um projeto. Este tempo pode ser significativamente reduzido através de Comissionamento Virtual (CV), que também reduz riscos e custos associados aos métodos de comissionamento tradicionais. Apesar disto, o CV ainda não foi utilizado para comissionar módulos a ser adicionados a Sistemas de Produção Modulares (SPM).

O uso de SPMs tem vindo a crescer já que estes permitem um menor tempo de manufatura de produtos, uma maior customização dos mesmos, e uma maior flexibilidade do sistema.

Para além disto, a adição de módulos a um sistema é feita da melhor forma quando esta pode ser visualizada antes da sua compra para prevenir atrasos e custos desnecessários.

Este documento propõe uma solução que permite que um fabricante teste o software de controlo de um módulo que este queira adicionar a um SPM através de CV e usando Realidade Aumentada (RA) para permitir uma visualização precisa da aparência, do comportamento, e da interação deste com o restante sistema.

A solução foi testada e concluiu-se que esta resolve o problema apresentado, mas que deve ser testada em sistemas industriais e adaptada para diferentes tipos de sistemas.

Palavras-chave: Indústria 4.0, Sistemas Multi-Agente, Sistemas de Produção Modulares, Simulação, Comissionamento Virtual, Realidade Aumentada

Contents

List of Figures

List of Tables

Acronyms

ABM	Agent-Based Models
ABMS	Agent-Based Modeling Simulation
AchieveRE	Achieve Rational Effect
ACLMessage	Agent Communication Language Message
AGV	Automated Guided Vehicle
AMS	Agent Management System
API	Application Programming Interface
AR	Augmented Reality
AV	Augmented Virtuality
COI	Conveyor Ordering Interface
CPS	Cyber-Physical Systems
CV	Comissionamento Virtual
DES	Discrete Event Simulation
DF	Directory Facilitator
DT	Digital Twin
FIPA	Foundation of Intelligent Physical Agents
GPS	Global Positional System
GUI	Graphical User Interface
HiL	Hardware-in-the-Loop
HS	Hybrid Simulation
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
I/O Controller	Input/Output Controller

I4.0	Industry 4.0
ICPS	Industrial Cyber-Physical Systems
ICS	Industrial Control Systems
ID	Identification
IDE	Integrated Development Environment
IoT	Internet of Things
IPC	Industrial Personal Computer
JADE	Java Application DEvelopment Framework
JAR	JADE Archive
LAN	Local Area Network
MAS	Multi-Agent Systems
MPS	Modular Production Systems
MR	Mixed Reality
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
OS	Operating System
PLC	Programmable Logic Controller
QR	Quick Response
RA	Realidade Aumentada
RE	Real Environment
RFID	Radio-Frequency Identification
RiL	Reality-in-the-Loop
SD	System Dynamics
SiL	Software-in-the-Loop
SLF4J	Simple Logging Facade for Java
SPM	Sistemas de Produção Modulares
TC	Traditional Commissioning
UI	User Interface
UML	Unified Modeling Language

VC Virtual Comissioning
VE Virtual Environment
VR Virtual Reality

WSN Wireless Sensor Networks

1 Introduction

This chapter intends to introduce this document in the following way. It starts by giving a context to its theme in the Motivation section (Section 1.1). It then presents the problem that inspired the creation of this document in the Problem section (Section 1.2). Then, in Section 1.3 (Objectives), it presents the objectives that the project associated with the document should fulfill for it to be a good solution to the previously introduced problem. Section 1.4 (Solution) presents this solution in a broad way and explains how it works. Finally, the chapter concludes by indicating how the document is organized by presenting the different chapters that compose it and explaining the purpose of each of them in Section 1.5 (Document Organization).

1.1 Motivation

The fast rate at which technology has been evolving has created a great amount of opportunities for its use in the industrial context. This potential for the use of technology in the industry has culminated in the emergence of the term Industry 4.0 (I4.0), which represents the fourth industrial revolution. This term refers to this industry's digitalization, which, in the manufacturing context, translates into the automation of production lines through the use of internet technologies and smart objects (products and machines), also known as Cyber-Physical Systems (CPS) [2, 3].

With these industrial developments, factories can turn into smart factories, which make use of Modular Production Systems (MPS). MPSs are composed of individualized modules that can execute one or more functions. The individualization of the modules allows the customization of the manufactured products since these can interact with any module of the system in any necessary order. This occurs because the modules are not part of a fixed production line, instead being able to interact with each other and with the product in any necessary way. MPSs also have the capability of reducing the life cycle of products since there is no path through which every product must go, in other words, a product can make the most optimal path through the factory to become ready as fast as possible [2, 3].

Nowadays, smart factories with MPSs are in constant change in order to satisfy the consumers' ever-changing necessities and desires. These changes are usually done by removing and adding modules to the smart factory's MPSs. In order to test the software that controls a module that is to be added to the system, this new module is usually integrated into the system, and the whole system is tested. This can be referred to as Traditional Commissioning (TC) and can be summarized as the use of a real controller to test a real module or system.

1.2 Problem

Although the commissioning of a module or a system is always necessary, the TC approach has some problems associated with it. The commissioning of a project through a traditional approach can take from 15% to 25% of its total time, of which more than 60% is associated with software debugging, as displayed in Figure 1.1 [4].

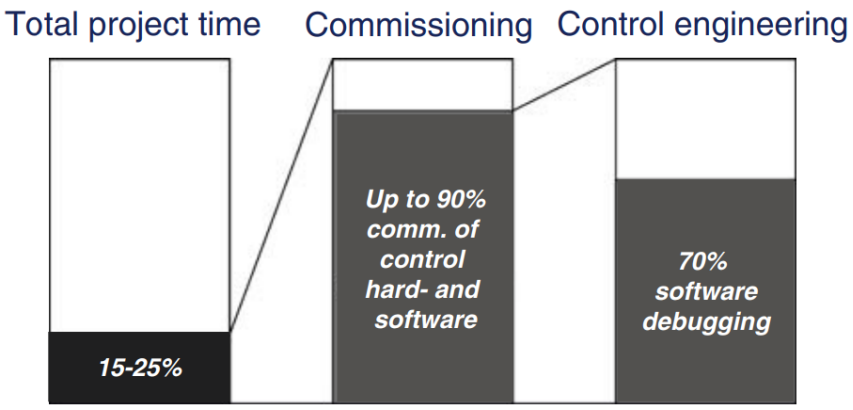


Figure 1.1: Project delay associated with Traditional Commissioning [4].

In addition to this, TC is also associated with risks, since the real hardware is being used to test software code that hasn't been totally validated yet, which can lead to unpredictable behaviors [5]. Finally, TC implies various costs, these costs are associated with the system's hardware acquisition and the energy consumed during its operation [6].

A solution to these presented problems is the use of Virtual Commissioning (VC). VC refers to the commissioning of a virtual module or system, instead of a physical one. This can be done through the use of a real or a simulated controller to control a simulation of the physical module or system. This still allows the testing of the control software (commissioning) but doesn't have the delays, risks, and costs associated with TC [5, 6].

Although the use of VC in the manufacturing industry has been growing, it is usually used to commission an entire system or an individual module, but without testing the interaction of this module with the rest of the system, therefore not actually simulating the commissioning of the module, since it is not being integrated into its actual operating environment. The only instances that tested the integration of virtual modules in a real

system do not test the control software that would be used if these modules were physical, which means these are not commissioning approaches.

In summary, the addition of modules to a system is usually tested through TC or through simulations that don't fully commission these modules since their control code is not tested throughout these simulations. This means that there is no approach to fully virtually commission the addition of modules to MPSs.

As mentioned previously, VC is done through the simulation of hardware. This simulation can be done in many different ways, from the simple display of the hardware's components state in a monitor to the creation of a Virtual Reality simulation to allow a realistic visualization of the hardware's operation.

To virtually commission the addition of a module to a MPS, it is extremely advantageous to have a good visualization of the module in operation. A good visualization of the virtual commissioning of a module allows the manufacturers to have a good idea of how the system will behave and look after the physical addition of this module.

This means that these manufacturers can test the acquisition of the module during its virtual commissioning, instead of having to commit to the monetary cost of the module and risk it not behaving or looking exactly as intended, which would lead to the use of a not completely fitting module, or even worse, the necessity to buy a new module and spend more monetary resources. Additionally, it is much easier for a manufacturer to visualize a module in operation on a monitor than to analyze the data associated with a simulation created for the module's commissioning.

1.3 Objectives

This document intends to present a method of commissioning a module that a manufacturer wants to add to a MPS without the need to purchase it while allowing it to be visualized in operation with the remaining system.

For this, a project will be developed to demonstrate an application of this method. This project should fulfill the following objectives in the context of MPSs:

- Testing the controller code for a new module without having to purchase it.
- Visualizing the integration of a new module into an existing real system by watching it operate with that system in its environment.

The main steps taken throughout the development of this project to fulfill the previous objectives were the following:

- Programming a Graphical User Interface (GUI).
- Programming the control logic of a MPS.
- Creating a 3D module of a Physical Module (Virtual Module).

- Programming an application to allow the visualization of a Virtual Module in the Real Environment using Augmented Reality.
- Programming a Virtual Module to act as a Physical Module (Create a Digital Twin of a Physical Module).
- Enabling and programming the communication between different applications.

1.4 Solution

The solution proposed by this document for the fulfillment of these objectives intends to commission a new module of a MPS through a Virtual Commissioning (VC) approach and to allow its visualization through Augmented Reality (AR).

The developed solution uses a MPS composed of two small-scale industrial conveyors. A third conveyor is to be introduced in this system, and its control software is tested through VC, which means this conveyor is simulated and there is no necessity for its physical acquisition. This conveyor can be visualized in AR by scanning an AR marker with a mobile device (smartphone for example).

Since the system is a MPS, its modules can interact with each other in any necessary way. For this reason, the user can arrange the system's conveyors in any order he wants and insert this order in a Graphical User Interface (GUI). The virtual conveyor can be introduced into the system through the placement of the AR marker in the intended position.

The user can then start the simulation, which allows them to verify if the new module is behaving as intended and if the complete system is operating in the intended way after its addition.

1.5 Document Organization

For the demonstration of this solution, the document possesses the following structure. It is composed of the chapters: State of the Art; Architecture; Implementation; Tests and Validation; Conclusions and Future Work.

Chapter 2 (State of the Art) starts by introducing the concept of Industry 4.0 (I4.0). It presents the most relevant simulation approaches in the manufacturing industry, while also focusing on analyzing the different methods of commissioning and visualization in this context. This chapter also presents the existing work in commissioning and visualization in this industry and concludes which technologies will be used throughout the project's implementation by effectuating a comparison between the main existing options.

Chapter 3 (Architecture) presents, as the name indicates, the architecture of the project. It starts by presenting important concepts that the reader should know before it continues reading. It then deduces the requirements that the architecture must comply with to fulfill

the previously mentioned objectives. The chapter then presents the blocks that compose the architecture, explains the function of each of them, and how they interact with each other. It concludes by demonstrating how the system should operate.

The Implementation chapter (Chapter 4) explains how the project was implemented. It explains the technologies used for this implementation and then explains how each of the project's blocks was implemented.

The chapter Tests and Validation (Chapter 5) shows the tests that were done on the developed project and validates it by concluding if it works in the intended way by verifying it fulfills the previously presented objectives.

Finally, Chapter 6 (Conclusions and Future Work) concludes the document by summarizing both it and the development of the project. It also presents the main limitations of the proposed method and suggests work that can be done to enhance it.

2

State of the Art

This chapter introduces and explains the main concepts related to today's industry and to simulation in its context. It also presents the existing work on the use of Augmented Reality for simulation in manufacturing.

The chapter is divided into four sections. Section 2.1 introduces the term Industry 4.0 and explains the main technologies associated with it. Section 2.2 presents the most relevant ways of simulating industrial systems by presenting the most relevant simulation approaches and the technologies relevant in the context of this project. The chapter also presents existing work in Virtual Commissioning and Augmented Reality for manufacturing in Section 2.3. Finally, Section 2.4 summarizes this chapter.

2.1 Industry 4.0 and its enabling technologies

This section introduces the term Industry 4.0 and its enabling technologies. In Section 2.1.1, the term Industry 4.0 is defined and explained.

The following three sections introduce the three main enabling technologies of Industry 4.0. Section 2.1.2 talks about the Internet of Things, Section 2.1.3 explains cloud computing, and Section 2.1.4 introduces the concept of Cyber-Physical Systems (CPS).

Finally, Section 2.1.5 explains the impact that CPS have in the industry by introducing and explaining the term Industrial Cyber-Physical Systems (ICPS).

2.1.1 Industry 4.0

The term Industry 4.0 (I4.0), also referred to as the Fourth Industrial Revolution, was introduced in 2011 at the Hannover Fair in Germany [7, 8].

I4.0 symbolizes the origin and evolution of technologies that enable automation in manufacturing, the three main enabling technologies associated with I4.0 are: the Internet of Things (IoT); cloud computing; and Cyber-Physical Systems (CPS) [8].

I4.0 represents the evolution from simple embedded systems to systems controlled digitally through IoT technologies, machine-to-machine communication, and many other technologies, turning these systems into CPS [8].

The three following sections (Sections 2.1.2, 2.1.3, and 2.1.4) will introduce these three enabling technologies that are associated with I4.0.

2.1.2 Internet of Things

IoT can be seen as a network of devices that are connected to each other through smart sensors and wirelessly, in which these devices are identifiable through techniques of Near Field Communication (NFC) and can communicate without the help of humans [9–11].

When the term IoT was first introduced, the devices of a network were identified through Radio-Frequency Identification (RFID) technology, which allows the identification and real-time tracking of devices that use RFID tags when the corresponding RFID reader is connected to the internet [8, 11, 12].

Nowadays there are more options for the execution of this and more tasks, which means there are more technologies associated with IoT, some of the technologies that support and are used in the context of IoT are RFID, Wireless Sensor Networks (WSN), Global Positional System (GPS), etc [8].

Figure 2.1 shows many of the technologies that support IoT.

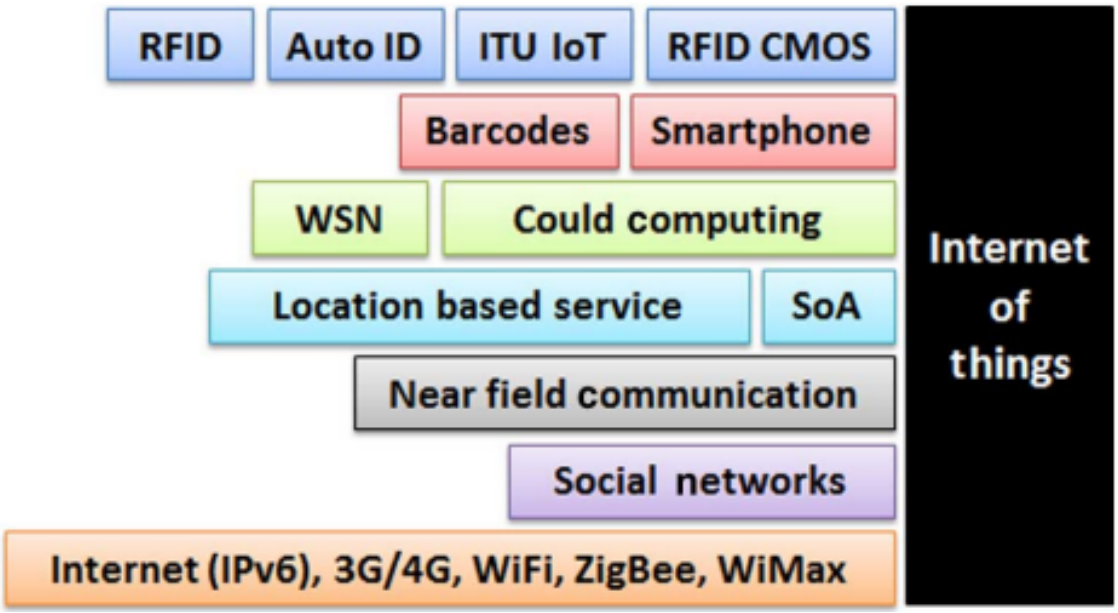


Figure 2.1: Technologies that support IoT [11].

2.1.3 Cloud computing

As many other terms introduced in this document, the term cloud computing doesn't have a fixed definition [13]. However, the term represents accessing hardware or software through a network (most commonly the internet) instead of acquiring and using these locally.

A good definition of this term was provided by the National Institute of Standards and Technology (NIST), which stated the following: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [14].

Cloud computing brings many advantages to its users and providers, some of which were referenced in the article: *Cloud computing: state-of-the-art and research challenges*, which served as a reference for the presentation of the advantages displayed next and their corresponding explanations [13].

No investment in equipment: With cloud computing there is no need to invest in the equipment required to realize the desired computational actions or in its associated preparations. The user only needs to, if necessary, pay for the service that he requires, and this service can easily be upgraded or downgraded by paying more or less to the infrastructure provider.

Lower operation cost: A service provider that uses cloud computing can easily allocate or de-allocate the necessary resources according to their usage, which means he can lower the costs by releasing resources when they aren't being used.

Scalability: Service providers can obtain a large number of resources by buying them from data centers and, by doing so, they can easily upgrade the services they provide. This means that the services can be highly scalable, which makes the provider's job easier.

Easily accessible: The services provided by cloud computing are generally easily accessible by anyone since most of them are provided through the internet, therefore being accessible through most devices (desktop computers, laptops, smartphones, etc.).

Cost reduction in risks and maintenance: This advantage applies especially to businesses since they would have costs associated with maintenance and risk management of the resources needed. With cloud computing, these businesses can avoid these costs by passing them to the infrastructure providers. This advantage also facilitates the creation of new businesses since there are fewer preoccupations for the business creator or creators.

2.1.4 Cyber-Physical Systems

Cyber-Physical Systems (CPS) are systems capable of executing physical tasks that are managed by computational processes, these processes are mainly used to control, coordinate, and monitor such systems [15].

CPS are most often composed of multiple embedded modules that have their own role in completing the desired objective of the whole system [16].

Smart cities, IoT technologies, and Industrial Control Systems (ICS) are all examples of CPS and demonstrate the importance they have nowadays [17].

2.1.5 Industrial Cyber-Physical Systems

The term Industrial Cyber-Physical Systems (ICPS) represents the use of CPS in the industry, although the term Cyber-Physical Production Systems (CPPS) is also used to refer to the CPS in the industry, most particularly in the production and manufacturing contexts of it [18, 19].

The importance of CPS in the industry derives from the benefits that technology can bring to it. With the evolution of technology, the computational layer of ICPS is constantly evolving to allow a better way to control and monitor the physical processes of industrial systems, usually relying on feedback loops between the physical and computational layers to do this [7]. CPS bring so much value to the industry that the term I4.0 itself refers to the digitalization of the industry sector through CPS [7].

2.2 Simulation in Industry 4.0

Simulation is a large part of I4.0. Since it surged, many technologies have been created and developed in order to facilitate simulation and obtain more information regarding industrial systems through it.

Simulation plays a large role in the industry of modern times since it is one of the most important technologies responsible for establishing models for optimizing the decision-making, design, and operation of industrial systems [20]. It is also capable of helping with the evaluation of costs, risks, challenges, and performance of such industrial systems [20].

There are several technologies able to aid in the simulation of industrial systems. In this section, there will be presented, defined, and explained some of the main simulation technologies.

Section 2.2.1 will focus on the three main simulation approaches. The remaining technologies that will be discussed are Digital Twins (Section 2.2.2), commissioning (Section 2.2.3), and reality-virtuality technologies (Section 2.2.4).

Additionally, Section 2.2.5 performs a comparison between many of the previously presented simulation technologies and approaches.

2.2.1 Simulation approaches

This section introduces the three main simulation types, which are: Discrete Event Simulation (Section 2.2.1.1); System Dynamics (Section 2.2.1.2); Agent-Based Modeling and Simulation (Section 2.2.1.3) [21]. The section also introduces the concept of Hybrid Simulation (Section 2.2.1.4) and concludes by making a comparison between the three mentioned simulation approaches (Section 2.2.1.5).

2.2.1.1 Discrete Event Simulation

Discrete Event Simulation (DES) is the most used simulation approach in manufacturing [22]. This simulation approach is able to model systems that can be represented as a chronological series of events that start and end at specific points in time [23].

The system's behavior is defined by state variables that represent, as the name says, the state that the system is in [23]. The sequence of events controls the simulation since its events trigger the system's state variables to change [21, 24]. Events can also trigger the addition of new events to the sequence [24].

DES can be and is usually represented by flowcharts, which ease the analysis of the system regarding the measurement of its performance and the pinpoint of process bottlenecks [25]. Although DES is a static representation of the system, the given inputs can be randomized in order to evaluate the behavior of the system under different circumstances [25].

2.2.1.2 System Dynamics

The System Dynamics (SD) approach allows for a study and evaluation of real-world systems [22]. This method allows a continuous simulation of the modeled system.

The model focuses on representing the relationship between system objects as causal relationships [22]. These relationships form feedback loops that react to changes in variables by affecting other variables over time [22]. These variables will consequently end up influencing the original variables since they're contained in the same feedback loop [22].

There are two types of feedback loops. A positive feedback loop increments variables and therefore amplifies existing system disturbances [22]. A negative feedback loop, on the other hand, decrements variables, which makes the system approach a goal [22].

An SD can therefore be represented as a causal loop diagram which is composed of interconnected feedback loops formed by the causal relationships mentioned before.

Figure 2.2 shows examples of positive and negative causal relationships and of a positive feedback loop.

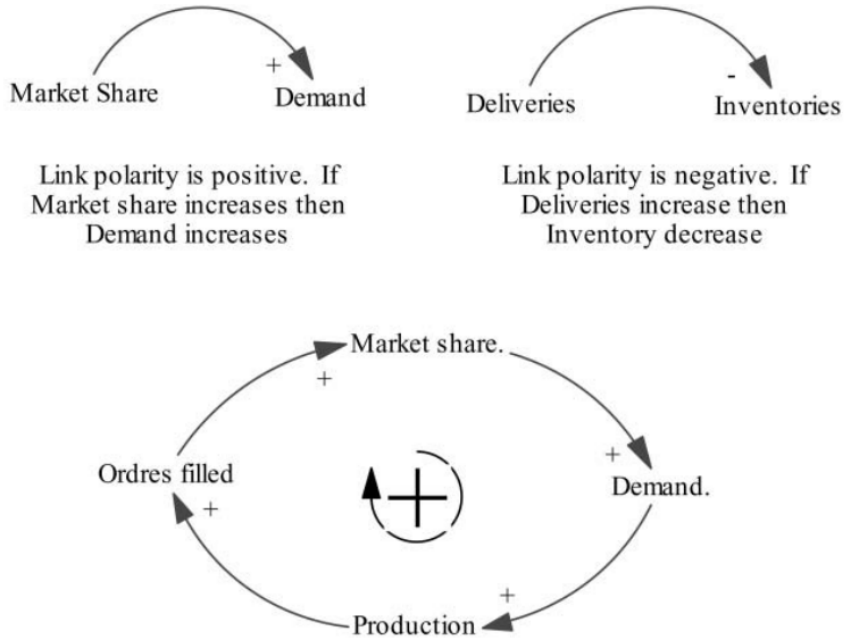


Figure 2.2: Example of a positive causal relationship, a negative causal relationship, and a positive feedback loop [22].

From causal loop diagrams, it is possible to create stocks and flows diagrams [22]. Stocks are boxes that represent a part of the system and that inform about its state [22]. Inflows and Outflows control, respectively, the rate at which the stock levels increase and decrease [22].

Figure 2.3 displays an example of a stock being affected by an inflow and an outflow.

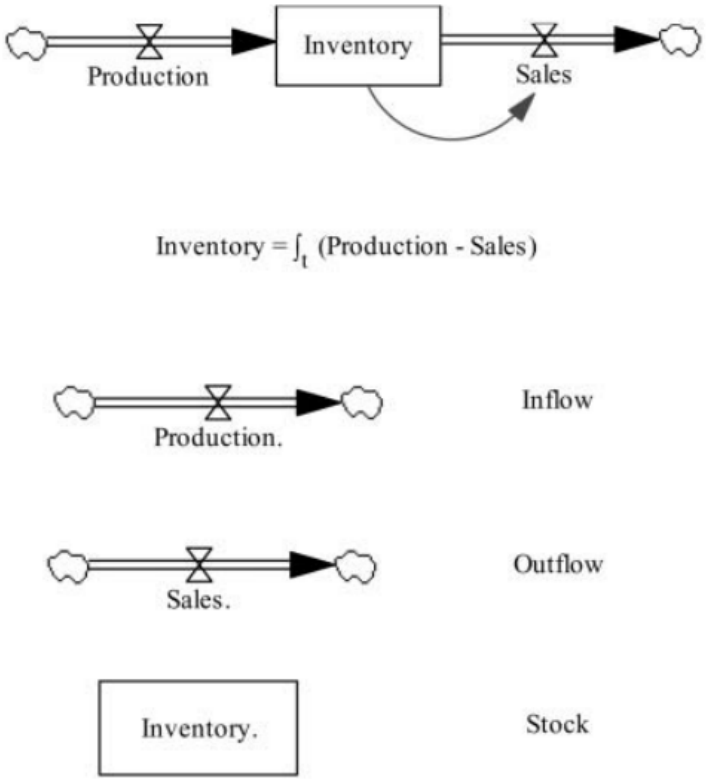


Figure 2.3: Example of a stock being affected by an inflow and an outflow [22].

It is then possible to translate these diagrams into mathematical equations for computers to solve, which results in the execution of accurate simulations [22].

2.2.1.3 Agent-Based Modeling and Simulation

Agent-Based Models (ABM) or Multi-Agent Systems (MAS) are systems composed of a set of agents that share the same environment [26]. These agents interact and communicate with each other through a given set of rules [26].

An agent is an entity capable of executing one or more tasks and communicating with users or other agents. In the industry, agents are usually pieces of software with the characteristics previously mentioned [27]. Agents can be seen as autonomous entities since each agent can execute its corresponding actions independently of users or other agents [27, 28].

The term Agent-Based Modeling Simulation (ABMS) refers to the use of computational models to process the actions and reactions of a MAS, including the communication protocols between the agents involved [28].

With ABMS it is possible to decompose a complex system into agents. This divides the system into different and simpler modules that can be approached individually, easing the modeling of complex systems [28]. ABMS allows for an analysis of a system's design and performance, therefore enabling an evaluation of the said system to be made [28].

2.2.1.4 Hybrid Simulation

The term Hybrid Simulation (HS) refers to the simulation through a model that uses two or more simulation approaches, the most commonly used approaches are the three that were previously mentioned [29, 30].

2.2.1.5 Comparison between simulation approaches

This section presents some of the advantages and disadvantages of the three simulation approaches previously introduced. These advantages and disadvantages are presented in order to establish a comparison between these approaches.

Table 2.1 presents the advantages and disadvantages previously presented.

Table 2.1: Advantages and disadvantages of DES, SD, and ABMS.

Simulation approach	Advantages	Disadvantages
Discrete Event Simulation (DES)	1. There is no need to assume simplifications like when using artificial intelligence and mathematic optimization.	1. Has difficulty in representing social behaviors.
	2. It's a flexible approach since it has many applications.	2. Modeling entities' movements and real-time decision-making are difficult since DES proceeds through discrete time steps.
	3. Can model complex systems even when they have uncertainties or stochastic elements, which is difficult to do with other approaches like SD and ABMS.	3. It's hard to model complex integrated systems with DES especially if they involve continuous dynamicity, and it also becomes hard to interpret the simulation results of such systems.
	4. It's possible to analyze specific resources and entities.	4. The model doesn't account for instability in the system's variables, which makes it a bad approach for the simulation of systems that can get highly influenced by small changes in these variables.

Table 2.1 – continued from previous page

Simulation approach	Advantages	Disadvantages
	5. The execution in discrete time makes DES models run quickly.	5. Its operation method can't be adapted while the simulation is running, so the system's behavior must be completely known in advance.
	6. It's good at modeling queues.	
	Source: Scheidegger et al. (2018) [21].	**Source:** Scheidegger et al. (2018) [21].
System Dynamics (SD)	1. Allows for a good vision of the whole system instead of focusing on its entities.	1. Since SD uses high-level components, it becomes harder to simulate the system in a low-level approach, with more details.
	2. Efficiently represents manufacturing systems because it's easily understandable, mostly due to the simulation happening in continuous time.	2. Makes use of differential equations, which means its user must have a background in mathematics.
	3. Is capable of making the whole simulated system react to the effect of an entity, correctly establishing the system's cause-effect relationships.	3. can't simulate complex systems with heterogeneous entities.
	4. It's good for simulating non-linearities, time delays, and feedback loops.	4. Although it is possible to attribute a causal loop diagram to a system, this approach doesn't provide a deep understanding of all the system's feedback loops.

Table 2.1 – continued from previous page

Simulation approach	Advantages	Disadvantages
	5. Can integrate the system's components correctly, getting a holistic view of it.	5. The system's uncertainties aren't normally modeled, so the solutions presented by the model aren't exactly correct.
	6. Generally doesn't require too much data from the system.	6. If the program is built by simulation experts, its users won't be able to understand it due to its complexity.
	7. It's capable of integrating hardware and software components, which makes it capable of providing more in-depth analysis.	
	8. Usually allows quick modeling.	
	9. It's flexible since it applies to many types of systems.	
	Source: Scheidegger et al. (2018) [21].	**Source:** Scheidegger et al. (2018) [21].
Agent-Based Modeling and Simulation (ABMS)	1. Can easily model complex systems in which entities interact with each other.	1. It's hard to model agents that depend on subjectiveness, for example, humans.
	2. It's great at analyzing adaptive systems.	2. The simulation of many individual agents can be time-consuming for computers, which means that modeling systems with lots of agents can take a lot of time.

Table 2.1 – continued from previous page

Simulation approach	Advantages	Disadvantages
	3. ABMS is good at warning about possible unexpected events that can happen throughout the system's execution and at displaying how they affect it.	3. Requires previous programming knowledge in order to use the main existing tools.
	4. Good at simulating non-linear behaviors.	4. Sometimes it is necessary to make assumptions about the behavior of some of the system's agents.
	5. Doesn't require a deep mathematical background.	5. Analyzing the results of many agents can be overwhelming, the user must know where to look for the information he pretends to obtain.
	6. Since the interaction between agents can be easily analyzed through ABMS, it becomes easier to make decisions about the system.	6. Changes to the model can make it even harder to analyze because of the possible non-linearities.
	7. Can make use of machine learning techniques.	7. Routine and deterministic models can take more effort to implement with ABMS than with other approaches.
	8. Randomness can be managed and attributed to specific parts of the system.	
	9. Unlike other approaches, ABMS models can be easily adapted since they allow the user to, for example, apply changes at a local level, by changing the way a specific agent behaves.	
	Source: Scheidegger et al. (2018) [21].	**Source:** Scheidegger et al. (2018) [21].

Each of the simulation approaches has its own orientation [21]. DES works at discrete times and is process-oriented since it can model systems in detail [21]. SD works in continuous time and is system-oriented because it focuses on modeling the system's observable entities [21]. ABMS works in continuous and discrete time and is individual-oriented since it focuses on modeling its entities, the relationships between them, and their behaviors [21].

2.2.2 Digital Twins

A Digital Twin (DT) is a simulation and a dynamic model that uses external information like sensor data, physical models, etc. to imitate the behavior of the system it represents (its twin) [31, 32]. Many times DTs are made by creating a 2D or 3D model of the physical system intended to be simulated, which allows better visualization of the way the actual system is supposed to work when suggested to the same conditions as its DT.

The use of DTs allows for an improvement in the design and diagnosis tasks since it is not necessary to waste physical resources to perform them [33–35]. In the industry sector, DTs can be used, for example, in manufacturing, which usually occurs through the creation of a DT of the physical system in order to analyze its behavior and perform a diagnosis [33].

2.2.3 Commissioning of manufacturing systems

In the context of manufacturing, commissioning refers to the testing of the production state of a system [36]. Commissioning is necessary to validate the correct operation of a system and to verify the functionality of the code that controls it.

This section introduces the four manufacturing commissioning approaches and explains how each of them works, concluding with a comparison between their characteristics.

2.2.3.1 Virtual Commissioning

A common definition of Virtual Commissioning (VC) in the manufacturing industry is the testing of manufacturing controller code through the use of a real or simulated controller, such as a real or virtual Programmable Logic Controller (PLC), to operate a virtual simulation of the real manufacturing system [37–39]. However, other sources define VC broader a broader technology, stating that it encompasses the testing of a full system rather than just its controllers [40].

A lot of companies still don't build their manufacturing systems' modules taking the whole system into account, which many times translates into an occurrence of commissioning errors that can take a lot of testing time to resolve [38]. This problem is increasing

because the manufacturing systems are suffering more frequent alterations due to the production of products with shorter lifecycles, which results in an increase in commissioning time [38].

Compared to the traditional commissioning method (commissioning the real system with a real controller), the use of VC enables faster and less expensive commissioning since the testing is first done with a simulation of the manufacturing system to resolve all the non-physical issues, and only after is it done with the real system [38], which means the manufacturers can resolve the systems' issues without going through unnecessary delays, downtime, and energy costs [6]. In addition, many risks can be avoided through industrial VC because since the manufacturing system is being tested virtually, a lot of the hardware risks that could potentially exist in physical commissioning are completely avoided [5]. It must be taken into account that energy management and safety assurance are not usually tested during virtual commissioning [38].

There are two types of virtual commissioning approaches: Hardware-in-the-Loop and Software-in-the-Loop, which will be respectively explained in Sections 2.2.3.2 and 2.2.3.3.

2.2.3.2 Hardware-in-the-Loop

In the context of VC, Hardware-in-the-Loop (HiL) refers to the use of a manufacturing system's physical controller to manage a simulated version of that system [38]. This approach allows the manufacturer to completely test the physical controller without the delays and costs of testing the real manufacturing system [38].

2.2.3.3 Software-in-the-Loop

Software-in-the-Loop (SiL) in VC refers to the use of a simulated controller to manage a simulated manufacturing system [38]. Although this approach doesn't test the real manufacturing system or its controller, it has the advantage of avoiding the delays and costs of not only the real system but also the controller itself [38].

2.2.3.4 Other commissioning approaches

In addition, manufacturing systems' commissioning can also be done through two other approaches, the previously described traditional method, which makes use of a real system and a real controller to test the entire plant, and through Reality-in-the-Loop (RiL), which makes use of a simulated controller and the real system to test its physical elements [36, 38]. Since both of these strategies rely on the physical manufacturing system, they come with higher risks and expenses.

Figure 2.4 represents the four commissioning approaches introduced in this section with a PLC as an example of the system's controller.

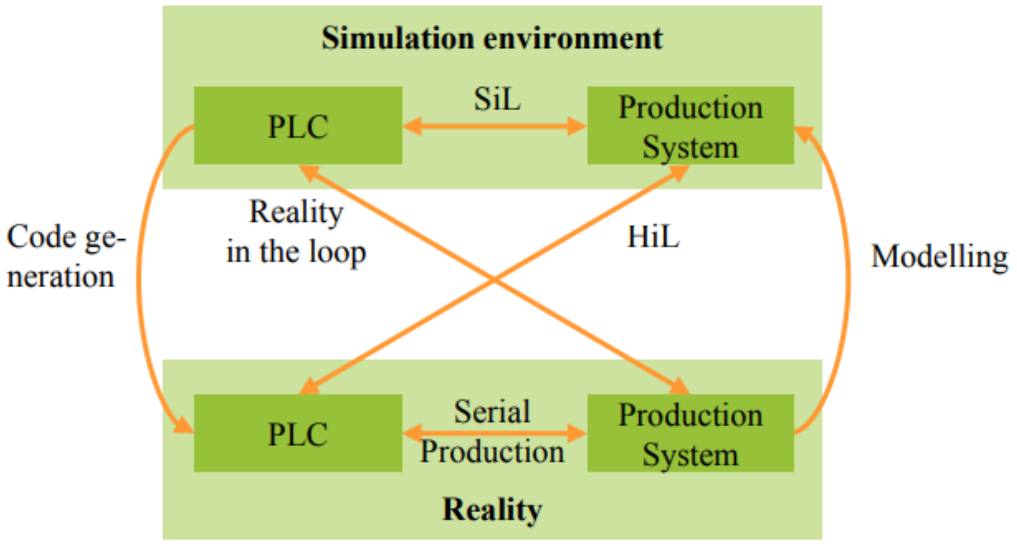


Figure 2.4: Representation of commissioning approaches [38].

2.2.3.5 Comparison between commissioning approaches

The next page Table 2.2 was constructed in order to more easily compare the four introduced commissioning approaches, by not only focusing on evaluating their levels of simulation and reality but also on their benefits and limitations.

Table 2.2: Comparison of commissioning approaches.

Approach	Controller	System	Benefits	Limitations	Sources
Traditional Commissioning	Real	Real	• No need to create simulations for the controller. • No need to create simulations for the system.	• Has a cost associated with the controller. • Has a generally high cost associated with the system. • Usually results in numerous system-related delays. • Requires taking system-related financial risks. • Requires taking system-related safety risks.	• Lechler et al. (2019) [38]. • Soori et al. (2023) [6]. • Skýpala and Ruarovský (2022) [5].
Reality-in-the-Loop (RiL)	Simulated	Real	• The controller can be tested without associated costs. • No need to create simulations for the system. • Can be used to validate the operation of the real system's physical components (e.g. sensors and actuators).	• Requires the creation of a simulation of the controller. • Has a generally high cost associated with the system. • Usually results in numerous system-related delays. • Requires taking system-related financial risks. • Requires taking system-related safety risks.	• Lechler et al. (2019) [38]. • Soori et al. (2023) [6]. • Skýpala and Ruarovský (2022) [5].
Virtual Commissioning					
Hardware-in-the-Loop (HiL)	Real	Simulated	• No need to create simulations for the controller. • The system can be tested without associated costs. • Doesn't have system-related delays. • Doesn't have system-related financial risks. • Doesn't have system-related safety risks.	• Requires the creation of a simulation of the system. • Has a cost associated with the controller.	• Lechler et al. (2019) [38]. • Soori et al. (2023) [6]. • Skýpala and Ruarovský (2022) [5].
Software-in-the-Loop (SiL)	Simulated	Simulated	• The controller can be tested without associated costs. • The system can be tested without associated costs. • Doesn't have system-related delays. • Doesn't have system-related financial risks. • Doesn't have system-related safety risks.	• Requires the creation of a simulation of the controller. • Requires the creation of a simulation of the system.	• Lechler et al. (2019) [38]. • Soori et al. (2023) [6]. • Skýpala and Ruarovský (2022) [5].

2.2.4 Reality-virtuality visualization technologies

This section will introduce the term Mixed Reality (Section 2.2.4.1) and explain the following Reality-virtuality visualization technologies: Augmented Reality (Section 2.2.4.2); Augmented Virtuality (Section 2.2.4.3); Virtual Reality (Section 2.2.4.4).

The technologies about to be introduced all have their own degree of reality and virtuality. In order to evaluate each technology in regard to its reality and virtuality, it is first necessary to establish the environment in which they operate.

These technologies generally use electronic devices to sense the real world, process the virtual world, and display the result to its users, for example, computer vision technologies can be used to sense the real world, a computer can be used to process and model the virtual world and a virtual head-mounted display can be used to present the result to the user.

In the paper *A Taxonomy of Mixed Reality Visual Displays* [41], a virtuality continuum is introduced. This virtuality continuum places the Real Environment (RE) at the left extremity and the Virtual Environment (VE) at the right extremity. Figure 2.5 displays this virtuality continuum.

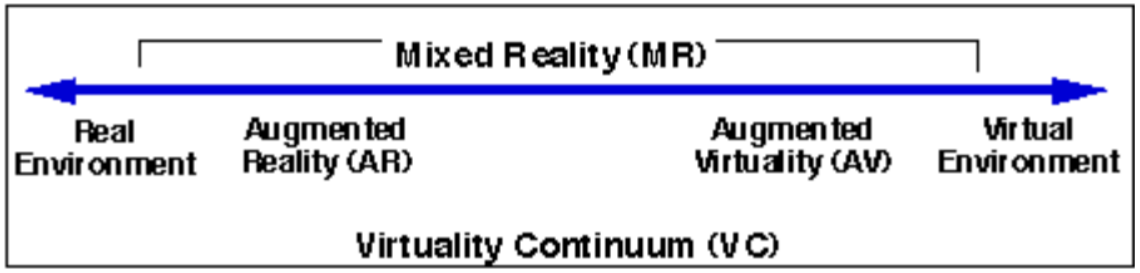


Figure 2.5: Virtuality continuum [41].

RE is an environment solely constituted of real objects, even if these objects are displayed by an electronic device it is still considered a real environment [41].

VE, on the other hand, is an environment composed solely of virtual objects. A good example of such an environment is a graphic simulation [41].

2.2.4.1 Mixed Reality

Mixed Reality (MR) technologies are technologies that merge real objects with virtual objects, therefore operating in a MR environment. A MR environment is an environment composed of both real and virtual objects [41].

2.2.4.2 Augmented Reality

Augmented Reality (AR) is a MR technology that merges the real and the virtual world by presenting virtual objects in an originally real environment [42].

Because in an AR environment, the virtual layer is modeled on top of the displayed physical layer, it is possible to see that this environment is closer to a RE than to a VE, thus being placed closer to the RE in the virtuality continuum (Figure 2.5).

In the industry context, AR is normally referred to as Industrial Augmented Reality (IAR). IAR has many uses in the manufacturing industry, it can contribute in many ways to train workers, design systems, aid in the execution of auxiliary operations, facilitate the consulting of system information, and even aid in the execution of manufacturing tasks [43].

2.2.4.3 Augmented Virtuality

Augmented Virtuality (AV) is a MR technology that also operates in a MR environment but, unlike AR, AV merges the real and the virtual worlds by presenting real objects in an originally virtual environment [44].

Because in an AV environment, the physical layer is displayed on top of the modeled virtual layer, it is possible to see that this environment is closer to a VE than to a RE, thus being placed closer to the VE in the virtuality continuum (Figure 2.5).

Contrary to AR, AV hasn't been as vastly studied in industrial contexts, nor has it been used as much in the manufacturing industry.

2.2.4.4 Virtual Reality

Virtual Reality (VR) is a technology that allows its users to immersively experience a virtual environment [45].

VR isn't usually considered to be a MR technology because it doesn't present any real objects in its resulting environment.

In a VR environment, there is a virtual layer but there is no physical layer. Because of this, VR is many times placed at the virtual end of Milgram's virtuality continuum since it completely immerses its user in a VE [46].

VR has been used in the industry to improve space visibility by not restricting a human's vision field through their posture and position, to observe and improve ergonomy, to provide a better organization of, most commonly, large spaces, and even to evaluate the aesthetics of objects. VR has also been used to allow the storytelling of a product's course to someone in order to help them visualize what happens to that product. It can help in the visualization of digital data and in the communication between disciplines (between people from different backgrounds) because of the great visualization power that VR provides [45].

2.2.4.5 Comparison between reality-virtuality visualization technologies

Table 2.3 was constructed with the objective of facilitating the comparison between the three reality-virtuality visualization approaches introduced in this section. This table

presents the characteristics of each technology by evaluating the environment and its objects as real or virtual and providing a brief explanation of how these interact.

Table 2.3: Comparison of reality-virtuality visualization technologies.

Technology	**Environment**	**Objects**	**Explanation**	**Sources**
Augmented Reality (AR)	Real (RE)	Virtual	AR allows for the visualization of virtual objects in Real Environments.	• Krevelen and Poelman (2010) [42].
Augmented Virtuality (AV)	Virtual (VE)	Real	AV allows for the visualization of real objects in Virtual Environments.	• Regenbrecht et al. (2004) [44].
Virtual Reality (VR)	Virtual (VE)	Virtual	VR allows for a complete virtual experience, in which the environment and its objects are both virtual.	• Berg and Vance (2017) [45].

2.2.5 Evaluation of the main simulation technologies

In order to more generally evaluate the simulation technologies mentioned throughout this section in the context of industry, the main I4.0 design principles should be taken into account.

The article *Simulation in Industry 4.0: A state-of-the-art review* points out 17 of the most important design principles of I4.0, them being: vertical integration; horizontal integration; end-to-end engineering integration; smart factory; interoperability; modularity; real-time capability; virtualization; decentralization; autonomy; optimization; flexibility; agility; service orientation; smart product; product personalization; corporate and social responsibility [20].

Table 2.4 presents an evaluation of many of the main simulation technologies according to the previously presented principles.

Table 2.4: Comparison between simulation technologies according to Industry 4.0 principles [20].

SM approach	I4.0 design principle																
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17
ABMS	●	◑	○	◑	●	●	●	◑	●	●	●	●	●	◑	◑	◑	◑
DES	○	◑	○	○	○	○	◑	◑	○	○	●	●	●	●	○	○	◑
SD	○	◑	○	○	○	○	◑	◑	○	○	●	●	○	◑	○	◑	○
VR	◑	○	○	◑	○	○	●	●	○	○	●	●	○	●	●	●	◑
AR	○	○	○	◑	●	○	●	●	◑	○	●	●	◑	●	●	◑	○
VC	●	◑	◑	◑	◑	◑	◑	◑	○	○	●	●	◑	●	◑	◑	○
PN	●	◑	○	◑	◑	◑	◑	◑	○	○	●	●	◑	○	○	◑	○
AI	◑	○	○	●	◑	◑	●	●	●	●	●	●	◑	●	●	◑	○
DT	●	●	◑	●	●	●	●	●	◑	◑	●	●	◑	●	◑	◑	◑
HS	●	●	◑	●	●	◑	●	●	●	●	●	●	●	●	◑	◑	◑

The symbols mean I4.0 design principle captured (●), partially captured (◑), or non-captured (○) by the simulation approach. The abbreviations are: P1 — Vertical integration; P2 — Horizontal integration; P3 — End-to-end engineering integration; P4 — Smart factory; P5 — Interoperability; P6 — Modularity; P7 — Real-time capability; P8 — Virtualization; P9 — Decentralization; P10 — Autonomy; P11 — Optimization; P12 — Flexibility; P13 — Agility; P14 — Service orientation; P15 — Smart product; P16 — Product personalization; P17 — Corporate and social responsibility; ABMS — Agent Based Modeling and Simulation; DES — Discrete Event Simulation; SD — System Dynamics; VR — Virtual Reality; AR — Augmented Reality; VC — Virtual Commissioning; PN — Petri Nets Simulation; AI — Artificial Intelligence; DT — Digital Twins; HS — Hybrid Simulation;

2.2.6 Conclusions about the necessary technologies

After the analysis made throughout Sections 2.2.3 and 2.2.4, it is possible to conclude which technologies are appropriate for the fulfillment of this project's objectives. With this conclusion, it is possible to search for existing work in the project's context, since this search can be specifically made regarding the most relevant technologies.

2.2.6.1 Commissioning

To choose the most relevant commissioning approaches, the first project's objective, introduced in Section 1.3, should be considered. This first objective is to test the controller code for a new module without having to purchase it.

From Table 2.2, it is possible to notice that this can only be achieved through the Virtual Commissioning (VC) approaches, since they are the only approaches that don't require a real system in order to test a new module's control code.

It is then possible to conclude that the relevant commissioning approaches, in the context of this project, are Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL).

2.2.6.2 Visualization

For the choice of the most relevant visualization approaches, both objectives introduced in Section 1.3 should be analyzed. As concluded in Section 2.2.6.1, the first objective can only be achieved through the simulation of the new module, which means that it will be a virtual object.

The second objective is to see how a new module will be integrated into an existing real system by watching it operate with that system and in that system's environment. The real system, as the name indicates, is located in the Real Environment (RE), which means the new module (object) to be visualized also has to be visualized in the RE.

Table 2.3 informs that the only technology that can represent virtual objects in a RE is Augmented Reality (AR), making it the only relevant visualization technology in the context of this project.

2.2.6.3 Conclusions

In this section, it was possible to conclude that the most relevant technologies for this project are Virtual Commissioning (VC), which includes both Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL), and Augmented Reality (AR). Taking this into account, the next section (Section 2.3) focuses on exhibiting the existing work in the simulation of manufacturing systems through these technologies.

2.3 Existing work in Augmented Reality for Virtual Commissioning and similar manufacturing simulation tasks

This section intends to show and comment on the existing work related to Virtual Commissioning (VC) with Augmented Reality (AR) in manufacturing and other industrial simulation tasks.

Karlsson et al. (2017) [47] proposes a decision support system that allows a user to, through smart glasses, visualize a manufacturing system in operation and its associated information. This allows users to analyze the information of a simulation more easily since it is displayed in AR and in conjunction with the system. Instead of having to analyze raw data or graphics and interpret the results, the user can see how the system behaves in real-time and evaluate if each module is operating correctly in a more direct and practical way.

This solution enables the evaluation of the results of a manufacturing system's simulation, but it doesn't allow a user to visualize the addition of a new module to an already existing manufacturing system or its operation after such integration. The project also doesn't claim to be testing the control code that would be used in a real version of the system, so it isn't a clear VC solution.

Kokkas and Vosniakos (2019) [48] built an application that allows the user to, through their phone, or another mobile smart device, visualize different layouts for a specific manufacturing line. The authors used AR to display virtual modules of a system in an already existing system's environment. The real and virtual modules are able to operate in synchrony to help the user visualize the execution of a complete manufacturing operation.

This project was made to aid in the design of a manufacturing line with the help of AR. For this reason, the studied manufacturing line was controlled through time-based scheduled programming, which the authors recognize to be an approach less viable for coordination than the control through sensors by stating "Collaboration between real and virtual models can be achieved through sensors (including image recognition from cameras) as a replacement of time-base scheduled scenarios in order to achieve more interactive and flexible simulation mode." [48].

In addition, the real module has an integrated controller that was programmed online and offline, which means that if the virtual modules were replaced with real modules, they would be programmed in the same way. For the simulation, these virtual modules were programmed in a different software application and in a different language, which means that this solution doesn't test the code that would be used if the virtual modules were replaced by physical ones. It is then possible to conclude that this solution is not a VC approach and that it only has the purpose of allowing the visualization of different manufacturing system layouts.

Havlíek (2019) [49] uses smart glasses and AR technology to visualize a 3D DT of a system and enable the user to edit its workspace zones (space volume in which the system should operate) through hand commands, executing VC since the system's program that

defines the workspace zones is being tested virtually. This solution allows the VC of production systems using AR, but it only applies to testing individual systems.

This solution was not developed for modular systems or Multi-Agent Systems (MAS) and therefore doesn't allow for the complete VC of a module for a Modular Production System (MPS) (since it doesn't simulate tasks between different modules). It also doesn't allow the visualization of this module in operation with an already existing manufacturing system in AR.

Schumann et al. (2022) [50] develop a way to aid the commissioning of an electromechanical multipoint die cushion of a servo press. The solution uses AR technology to display information and better visualization of the press that is being commissioned by displaying a 3D Digital Twin (DT) of it and displaying relevant information related to it on a tablet. This tablet also displays the needed commissioning steps for the system and waits for the operator to make the necessary adjustments before advancing to the next step.

This project makes use of the real system and not only the virtual one, being, therefore, not exclusively a VC approach. Even though the approach is not fully virtual, it is still a good use of simulation to reduce risks and delays during commissioning. In addition, since the real system is acquired before the testing, there is still a relevant financial cost associated with it. This approach also doesn't account for the interaction between different modules of a more complex industrial system.

Table 2.5 allows for better visualization of the limitations associated with the previously introduced work on the subject of VC and visualization of MPS through AR.

Table 2.5: Limitations of the existing work for Virtual Commissioning (VC) of Modular Production Systems (MPS) and their visualization through Augmented Reality (AR).

Work Reference	**Limitations**
Karlsson et al. (2017) [47]	• Not a clear VC solution since it doesn't claim to use the control code that would be used in a real system. • Doesn't allow the user to visualize the addition of a new module to an already existing manufacturing system.
Kokkas and Vosniakos (2019) [48]	• The system was programmed with time-based scheduled programming, which is not as viable for coordination purposes as programming based on the information from sensors. • The virtual modules were programmed differently from the real modules so the control code being tested is not the same control code that would be used for a real system, which means it isn't used for VC.
Havlíek (2019) [49]	• Doesn't test the interaction between different systems, not being VC for a MPS but just for an individual system. • The tested module is only visualized by itself and not working with other modules, not allowing for the visualization of the addition of a new module to a manufacturing system.
Schumann et al. (2022) [50]	• Doesn't test the addition of a module by testing its DT, it tests the real module and uses its DT to help with its commissioning, so it's not a complete VC approach. • Has a relevant financial cost associated with the need for the acquisition of the real module. • Doesn't allow testing a system composed of multiple modules.

2.4 State of the art conclusions

This chapter introduced the main concepts related to today's industry and to simulation in its context. It also introduced the most relevant technologies that can aid in the execution of this project and established a comparison between them. From this comparison, it was possible to conclude that the most appropriate technologies for the project are Virtual Commissioning (VC) and Augmented Reality (AR).

The chapter continued by presenting the most relevant work in manufacturing on the subject of these technologies and other related subjects. Then, the limitations of the mentioned projects were explored, and there was no project totally suitable for the fulfillment of the objectives initially introduced in Section 1.3 of this document.

Therefore, it is possible to state that no solutions to the issue under study have been discovered and that the most appropriate technologies to build a solution are Virtual Commissioning (VC) and Augmented Reality (AR).

3

Architecture

This chapter proposes an architecture for the solution of the project's problem.

The chapter starts by presenting some important supporting concepts in Section 3.1. Afterward, the project's requirements are deduced from its objectives, in order for the architecture to be constructed, this happens in Section 3.2. The proposed architecture is then presented and explained in Section 3.3, which is followed by Section 3.4, which explains how the system should operate as a whole. Section 3.5 presents some use case scenarios for the system, with the help of diagrams. Finally, Section 3.6 makes a summary of the appropriation of the proposed architecture in the project's context.

3.1 Supporting Concepts

This section introduces concepts that will be mentioned in future sections. It introduces and explains server-client communication, Industrial Personal Computer (IPC), and the Unified Modeling Language (UML).

3.1.1 Server-Client Communication

When communicating through the network between different devices, it is common to define some devices as servers and others as clients or to specify the part of the code responsible for this communication as being a server or a client.

A server refers to a device that provides a service to a client. A client is a device that starts the communication with a server.

Usually, a server receives a request from a client and executes an action based on the client that sent the request or on the information that it sent. When the server has completed the action associated with this request, it sends a response back to the client, informing it that the execution has been completed and sending additional information if necessary. Figure 3.1 displays the communication between a server and three clients.

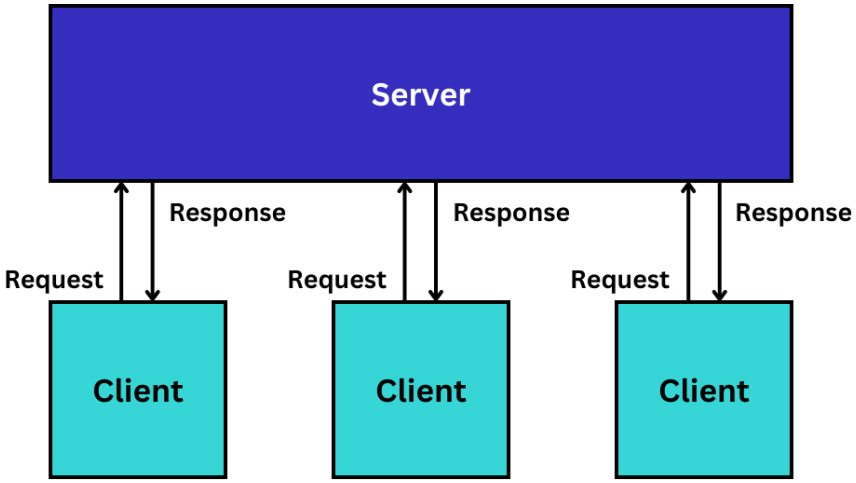


Figure 3.1: Server-Client Communication.

A simple example of server-client communication is the consultation of someone's social media profile. The user utilizes a device (computer, or phone for example), which is the client, to request the person's profile information and receive it as a response. The social media server (a computer for example), receives the request, accesses its database to find the necessary information, and sends it as a response back to the client.

3.1.2 Industrial Personal Computers

An Industrial Personal Computer (IPC) is a computer designed for industrial environments. These computers tend to have a high processing power so they can control and even automate industrial systems when necessary. an IPC is usually built to last a long time and resist industrial conditions (temperatures, humidity, etc.).

In addition, IPCs tend to be modular, usually possessing many optional modules capable of different industrial functions. A common module for many industrial systems is a digital Input/Output (I/O) module, which is composed of input and output pins. The input pins convert wire signals into software signals, allowing the IPC to be informed about the state of the module that is connected to these pins. The output pins do the opposite, enabling the IPC to send wire signals to the module and therefore allowing its control.

3.1.3 Unified Modeling Language

Unified Modeling Language is a modeling language that allows the design of systems in a standardized way. This language makes use of diagrams to represent different aspects of a system. There are many types of diagrams, each with their own purpose. Some of the most used diagrams are Class Diagrams, Use Case Diagrams, and Sequence Diagrams.

A Class Diagram divides the different system components into different classes and displays the relationships between them, many times displaying the attributes and methods of each class.

The Use Case Diagrams and the Sequence Diagrams are both used to facilitate the visualization of a system's operation by showing how it operates in a specific use case scenario.

3.2 Requirements

To establish an architecture for the project, it is first necessary to note what is required for it to execute its function. To deduce these requirements, the project's objectives presented in Section 1.3 should be taken into account. Once more, the project has the objectives of:

- Testing the controller code for a new module without having to purchase it.
- Visualizing the integration of a new module into an existing real system by watching it operate with that system in its environment.

From the first objective, which is testing the controller code for a new module without the need to purchase it, it is possible to deduce that it is necessary to control a simulation of the new module, making sure that this code can be used for the real version of the module too (Virtual Commissioning).

The second objective corresponds to the visualization of this new module in operation with the rest of the real system and in its environment. From this objective, it is possible to note that a real system also has to be controlled. For this real system to give and receive data, it is also necessary to convert software data into real information for the system and vice-versa.

It is also possible to note that the real system is in a Real Environment (RE), which means that in order to fulfill the second objective, the new module's simulation also has to be seen in the RE. Since the new module is virtual (due to it being a simulation) and it has to be shown in a RE, it must be presented with the aid of Augmented Reality (AR).

The deduced requirements are presented next in a summarized way. For the project to operate as intended, it must:

- Control the real modules of the system.
- Control the virtual module of the system in the same way that a physical module is controlled.
- Convert software data into real information and vice-versa (Input/Output control).
- Allow the visualization of the virtual module in a Real Environment (RE) through Augmented Reality (AR).

3.3 Proposed Architecture

From the requirements deduced in the last section, an architecture for the project is proposed. This section presents the composition of the architecture, explains the function of each of its blocks, and shows how it should operate.

3.3.1 Architecture Composition

This architecture is composed of a System Controller, an Input/Output Controller (I/O Controller), one or more Physical Modules, and an Augmented Reality (AR) application that has a Virtual Module associated with it.

The System Controller corresponds to an application on a Personal Computer (PC). The I/O Controller is also an application that runs on an Industrial Personal Computer (IPC) and makes use of one or more digital I/O modules to convert information from software data to hardware signals and vice-versa. The system's Physical Modules can output data related to the state of their sensors and/or receive inputs to control their actuators. Finally, the AR application runs on a smartphone and displays a Virtual Module in the Real Environment (RE).

3.3.2 Detailed System Explanation

The proposed architecture can be observed in Figure 3.2, for a situation in which the real system is composed of two modules and the IPC of the I/O Controller has one digital I/O module connected to it. A brief explanation of how the architecture components operate is provided below.

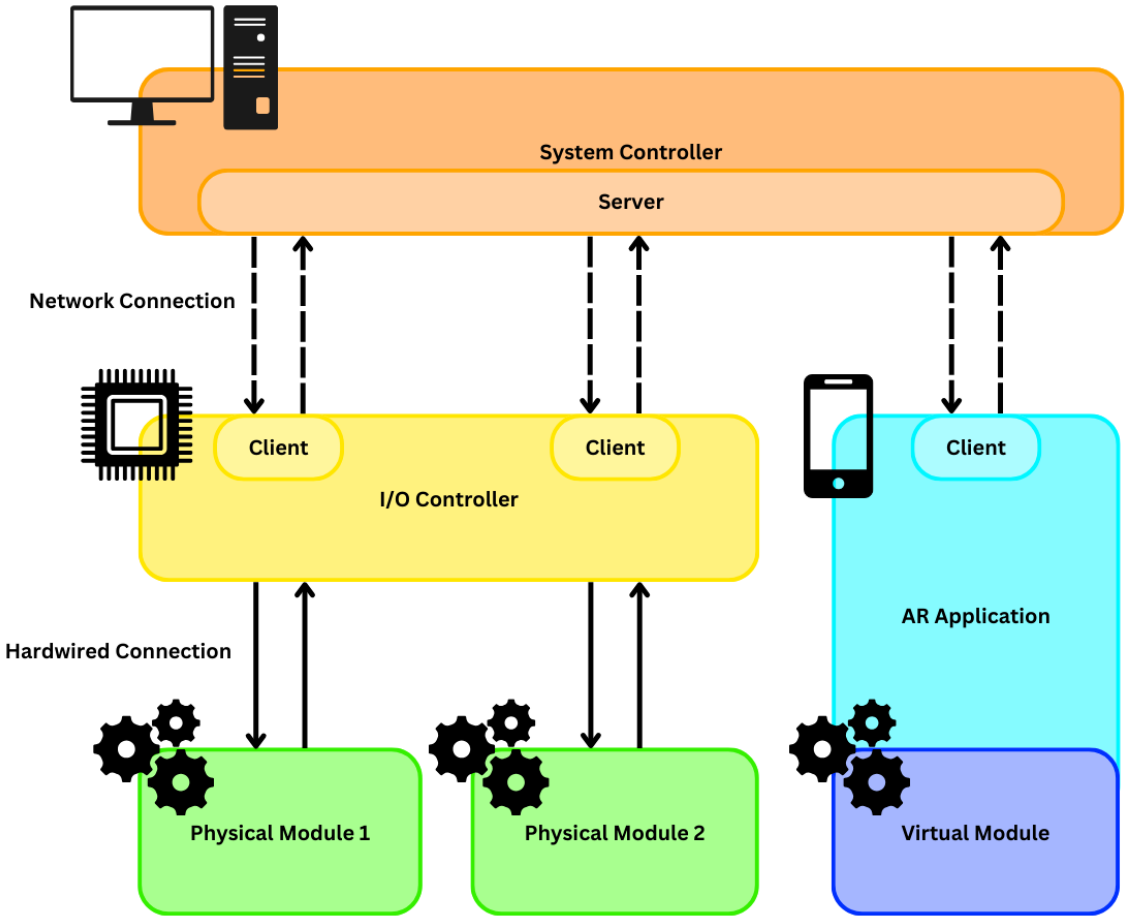


Figure 3.2: Proposed Architecture.

System Controller:

The System Controller is an application responsible for controlling the entire manufacturing system, which includes both the Physical Modules and the Virtual Module.

After being informed by the user of the modules' disposition (the order in which the modules are arranged in the system, or the location of each module), the application is responsible for instructing the system to execute the necessary skills in the correct order, with the objective of having a product go through the necessary processes.

The operations executed by the controller require its ability to receive information about the modules' sensors' state and also send instructions to these modules' actuators, that can be activated and/or deactivated.

Input/Output (I/O) Controller:

The I/O Controller makes use of an IPC and one or more digital I/O modules. The application is programmed on the IPC and it has two functions.

The first function is to receive instructions from the System Controller and use the digital I/O modules to activate or deactivate the appropriate Physical Modules'

actuators.

The second function is to receive information from the Physical Modules' sensors through the digital I/O modules and inform the System Controller of their state.

Physical Modules:

The Physical Modules are controlled by having their actuators activated and deactivated by the signals received from the I/O controller. They also inform the I/O controller of their initial sensor state and of the sensor state changes that occur during their operation.

Augmented Reality (AR) Application and Virtual Module:

The AR Application simulates the addition of another module by displaying a Virtual Module operating in the Real Environment (RE) with the rest of the system. This module is a Digital Twin (DT) of the physical module that the manufacturer wants to add to the system.

The application allows the Virtual Module to be controlled by the System Controller, just as the Physical Modules are. The application receives its instructions from the System Controller. These instructions correspond to the activation or deactivation of the Virtual Module's actuators. The AR Application also informs the System Controller of the Virtual Module's initial sensor state and of the sensor state changes that occur during its operation.

Communication: The system's communication is done through hardwired connections and a network connection. The hardwired connections refer to connections done through physical wires, while the network connection represents a connection done, as the name indicates, through a network.

The System Controller acts as the main server for the network connection since it provides the service of informing the modules about what they need to do. The I/O Controller and the AR application are clients since they make use of the server's service, starting the communication by informing it about the current state of the modules' sensors.

It is to be noted that the proposed architecture takes into account that the system should be modular, for this reason, each module should have an associated and independent control code. Each control code has an associated client that receives instructions from the System Controller. For this reason, the I/O controller has a total of two client instances for the case presented in Figure 3.2, one for each of the Physical Modules it controls, while the AR Application has one since it only controls the Virtual Module.

3.3.3 Class Diagram

To provide a better and more formal understanding of the system, a Unified Modeling Language (UML) Class Diagram was constructed. This diagram isn't related to code so it will have no attributes or methods associated with its classes, it simply presents the relationships between each of the previously mentioned system components. The diagram represents a generic project with an undetermined amount of Physical Modules in the manufacturing system and can be observed in Figure 3.3.

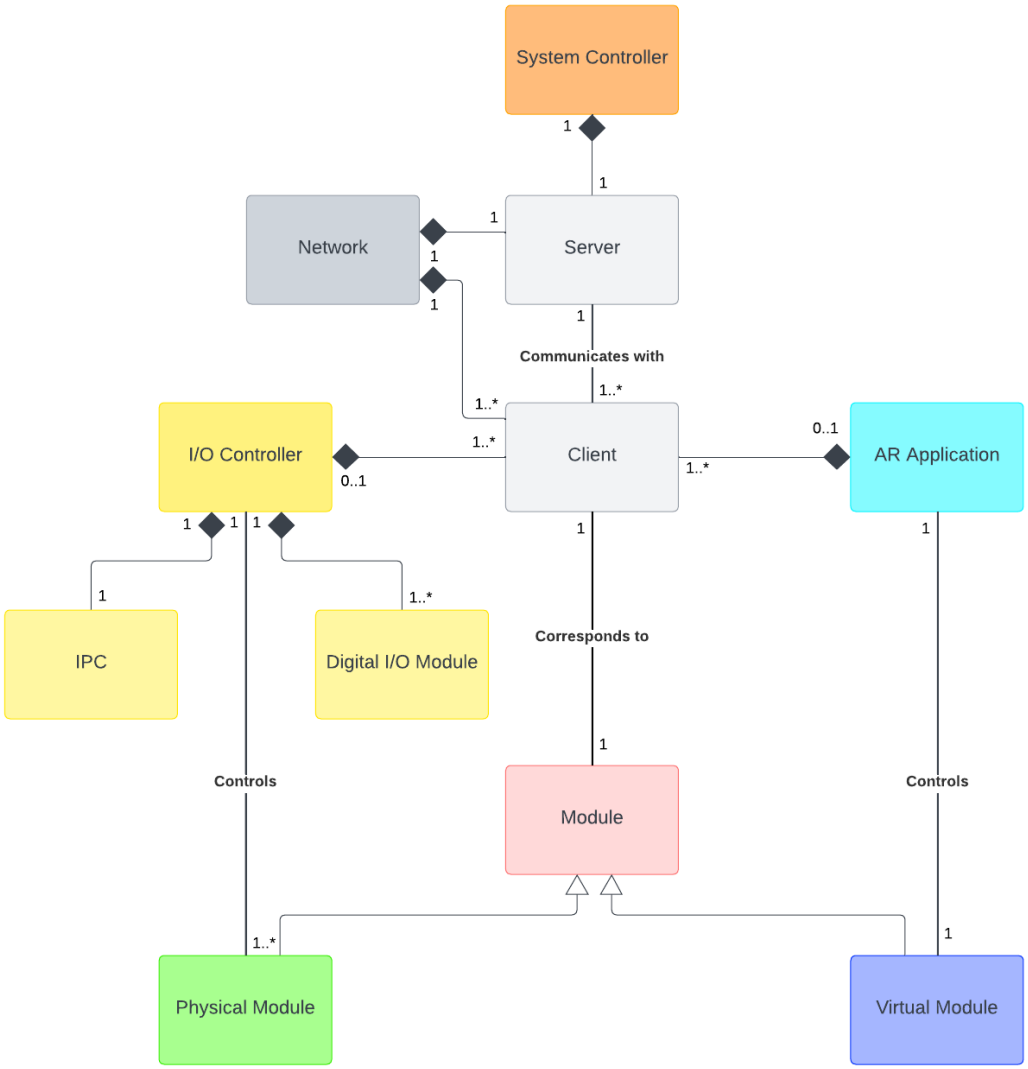


Figure 3.3: Architecture's UML Class Diagram.

From the diagram, it is possible to observe that the system only has one server and that it is part of the System Controller. This server is also part of a network, which must also contain clients. The network's clients can be a part of the I/O Controller or the AR

Application, and each client corresponds to one and only one module. The I/O Controller is composed of an IPC and the necessary amount of Digital I/O modules (depending on the number of Physical Modules and the number of pins of the Digital I/O module). This I/O Controller controls the existing amount of Physical Modules on the system. Similarly, the AR Application controls and displays the Virtual Module of the system.

3.4 Intended Operation

The system operation commences when the user concludes starting the System Controller, the I/O Controller, the Physical Modules, and the AR Application (which prepares the Virtual Module). The user must begin by starting the System Controller, and the Physical Modules can only be started after turning the I/O Controller on, for it to be able to interpret their initial sensors' state.

This user can then submit the order in which the system's modules are arranged. From this submission, the System Controller commands the system to prepare a product through the execution of the necessary skills.

For this control to happen, the System Controller sends the necessary actuator-directed instructions to the I/O Controller or the AR Application when controlling a Physical Module or a Virtual Module, respectively. During the execution of the system, the modules inform the System Controller of any of their sensors' state changes (the Physical Modules do this through the I/O Controller, and the Virtual Module does this through the AR Application).

When the control is concluded and the user is done simulating the system, he can shut down each application and turn the Physical Modules off in any order (the Virtual Module is automatically "turned off" since it is a part of the AR Application).

3.5 Use Case Scenarios

This section intends to clarify how the proposed architecture works by presenting some relevant case scenarios through UML diagrams.

3.5.1 UML Use Case Diagram

A UML Use Case Diagram is presented in Figure 3.4, and it gives a general idea of the solution's functions and possible actions.

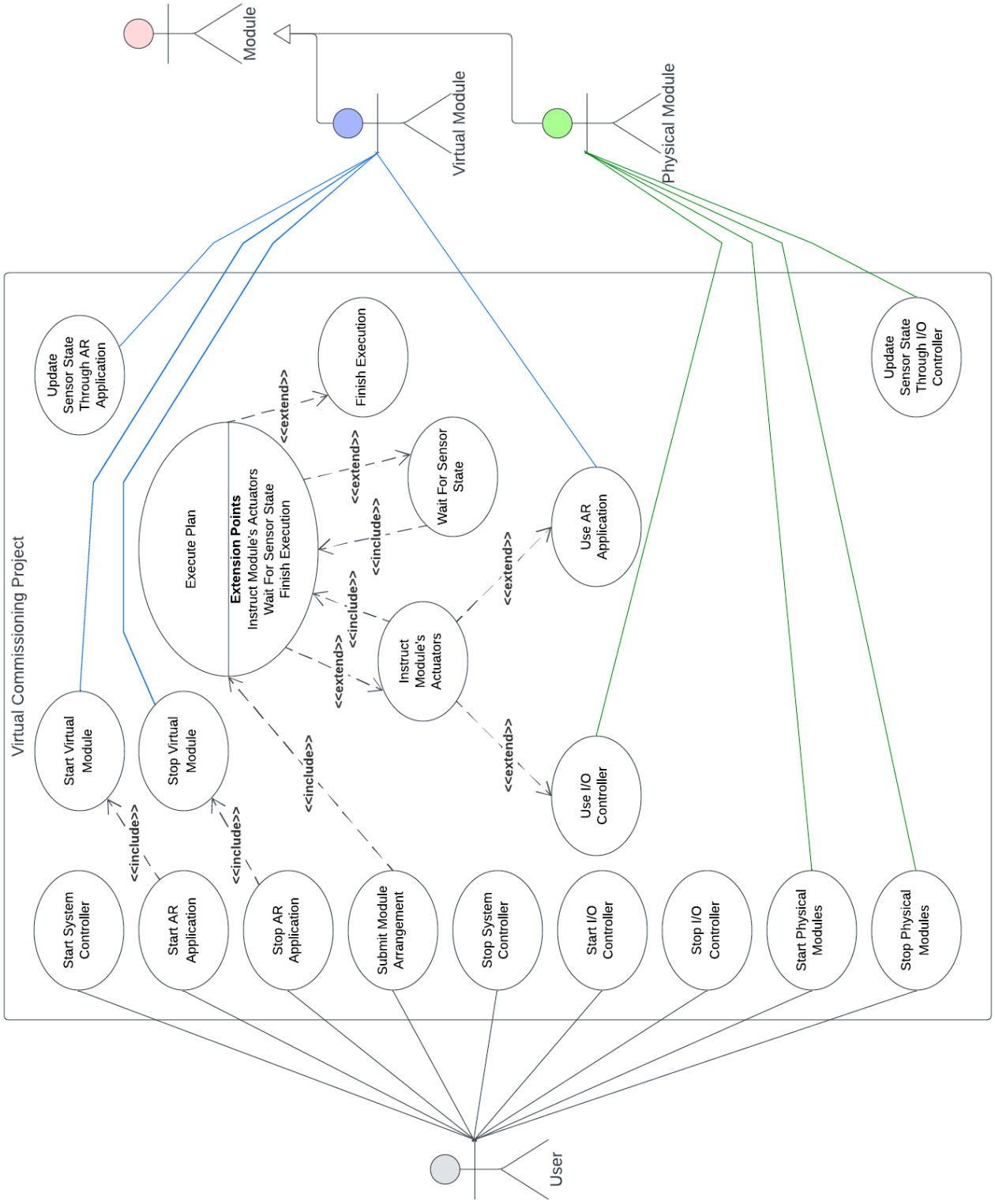


Figure 3.4: Architecture's UML Use Case Diagram.

From this UML Use Case Diagram, it is possible to observe that the user can interact with the system by starting and stopping the different system applications when needed. It is possible to see that starting or stopping the AR Application translates respectively into enabling or disabling the Virtual Module. The user can also power or turn off the Physical Modules.

In addition to this, the user can interact with the System Controller by submitting the module arrangement. After this submission, a plan starts its execution, and it can initiate one of three actions.

If it instructs the module's actuators, it uses the I/O Controller or the AR Application to interact with a Physical Module or with a Virtual Module respectively. After this, the system keeps executing its plan. If it has to wait for a specific sensor state, it will keep trying to execute its plan until the modules communicate the desired state to the system. Finally, if the plan is complete, its execution ends.

It is also possible to observe that the Virtual Module and the Physical Modules can independently communicate their states to the system at any point.

3.5.2 UML Sequence Diagram

A UML Sequence Diagram of the system is also presented. This diagram explains the previously introduced use cases in more detail, by displaying the communication between the different system components. The diagram can be observed in Figure 3.5.

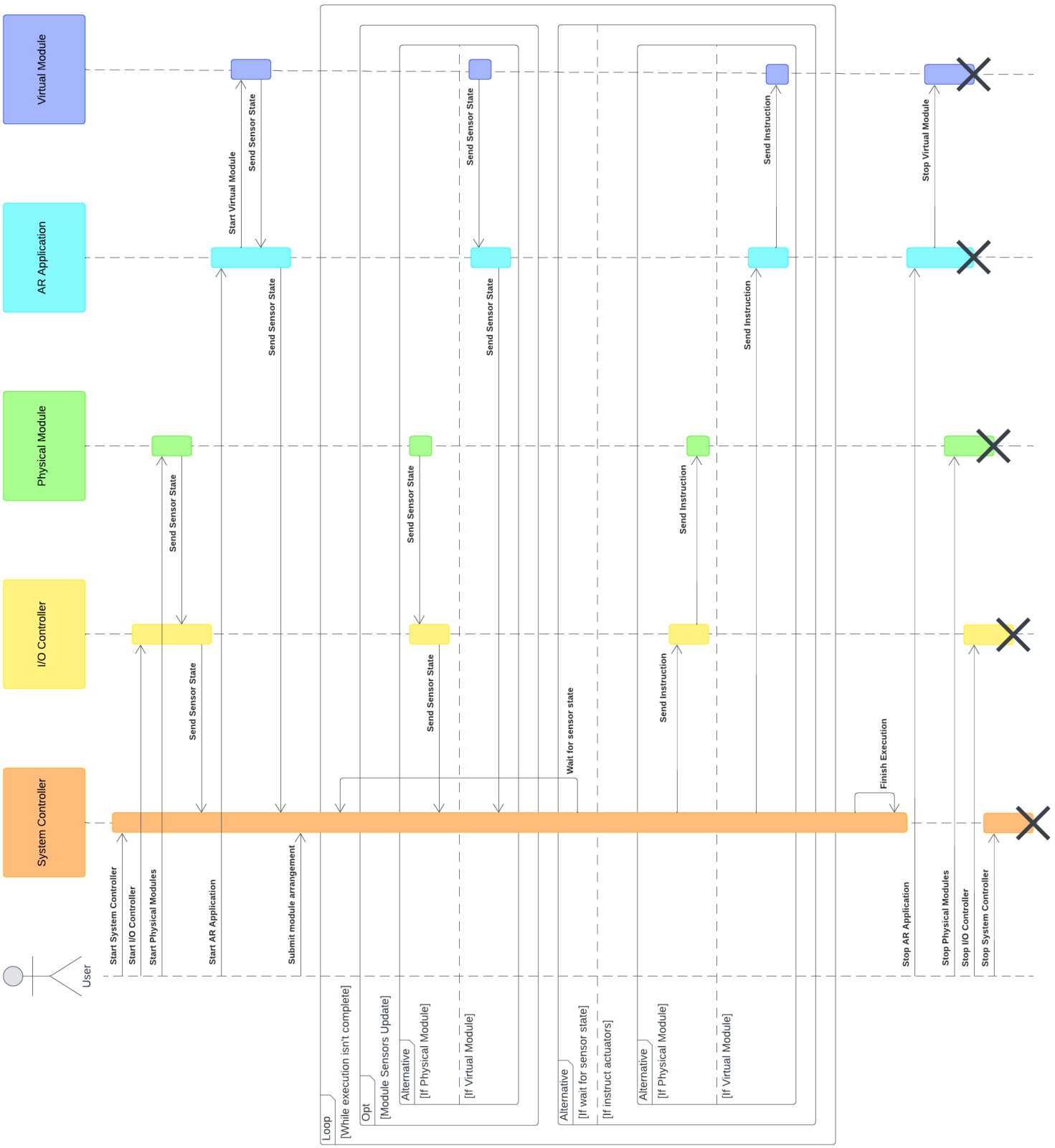


Figure 3.5: Architecture's UML Sequence Diagram.

In the use case example shown through the sequence diagram, the user starts by respectively starting the System Controller, the I/O Controller, the Physical Modules, and the AR Application, which then starts the Virtual Module.

After the modules are initiated they communicate their initial states to the System Controller, through their appropriate associated applications (I/O Controller for the Physical Modules and AR Application for the Virtual Module).

The user then submits the system's module arrangement. The System Controller then executes the necessary actions to operate the system.

When the modules' sensors' state changes, their corresponding applications inform the System Controller of these changes.

If the System Controller is expecting a specific module's sensor state, it keeps waiting for these state changes. In case it wants to instruct a module's actuators, it uses the module's corresponding application to communicate these instructions to it. The System Controller keeps executing these actions when necessary until the desired system operation is concluded.

Afterwards, the System Controller finishes the operation's control and the user stops all the applications and the Physical Modules. The Virtual Module is automatically disabled when the AR Application is stopped.

3.6 Summary

This chapter proposed an architecture for the solution to the project's problem. The proposed architecture intends to allow manufacturers to simulate the manufacturing operations of a Modular Production System (MPS).

The simulated system is composed of one or more Physical Modules and a Virtual Module, and this Virtual Module is controlled in the same way that its corresponding Physical Module would, therefore allowing its Virtual Commissioning (VC).

In addition to this, the solution contains an Augmented Reality (AR) application that displays this Virtual Module operating in the Real Environment (RE) and in conjunction with the Physical Modules. This allows the intended visualization of the addition of a new module to the MPS.

It is then possible to conclude that the proposed architecture fulfills the two project objectives, which once again are, in the context of MPSs:

- Testing the controller code for a new module without having to purchase it.
- Visualizing the integration of a new module into an existing real system by watching it operate with that system in its environment.

This makes it a viable architecture in the project's context. The next chapter will focus on the implementation of this architecture.

4

Implementation

This chapter explains how the previously introduced architecture was implemented for a system composed of two Physical Conveyors, and to which a manufacturer wants to add a third one. It defines the necessary steps for the implementation of the project, presents the devices and technologies that were used for this purpose, and then explains how each of the system's main components that were introduced in the previous chapter were implemented.

4.1 System and Implementation Steps

The architecture presented through Chapter 3 will be implemented in a system composed of two Physical Conveyors, which will be the Real Modules of the project. The module that will be added to this system is another conveyor (the third module of the system is, therefore, a Virtual Conveyor). Additionally, only one digital I/O module will be necessary for the I/O Controller. The implementation of this architecture can be divided into a number of steps.

The implementation steps are:

System Controller Implementation:

- Programming a Graphical User Interface (GUI) for the user to input the conveyor order.
- Programming the System Controller to control the system in accordance with the user's input.
- Programming the network's server-side code.

Input/Output (I/O) Controller Implementation:

- Programming the I/O Controller to react to the System Controller's WebSocket server messages by controlling the real conveyors.

- Programming the I/O Controller to react to the real conveyors' sensor state changes by communicating with the System Controller's WebSocket server (through the network's clients associated with the physical conveyors).

Augmented Reality (AR) Application Implementation:

- Creating a 3D model of a Physical Conveyor (Virtual Conveyor).
- Programming the AR application to allow the visualization of the Virtual Conveyor in the Real Environment through AR.
- Programming the AR application to make the Virtual Conveyor act as a Physical Conveyor (Creating a Physical Conveyor's Digital Twin).
- Programming the network's client-side code for the Virtual Conveyor.

4.2 Technology Stack

This section presents the equipment and technologies used for the implementation of this project, also explaining why these are good choices for the solution in question.

4.2.1 System Controller

The System Controller was implemented as a Java application that runs on a computer. Since the application controls a Modular Production System (MPS), it was programmed through a modular approach, allowing both an easy removal and addition of modules to the system. Therefore Java Application DEvelopment Framework (JADE) was used since it allows a Multi-Agent System (MAS) programming approach.

The application was programmed using IntelliJ IDEA, an Integrated Development Environment (IDE) that facilitates programming in Java and supports User Interface (UI) design.

4.2.1.1 Java Application Development Framework

Java Application DEvelopment Framework (JADE) is a framework that was programmed in Java (an object-oriented programming language) and enables the implementation of MAS through a middleware that satisfies the Foundation of Intelligent Physical Agents (FIPA) specifications [51].

With JADE, it is possible to program a system using a MAS approach, in which the system is divided into different modules (Agents). Usually, every module is associated with an object that extends the JADE's Agent class. The Agents run on Containers, and containers are associated with Platforms. A Platform can have one or many Containers, and a Container can have zero or more Agents. When running the framework on a single machine, it is common to initialize all agents in a Main Container.

Main Containers are the first ones to be started in their corresponding Platform, while the remaining containers connect to them. Unlike normal containers, a Main Container has two default JADE agents in it, the Agent Management System (AMS) and the Directory Facilitator (DF) agents. The AMS agent is the only agent capable of starting or killing other agents and managing the whole platform. The DF agent provides the Yellow Pages service, this service allows the agents to register their services and to find agents that execute specific services [52].

The JADE architecture can be observed in Figure 4.1.

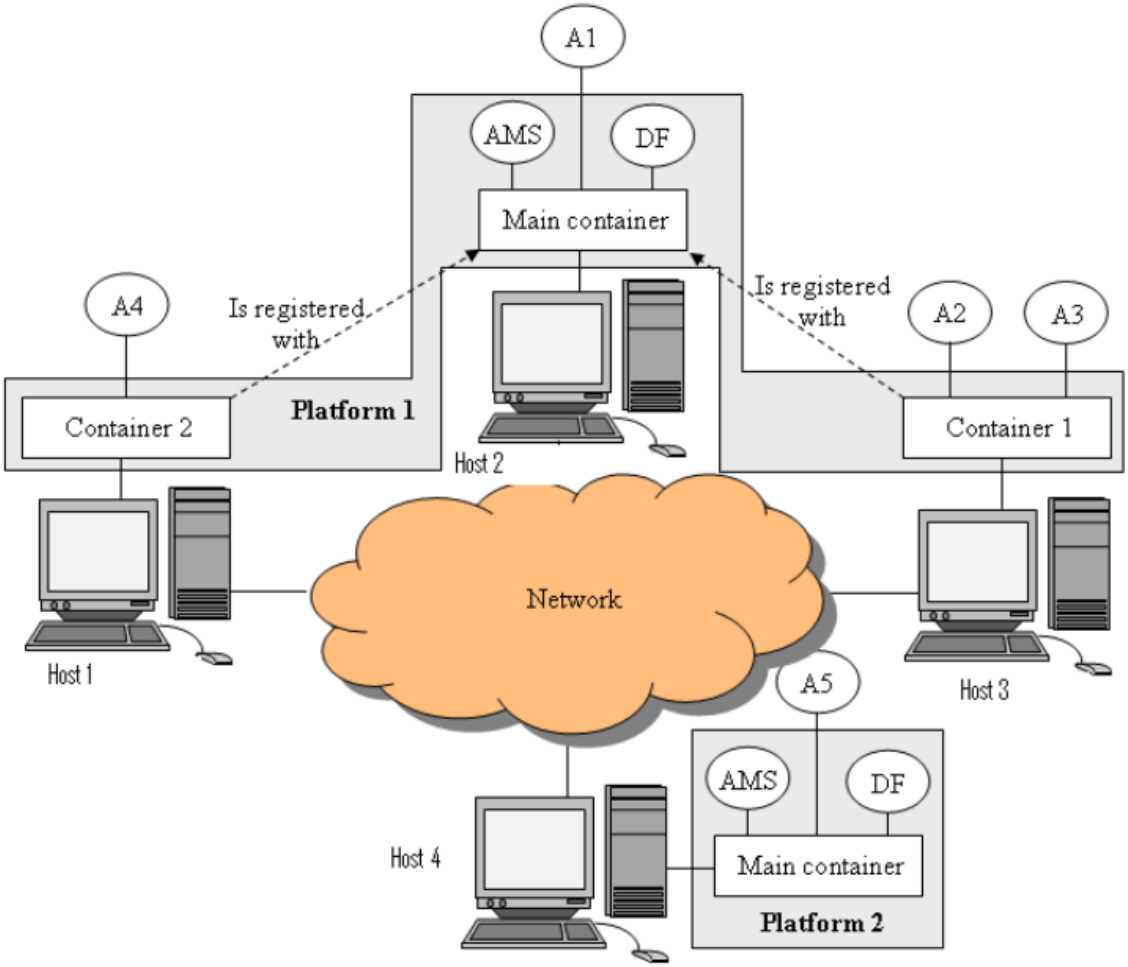


Figure 4.1: The JADE Architecture [52].

In order to communicate, Agents send Agent Communication Language Messages (ACLMessages) to each other. ACLMessage is a JADE class that allows the agent's messages to be standardized. These messages can be sent according to protocols that comply with the FIPA specifications. The ACLMessages have a *performative* attribute that defines the type of message to be sent, which can be set to *REQUEST*, *AGREE*, *INFORM*, *REFUSE*, or many others [53].

The Achieve Rational Effect (AchieveRE) protocol provides a simple communication

method between two agents. In this protocol, there is an interaction between an initiator, who sends the first message, and a responder who receives it. This protocol is used when the initiator wants the receiver to execute an action or service. The initiator starts by sending a *REQUEST* message, to which the responder replies with an *AGREE* message if it is available to execute the action, *REFUSE* message if it can't execute it, or *NOT-UNDERSTOOD* message if it wasn't able to understand the request. If the responder sends an *AGREE* message, it will conclude the communication by sending an *INFORM* message in case the action was well executed, or a *FAILURE* message in case the execution of the action failed [54].

In some cases, the actions to be executed are fast and there is no necessity for the responder to reply with an *AGREE* message immediately followed by an *INFORM* message. In these cases, the responder can reply to the *REQUEST* message directly with an *INFORM* or a *FAILURE* message, having no obligation to send an *AGREE* message previously [54].

Figure 4.2 represents the communication through the AchieveRE protocol.

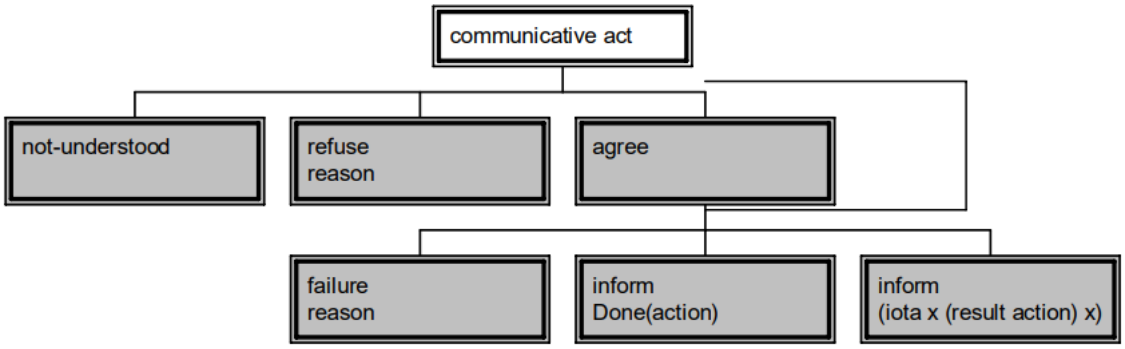
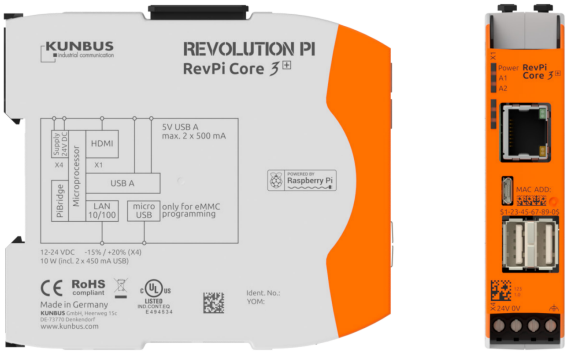


Figure 4.2: Achieve Rational Effect protocol representation [54].

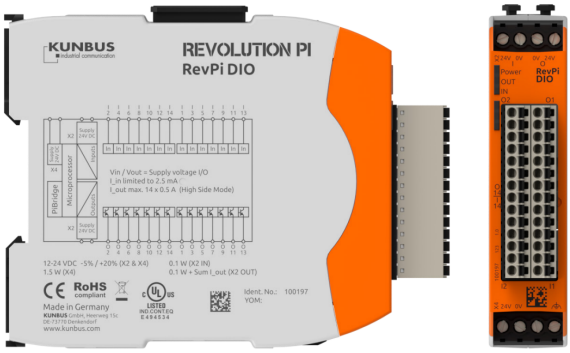
4.2.2 Input/Output Controller

As mentioned in the Architecture chapter (Chapter 3), the Input/Output Controller (I/O Controller) is composed of an IPC and a digital I/O module. The IPC used was the Revolution Pi's RevPi Core 3+, and the digital I/O module was the Revolution Pi's RevPi DIO. These devices can be connected to each other and allow, in conjunction, the connection to a network and to real systems such as the Physical Conveyors that are going to be used to implement the project's solution.

Figure 4.3 shows, respectively, the RevPi Core 3+ device in Subfigure 4.3(a) and the RevPi DIO device in Subfigure 4.3(b).



(a) Revolution Pi Core 3+ [55].



(b) Revolution Pi DIO Module [56].

Figure 4.3: Utilized Revolution Pi modules.

4.2.2.1 Revolution Pi - RevPi Core

The Revolution Pi's RevPi Core is an Industrial Personal Computer (IPC) with a Raspberry Pi Compute Module that uses an adapted Rapbian as its Operating System (OS). This IPC can programmed to execute computational tasks and can connect to a network through its Ethernet port. It also has PiBridge connectors, which enable the connection of this device to other Revolution Pi modules with their functions [57].

The RevPi Core also has the Node-RED software (an open-source flow-based development tool) pre-installed in it, which can be easily used for automation tasks involving the IPC since it has specific RevPi nodes dedicated to it [57].

4.2.2.2 Revolution Pi - RevPi DIO

The Revolution Pi's RevPi DIO Module is a module that can be connected to a Revolution Pi controller module (like the RevPi Core) and that enables the input and output of 24 V digital signals. This module allows the transmission of digital signals between the controller module and a system external to the Revolution Pi devices that are interconnected through the PiBridges [58].

4.2.3 Augmented Reality Application and Virtual Conveyor

The 3D modeling of the Virtual Conveyor was done with Blender since it allows the creation of 3D objects and their exportation.

The Augmented Reality Application was programmed in Unity because it allows the importing of files from Blender and the programming of the Digital Twin (DT) logic through scripting, enabling therefore the creation of the Virtual Conveyor (DT of the Physical Conveyors).

In addition, Unity enables the use of Augmented Reality (AR) and Virtual Reality (VR) and can be deployed to Android and iOS, enabling therefore the creation of an application that allows its user to visualize the Virtual Conveyor in a Real Environment (RE) through a smartphone.

Finally, the application's network interaction can also be programmed through Unity, satisfying the need for communication with the System Controller through the local network.

4.2.3.1 Blender

Blender is a powerful 3D modeling and animation software, which can be used to create virtual models of objects and then export them as files. Blender's 3D modeling functionalities allow its users to shape their 3D modules as they want, apply color and texture to them, simulate the addition of light to their environment, etc.

The software is mostly known for its 3D modeling capability but it also provides tools for 2D modeling, animation, rendering, video editing, and many other operations.

4.2.3.2 Unity

Unity is a game engine but it doesn't only allow the creation of games. Unity allows the rendering of 2D and 3D graphics, scripting, physics simulation, audio integration, AR and VR support, multiplatform deployment capabilities, network interaction, the importing of external models, etc.

Unity can therefore be used for game development, many types of simulation (including AR and VR simulation), modeling, rendering, animation, and many other functions.

Unity is fully prepared to work in conjunction with Microsoft Visual Studio 2019, which is an IDE that allows its users to easily program in various languages, including C#, which is the language used for programming Unity's scripts. For this reason, the Unity scripts were programmed using Microsoft Visual Studio 2019.

A Unity project has an associated Hierarchy, which is the group of objects that compose it. The project's Hierarchy can have one or many Scenes, which are environments in which GameObjects can be placed. These GameObjects can be empty GameObjects, 2D shapes, 3D shapes, lights, cameras, etc. GameObjects can have Components associated with them, like audio sources, animators, colliders, scripts, etc. Another relevant object in Unity is

a Prefab, which is a GameObject that was set as a template and can therefore be reused as necessary. Unity's scripts are programmed in C# and are usually used to program individual classes.

4.2.4 Physical Conveyors

The two Physical Conveyors used were the 24 V Conveyor Belts from Fischertechnik since they work at the same voltage as the Revolution Pi devices used for the I/O Controller, and because they can accurately represent large-scale modular manufacturing conveyor belts due to their sensors and actuators. This is true because the conveyor's sensors and actuators allow the conveyor to know when a product is at its extremities and therefore it can be programmed as an individual module that can execute its associated tasks without requiring information from the other system's modules.

Figure 4.4 shows the Fishcertechnik 24 V Conveyor Belt.

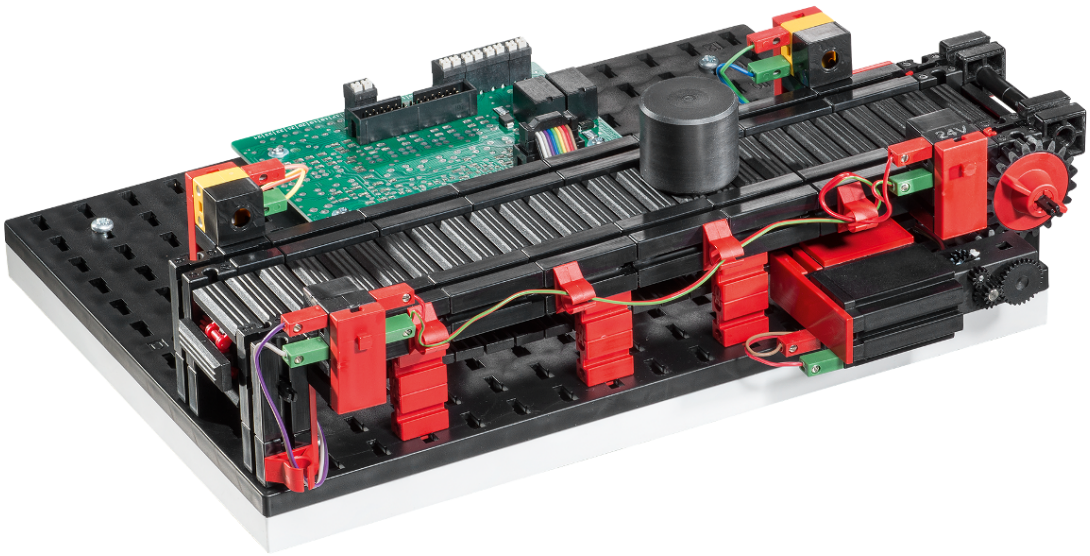


Figure 4.4: Fischertechnik's 24 V Conveyor Belt [59].

4.2.4.1 Fischertechnik 24 V Conveyor Belt

The 24 V Conveyor Belt from Fischertechnik is a small conveyor belt (with a length of 275 mm) that can easily be connected to many other Fishcertechnik modules, making it a great option for small-scale Modular Production Systems (MPS).

The conveyor is designed to be connected to PLCs (Programmable Logic Controllers), which means it can connect to any small-scale circuit pins, including those from a digital Input/Output module of an IPC.

This conveyor can receive input signals to turn on the actuators that control the belt's movement. There is an actuator that makes the belt move to the left and another one that makes it move to the right. In addition to this, the conveyor has a sensor at each of its

extremities. Each of these sensors is connected to an output that can be used to check if they are or aren't detecting a product at the conveyor's extremities.

4.2.5 Network Communication

The network communication was done through a Local Area Network (LAN) and the protocol used for this communication was the WebSocket protocol. The communication between the System Controller and both the I/O Controller and the AR Application should be bidirectional, full-duplex, and happen in real-time.

With a bidirectional communication protocol, the System Controller can send instructions to the conveyors' actuators when necessary, and the conveyors can send their state to the System Controller when they change. This approach is preferable to having the System Controller constantly send request messages to the conveyors while it waits for a specific sensor state.

A real-time and full-duplex protocol allows both the System Controller and the Conveyors to send messages to each other as soon as they intend to do so, resulting in the desired real-time control of the manufacturing systems (a manufacturing system should work as efficiently as possible).

4.2.5.1 WebSockets

WebSockets are a communication protocol that allows real-time, bidirectional, and full-duplex communication between servers and clients.

Unlike the Hypertext Transfer Protocol (HTTP) protocol, WebSocket protocol's clients don't have to send request messages to servers for them to then answer with reply messages. WebSockets allow both the clients and the servers to send messages to each other, and at the same time if necessary.

4.3 System Controller Implementation

The implementation of the System Controller starts with the creation of a new project in IntelliJ IDEA, and the import of the JADE library, which allows the programmer to use its classes. For this, a folder called *lib* is created in the project's directory, and the JADE's library JADE Archive (JAR) file that contains JADE's classes is downloaded and placed inside this folder.

The system was controlled using JADE's classes, directed at Multi-Agent Systems (MAS) programming. The Agent classes created for this project were the following: *GUIAgent*, *ConveyorAgent*, *BoxAgent*, and *ServerAgent*. The communication between these agents was done using JADE's Achieve Rational Effect protocol, described in Section 4.2.1.1. The Main Container's automatically created *DFAgent* class, which provides the Yellow Pages service, also mentioned in Section 4.2.1.1, is also used for this MAS implementation.

After a *ConveyorAgent* instance is created, all of its skills are registered in the *DFAgent* instance's DF (Directory Facilitator).

The application starts with the execution of a JADE command that creates an instance of the *GUIAgent*, the *ServerAgent*, and the necessary instances of the *ConveyorAgent* (one for each conveyor of the system, and an additional one for the transportation between different conveyors).

4.3.1 Graphical User Interface

The Graphical User Interface (GUI) is started by the *GUIAgent* instance at the beginning of the application's execution. It was programmed using IntelliJ IDEA's GUI design tools. Figure 4.5 displays this GUI, which was denominated Conveyor Ordering Interface (COI).

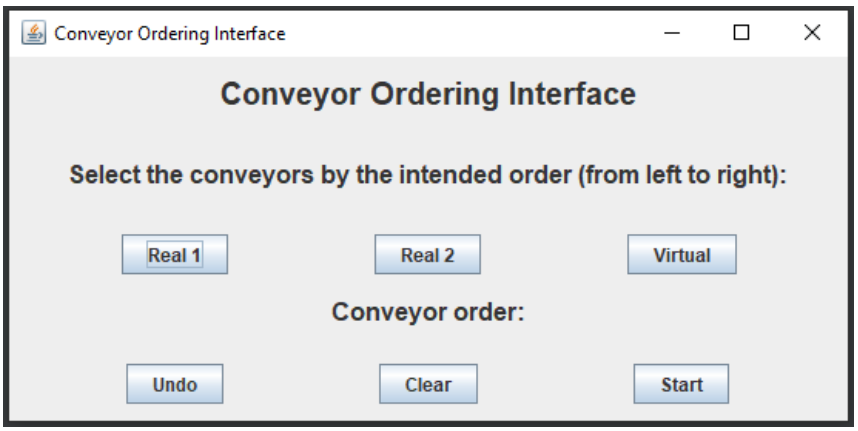


Figure 4.5: Conveyor Ordering Interface.

The COI is composed of six buttons. The top three buttons allow the user to select the conveyors the product will have to go through and the order in which they do so. The user should, therefore, press the buttons by the order in which the product should travel the conveyors. The bottom three buttons allow the user to edit the choice of the conveyor order and submit it. The *Undo* button removes the last conveyor that the user selected from the conveyor order, the *Clear* button allows the user to restart the selection of the conveyors' arrangement, and the *Start* button submits the order selected by the user.

After the user submits the conveyor arrangement, from left to right, the list of skills that the system should execute to move the product as intended is created by the *GUIAgent*. The system should transport the product from left to right and then from right to left. A *BoxAgent* instance is then created with this skill list as its attribute. After this instance is done executing its skills, the user can add a new product and submit the same or another conveyor order in which the conveyors are rearranged. The user can repeat this for as many products as he wants and conclude by shutting down the application.

4.3.2 Control Code

The two programmed JADE Agents responsible for the control of the manufacturing system are the *BoxAgent* and the *ConveyorAgent*. These Agents communicate with each other to inform the WebSocket server about the messages that it has to send to correctly control the modules of the system.

The four instances of the *ConveyorAgent* class created by the JADE command that starts the application are, as mentioned before, one for each conveyor and an additional one for transportation between conveyors. These instances were denominated: *Real1*, *Real2*, *Virtual*, and *Transport*.

4.3.2.1 Box Agent

The *BoxAgent* instance starts by searching the DF for a *ConveyorAgent* that can execute the first skill from its skill list. Afterward, the *BoxAgent* sends a *REQUEST* message to this *ConveyorAgent* through the method. The content of this message is set to the skill's Identification (ID) if the *ConveyorAgent* is not the *Transport* instance. In case it is the *Transport* instance, the *REQUEST* message's content must also include the name of the conveyor from which the product has to move and the name of the conveyor to which the product has to move.

The *BoxAgent* then receives a response message of the *AGREE* type if the *ConveyorAgent* is available, or of the *REFUSE* type if it is occupied. In case a *REFUSE* message is received, the *BoxAgent* goes back to searching for a *ConveyorAgent* that can execute the necessary skill.

When the reply received is an *AGREE* message, the *BoxAgent* waits for the *ConveyorAgent* to execute the necessary skill and receives an *INFORM* message from it that indicates the end of the skill execution.

The *BoxAgent* then goes back to searching for a *ConveyorAgent* that can execute its next skill. When all the skills of the *BoxAgent* instance's skill list have been completed, this Agent gets destroyed.

4.3.2.2 Conveyor Agent

The *ConveyorAgent* instances start their execution by registering their skills in the DF. Throughout their execution, they communicate with one or more instances of the previously introduced *BoxAgent*.

When a *ConveyorAgent* instance receives a *REQUEST* message from a *BoxAgent*, it responds with a *REFUSE* message if it is occupied, and if it isn't, then it becomes occupied and replies with an *AGREE* message. It then processes the content of the message and executes the corresponding skill. The instance then changes its status to unoccupied and sends an *INFORM* message to the *BoxAgent* to inform it that the skill has been executed.

In general, for a *ConveyorAgent* to execute a skill, it makes use of the *ConveyorLibrary* and *SkillLibrary* classes to communicate with the server-side code by verifying the conveyors' sensor state and informing it about the instructions that have to be sent to the conveyors. This is done the necessary number of times until the skill has been completed. The two mentioned classes will be explained in more detail in sections 4.3.2.3 and 4.3.2.4.

The *ConveyorAgent* instance is active until the closure of the application and executes the previously mentioned actions as many times as necessary.

4.3.2.3 Conveyor Library and Skills

These *ConveyorAgent* instances make use of a *ConveyorLibrary* class. This class allows a *ConveyorAgent* to check what its skills are and to start the execution of a specific skill.

The *Real1* instance of the *ConveyorAgent* class has the skills *SK_REAL1_LEFT* and *SK_REAL1_RIGHT*, respectively responsible for moving a product to the left and to the right, along the corresponding Physical Conveyor (from one sensor to the other). In a similar way, the *Real2* instance has the skills *SK_REAL2_LEFT* and *SK_REAL2_RIGHT* (associated with the other Physical Conveyor), while the *Virtual* instance has the skills *SK_VIRTUAL_LEFT* and *SK_VIRTUAL_RIGHT* (associated with the Virtual Conveyor). The *Transport* instance has the skills *SK_MOVE_LEFT* and *SK_MOVE_RIGHT*, respectively responsible for moving a product to the left or to the right, between two different conveyors (from the final sensor of the starting conveyor to the first sensor of the last conveyor).

4.3.2.4 Skill Library

To start the execution of a skill, the *ConveyorLibrary* makes use of a *SkillLibrary* class. This class is responsible for the execution of the *ConveyorAgent* skills. The execution of the skills is done by consulting the WebSocket Server class (*FactoryServer*) for the state of the conveyor's sensors and instructing it to send the necessary messages to the WebSocket clients to control the manufacturing system.

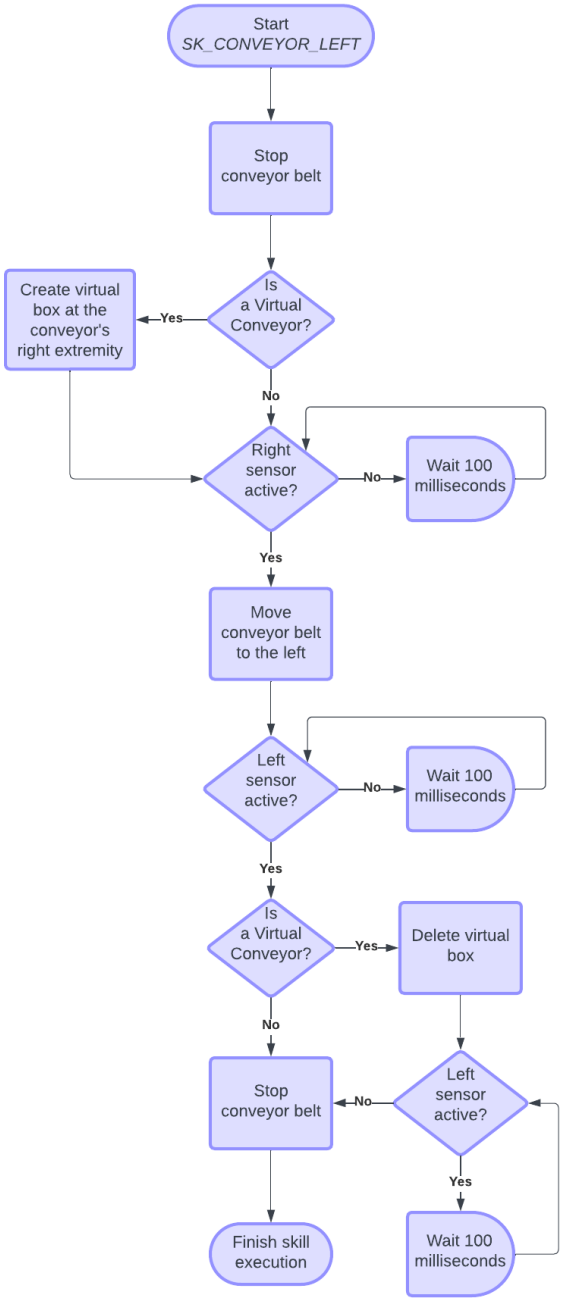
The skills responsible for moving a product to the left along a Physical Conveyor work in the following way. The conveyor belt is first instructed to stop, just to make sure that isn't moving when it shouldn't. The state of the conveyor's right sensor is then checked every 100 milliseconds until it is active. When the sensor is active, it means that the product is at its initial position (in front of the right sensor). The conveyor is then instructed to move its belt to the left. Finally, the conveyor's left sensor is also checked every 100 milliseconds until it is active. This sensor is active when the product has reached the left extremity of the conveyor, which means its transportation has been completed and the skill must finish its execution by instructing the conveyor belt to stop.

The skills responsible for moving a product to the right along a Physical Conveyor work in a similar way. Instead of starting by checking if the product is on the right and ending by checking if it's on the left, the system starts by checking if the product is on the left side and ends by checking if it's on the right side. In addition to this, since the

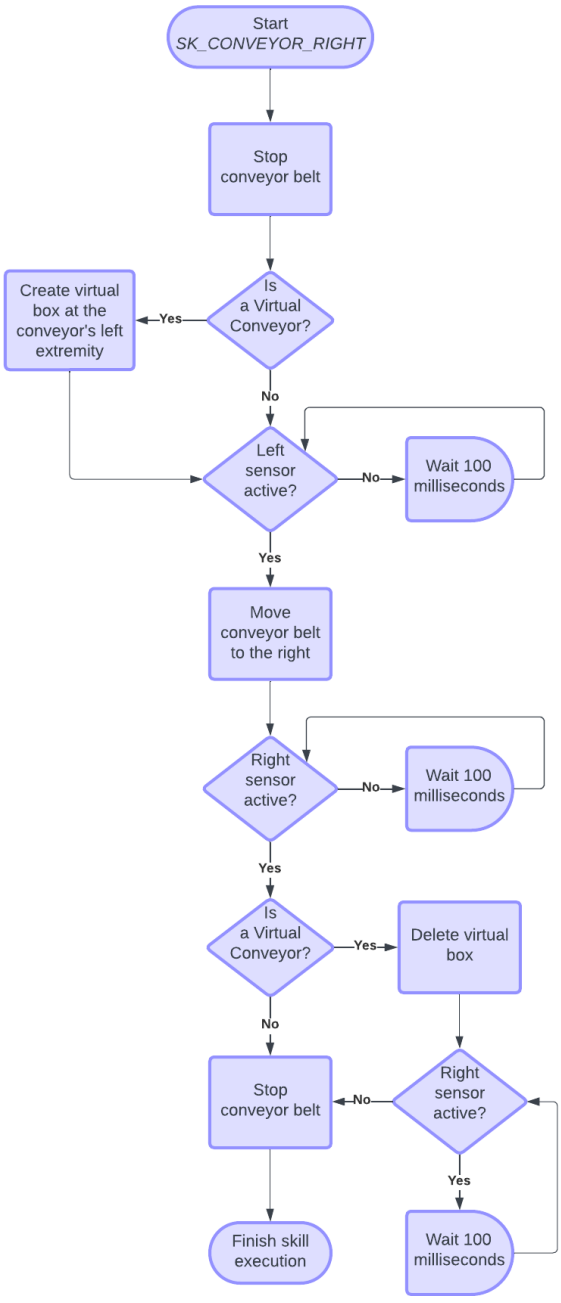
product has to be moved to the right, the conveyor belt is instructed to move to the right instead of to the left.

For the Virtual Conveyor, since it can't interact with the real product, some additional steps must be executed. After the first instruction for the conveyor belt to stop, the conveyor is instructed to make a virtual box appear at its correct starting extremity (right extremity for *SK_VIRTUAL_LEFT* and left extremity for *SK_VIRTUAL_RIGHT*). At the end of the movement skills execution, before the conveyor is stopped, it is instructed to delete this box and the sensor of its end extremity (left extremity for *SK_VIRTUAL_LEFT* and right extremity for *SK_VIRTUAL_RIGHT*) is checked every 100 milliseconds until it is inactive, to assure that the skill only advances when the virtual box has been deleted.

Figure 4.6 presents diagrams that provide better visualization of the logic for the execution of these skills. Figure 4.6(a) displays a diagram for the skills responsible for moving a product to the left through a conveyor, while Figure 4.6(b) presents a diagram for the skills that move a product to the right through a conveyor.



(a) Logic of the skills that move a product to the left through a conveyor.



(b) Logic of the skills that move a product to the right through a conveyor.

Figure 4.6: Logic of the skills responsible for moving a product through a conveyor.

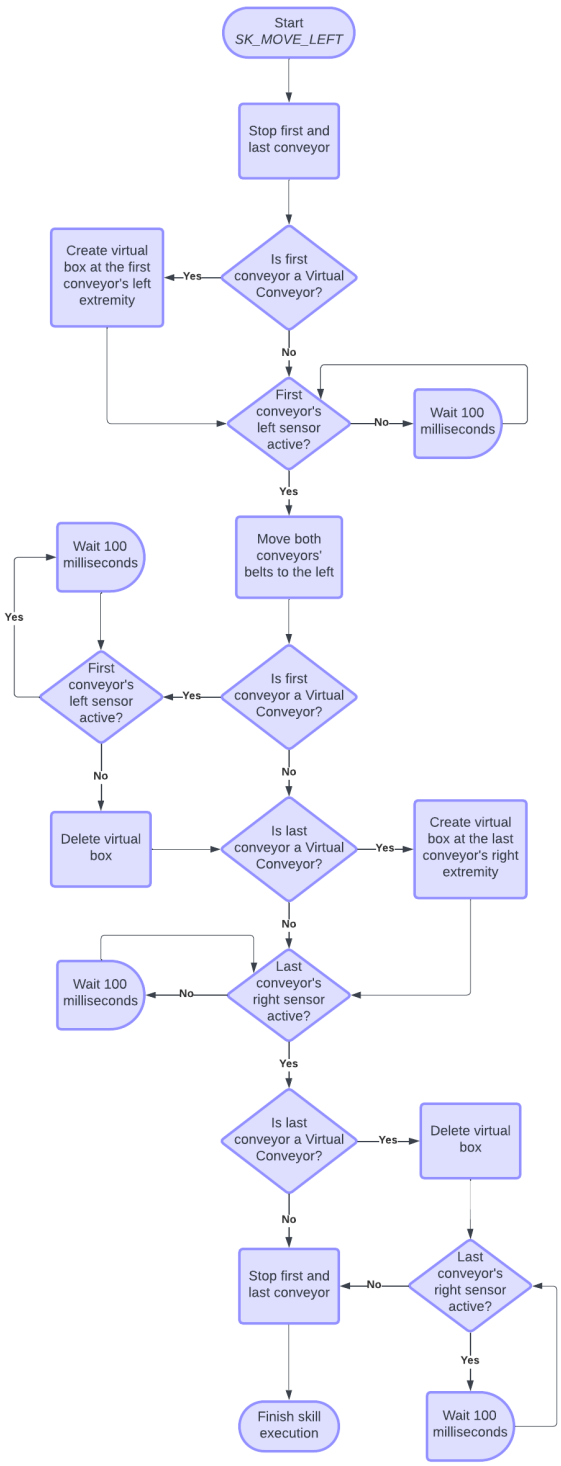
The skills of the *Transport* instance of the *ConveyorAgent* class work in a different way because they make use of more than one conveyor.

For the skill that makes a product transition between two conveyors to the left (*SK_MOVE_LEFT*), both the conveyors start by being instructed to stop their belts, just to make sure that they aren't moving at the start of the skill's execution. If the conveyor in which the box starts is a Virtual Conveyor, a virtual box is created at its left extremity. The

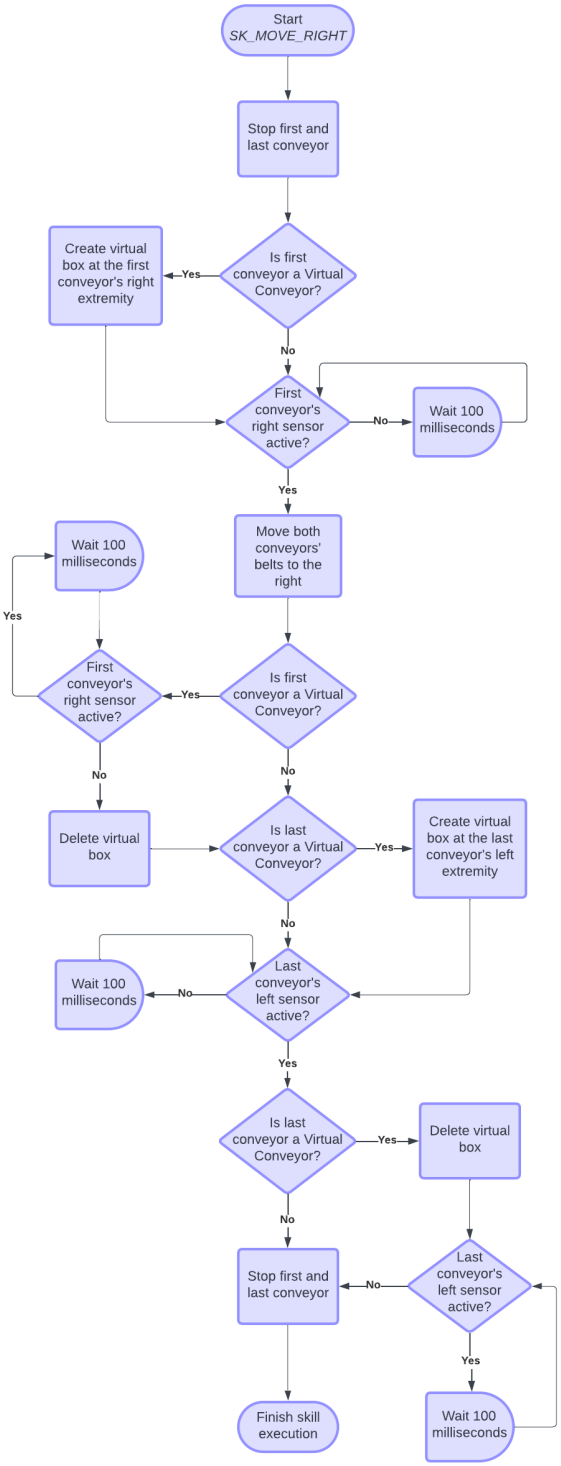
left sensor state of the first conveyor that the box goes through is then checked every 100 milliseconds until it is active. When this sensor is active, the box is ready to be transported, so both conveyors are instructed to move their belts to the left. Afterward, if the conveyor in which the product started is virtual, the state of its left sensor will be checked every 100 milliseconds until it is inactive and the previously created virtual box will be deleted. Then, if the conveyor to which the product goes is a Virtual Conveyor, a virtual box is created at its rightmost extremity. The right sensor of the conveyor to which the product must be transported is then checked every 100 milliseconds until it is active and, if the conveyor to which the product goes is virtual, the virtual box is deleted and the conveyor's right sensor is checked every 100 milliseconds until it is inactive, to make sure that the box has been deleted. Finally, the skill execution ends by stopping both conveyors.

The skill that transitions a product between conveyors to the right (*SK_MOVE_RIGHT*) works in a similar way. Every instruction related to the left side of the conveyor in which the product starts is replaced by the same instruction but regarding its right side, and every instruction related to the right side of the conveyor to which the product goes is replaced by the same instruction but regarding its left side.

Figure 4.7 displays diagrams that allow the visualization of the logic for the execution of these skills. Figure 4.7(a) shows a diagram of the skill that transports the product to the left between two conveyors (*SK_MOVE_LEFT*), while Figure 4.7(b) presents a diagram of the skill responsible for transporting the product to the right between two conveyors (*SK_MOVE_RIGHT*).



(a) Logic of the skills that transport a product to the left between two conveyors.



(b) Logic of the skills that transport a product to the right between two conveyors.

Figure 4.7: Logic of the skills responsible for transporting a product between two conveyors.

4.3.3 WebSocket Server

To implement the WebSocket server, the *Java WebSockets* library was imported into the project by the addition of its JAR file to the previously mentioned lib folder. In addition, the Simple Logging Facade for Java (SLF4J) Simple and Application Programming Interface (API) libraries were also imported since the *Java WebSockets* library makes use of it for logging purposes.

The *ServerAgent* creates an instance of a *FactoryServer* class, which is the WebSocket server's class. A server is then created at the System Controller's device network address and port number 8080.

The server communicates with each conveyor's associated WebSocket client through a specific endpoint. One of the real conveyors' client communicates through the endpoint */real1*, the other physical conveyor's client communicates through the endpoint */real2*, and the virtual conveyor's client communicates through the */virtual* endpoint.

It must be noted that the controller doesn't communicate directly with these conveyors but with the WebSocket clients associated with the controller of each of them.

The server receives messages every time the state of a conveyor's sensors changes, and it stores the state of each sensor of each conveyor in a different variable. This server is also responsible for receiving the instructions that the controller code wants to give to the conveyors and sending these to them through the appropriate endpoints.

The server can be instructed to send the messages *move_right*, *move_left* and *stop*, which respectively inform the client that its conveyor should move to the right, to the left, and stop moving. It can also receive the messages *create_box_left*, *create_box_right* and *delete_box*, which respectively indicate that a virtual box should be created at the client conveyor's left extremity, created at the conveyor's right extremity, and be deleted. These messages are sent through the endpoint that corresponds to the conveyor that should receive it.

The messages that can be received by the server as an indication of the conveyors' sensor state are *left_sensor_on*, *left_sensor_off*, *right_sensor_on* and *right_sensor_off*, that inform the server, respectively, that the conveyor's left sensor was activated, the conveyor's left sensor was deactivated, the conveyor's right sensor was activated, and the conveyor's right sensor was deactivated. It is possible to identify the conveyors to which these sensors belong by observing the endpoint from which the messages were received.

4.3.4 System Controller Implementation Logic

A general presentation of the controller code logic can be observed in Figure 4.8 through a diagram, that provides a visual representation of the System Controller's operation.

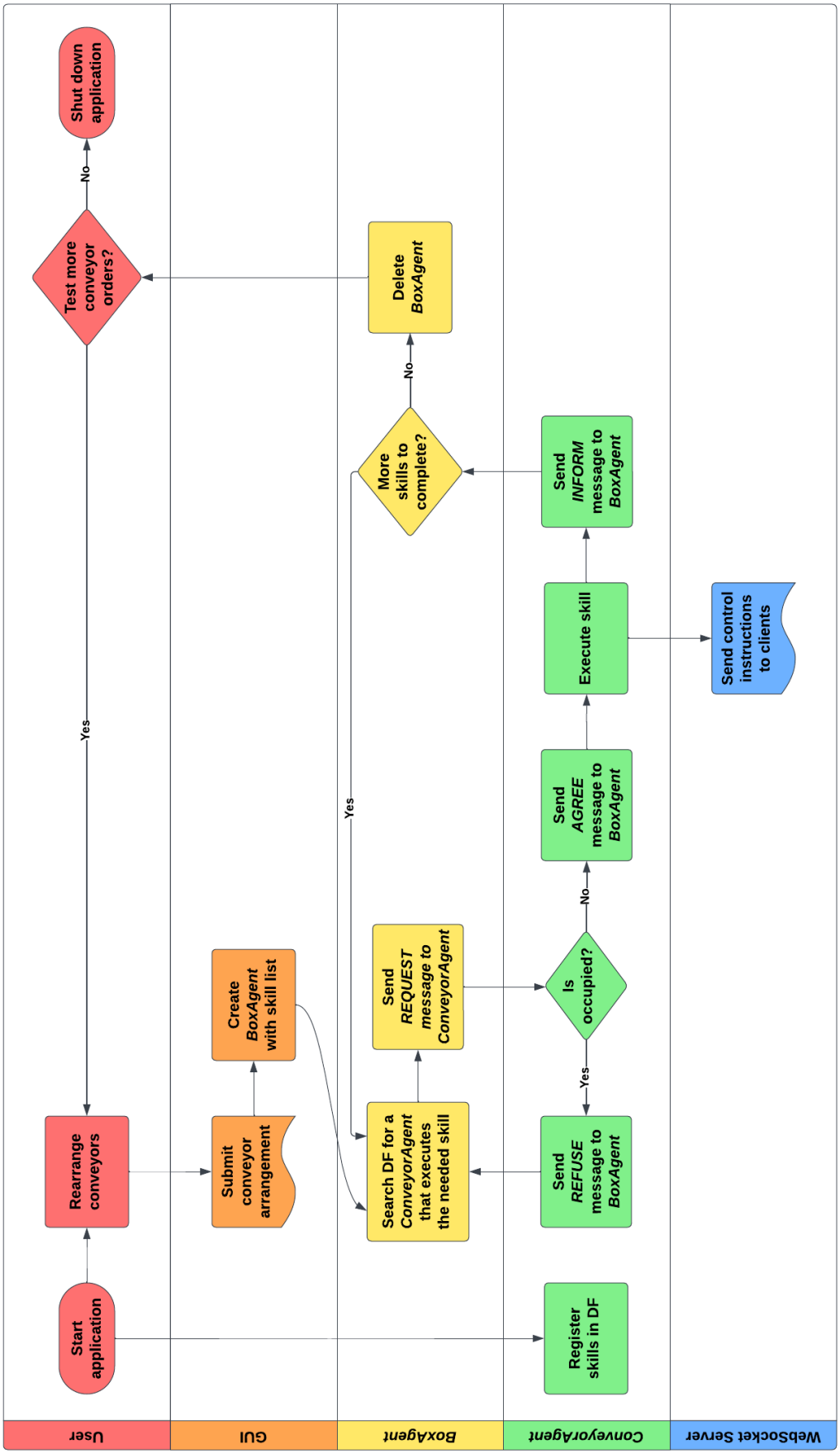


Figure 4.8: Diagram for the System Controller's implementation logic.

4.4 Input/Output Controller Implementation

The Input/Output Controller (I/O Controller) is composed of a RevPi Core 3+ IPC and an attached RevPi DIO module. Its implementation was done through the Node-RED software since, as mentioned in Section 4.2.2.1, the RevPi Core has a version of it pre-installed with dedicated nodes, from the *node-red-contrib-revpi-nodes* package, for the interactions between the software and the IPC. Each of the RevPi DIO's input pins has an associated node that activates every time the pin detects a signal change (the pin becomes active or inactive). On the other hand, the output pins can be deactivated or activated respectively when their corresponding nodes receive a zero or a one value.

Additionally, the pre-installed Node-RED version also has the nodes from the *node-red-contrib-websocket* package, that allows communication with WebSocket servers and clients.

The RevPi DIO module was connected to the RevPi Core 3+ through a PiBridge connection. Since both the Physical Conveyors and the RevPi DIO work at a voltage of 24 V, the Physical Conveyors' wires can be directly connected to the RevPi DIO module's pins.

In order to control the real conveyors, the RevPi Core 3+ was programmed through Node-RED to interpret messages received from the WebSocket server and convert them into instructions for the real conveyors. Additionally, the IPC must be able to detect inputs from the real conveyors' sensors and send messages to the WebSocket server to inform it of the sensors' state.

4.4.1 Reacting to the WebSocket Server's Messages

In order for the Real Conveyor 1 (the physical conveyor that receives instruction messages from the endpoint */real1*) to react to the WebSocket server's messages, a *web-socket-in* node was used. This node connects to the endpoint */real1* and is connected to a Switch node, that is used to check what message was received. This Switch node then activates the necessary Change nodes that prepare the intended state of the conveyor's actuators that have to be outputted. This state is then outputted to the conveyor through the appropriate pins, this is done through a connection from the Change nodes to the *node-red-contrib-revpi-nodes* package's Output nodes.

For the control of the Real Conveyor 2, the same approach was taken. The differences are that the corresponding *web-socket-in* node connects to the */real2* endpoint, and the final Output nodes are the ones that control the pins connected to the Real Conveyor 2.

The messages that can be received from the WebSocket server are *move_left*, *move_right* and *stop*. These messages are instructions that respectively command a Physical Conveyor to make its conveyor belt move to the left, move to the right, and stop.

The control logic for the reaction to the WebSocket's server messages can be visualized through the diagram presented in Figure 4.9.

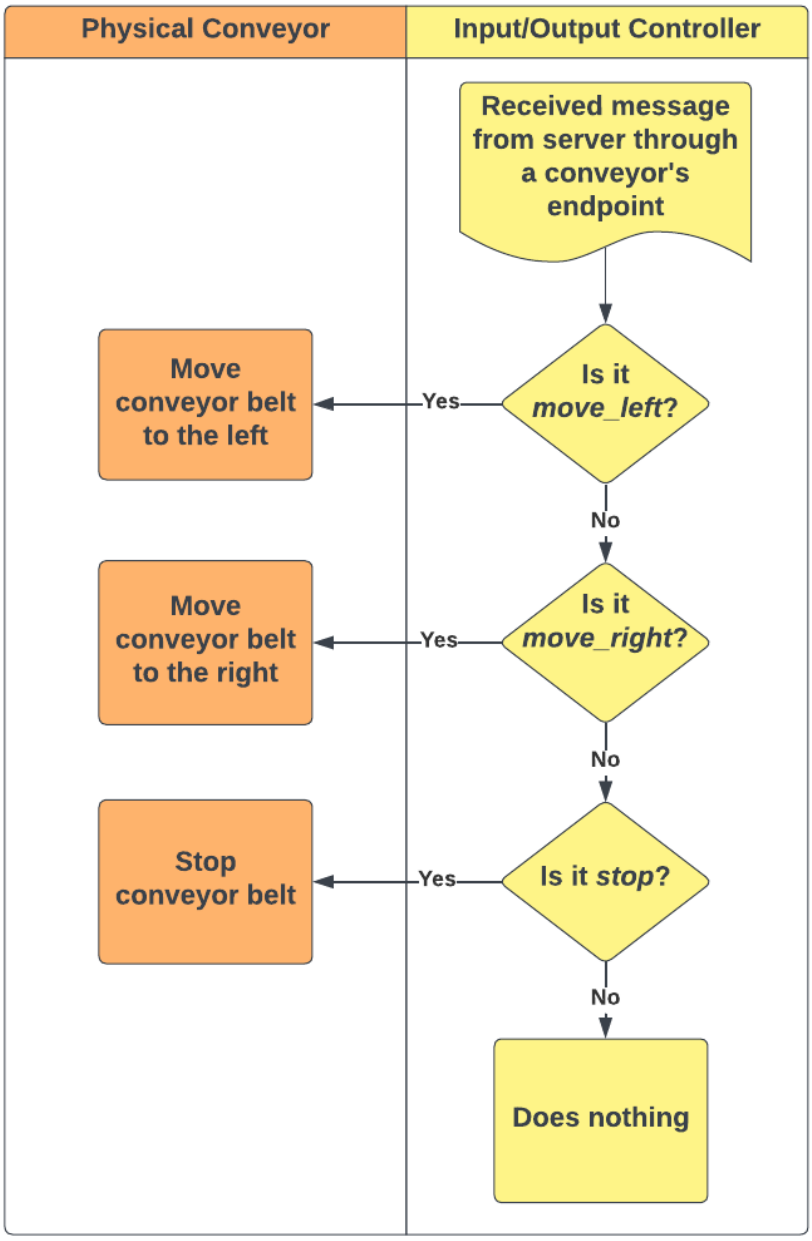


Figure 4.9: Diagram of the Input/Output Controller's operation logic when receiving messages from the WebSocket server.

4.4.2 Reacting to the Physical Conveyors' Sensors

The I/O Controller's reaction to the Real Conveyor 1's sensor state changes is implemented in the following way. The *node-red-contrib-revpi-nodes* package's Input nodes are used to receive signal changes from the RevPi DIO's module's input pins that are connected to the conveyors' sensors. This means that these Input nodes inform the I/O Controller about the conveyor's sensor changes. Each of these nodes is connected to a dedicated Switch node that checks if the corresponding conveyor's sensor was activated or deactivated. This Switch node is then connected to two Change nodes (one for each possible sensor state)

that prepare the appropriate message that has to be sent to the WebSocket server. Finally, the Change nodes connect to a *web-socket-out* node that sends the prepared message to the WebSocket server's address through the */real1* endpoint.

The I/O Controller's reaction to the Real Conveyor 2's sensor state changes was implemented through the same approach. The differences are that the used Input pins are the ones that correspond to the Real Conveyor 2's sensors and that the corresponding *web-socket-out* node connects to the endpoint */real2*.

The messages that can be sent to the WebSocket server are *left_sensor_on*, *left_sensor_off*, *right_sensor_on* and *right_sensor_off*. These messages respectively indicate that, as their name suggests, the conveyor's left sensor has been activated, its left sensor has been deactivated, its right sensor has been activated and its right sensor has been deactivated.

The control logic for the reaction to the real conveyors' sensor state changes can be visualized through the diagram presented in Figure 4.10.

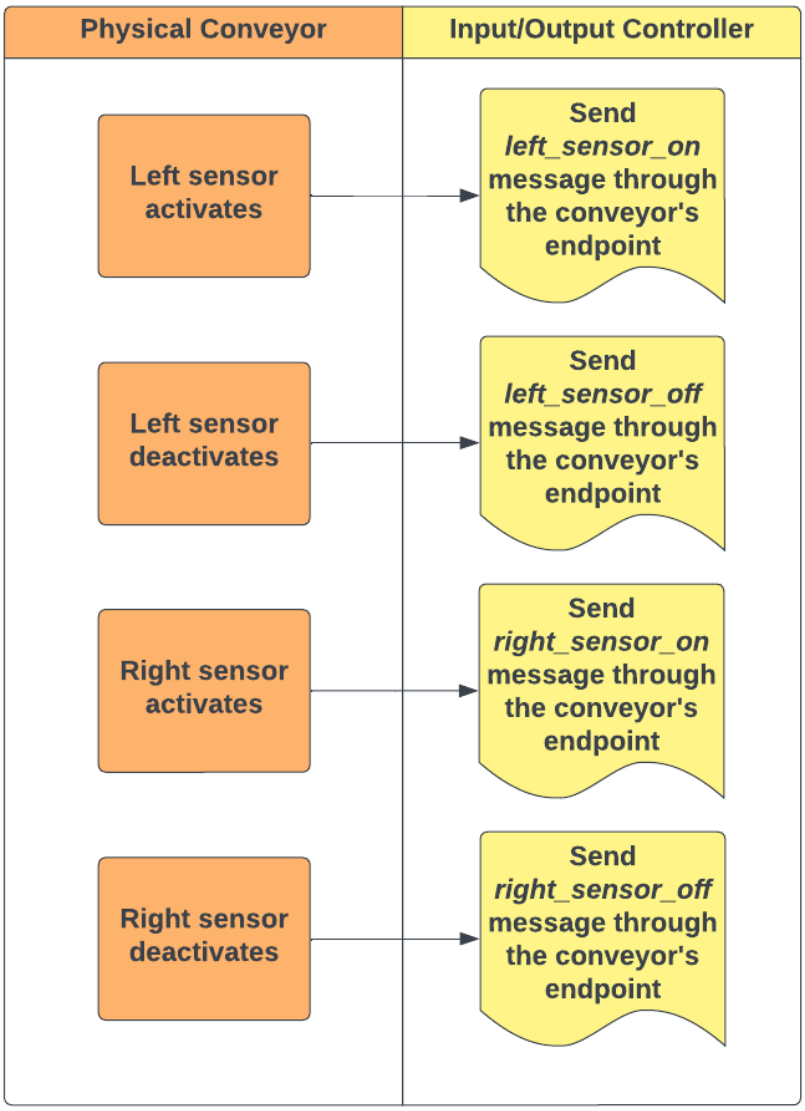


Figure 4.10: Diagram of the Input/Output Controller's operation logic when the Physical Conveyors' sensors state changes.

4.4.3 WebSocket Clients

As mentioned in Chapter 3, the I/O Controller has a WebSocket client for each of its real conveyors. The WebSocket client associated with the Real Conveyor 1 is composed of the previously mentioned *web-socket-in* and *web-socket-out* nodes that connect to the */real1* endpoint. Similarly, the WebSocket client associated with the Real Conveyor 2 is composed of the *web-socket-in* and *web-socket-out* nodes that connect to the */real2* endpoint.

4.4.4 I/O Controller Implementation Logic

Figure 4.11 presents a diagram that allows a more general visualization of the I/O Controller's code logic and its resulting operation.

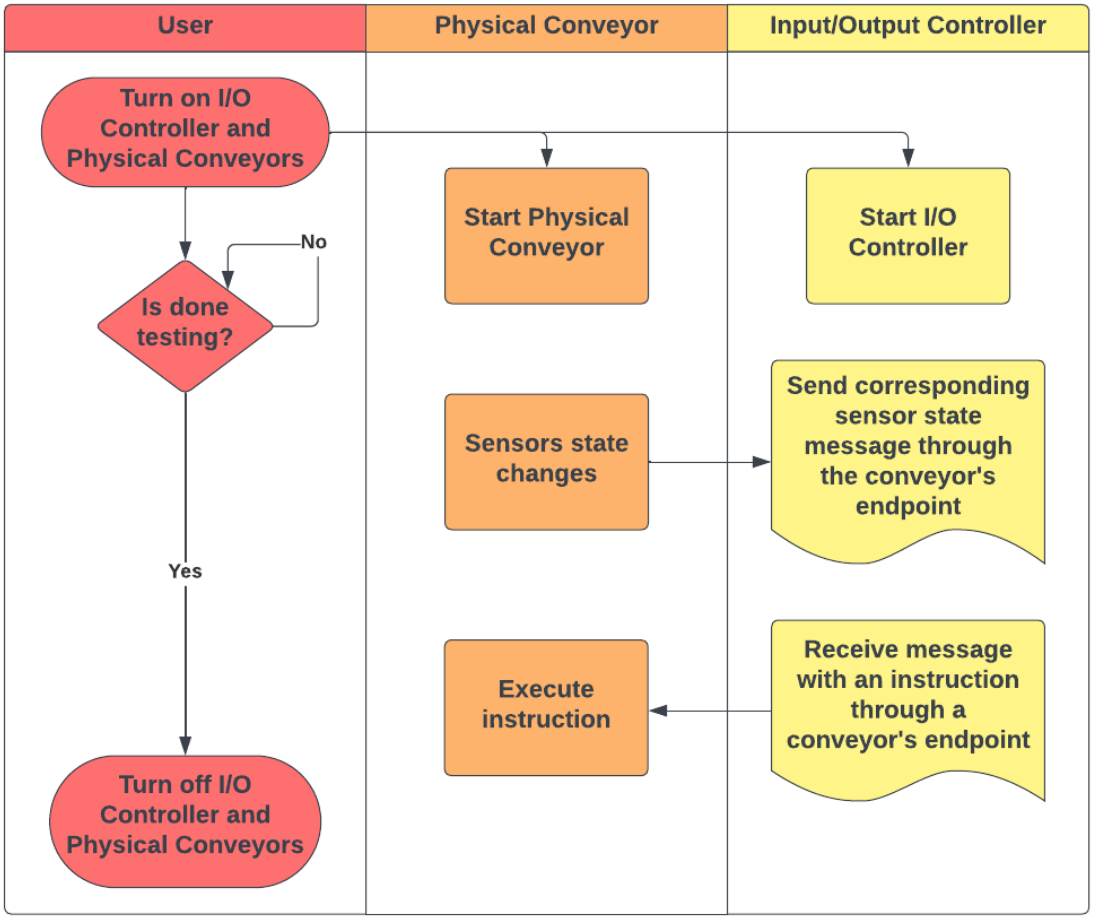


Figure 4.11: Diagram for the Input/Output Controller implementation logic.

4.5 Augmented Reality Application Implementation

The Augmented Reality (AR) application was implemented using Blender and Unity. Its implementation started with the creation of the Virtual Conveyor 3D model using Blender. Unity was then used to allow the visualization of this model in the Real Environment and to program the logic of its intended behavior since it must behave in the same way a Physical Conveyor does.

Unity was also used to implement the AR Application's WebSocket client, as well as to deploy this application to the *Android* Operating System.

4.5.1 Virtual Conveyor Model

The implementation of the AR application started with the creation of the 3D module of the Virtual Conveyor, which was done using Blender. The creation of this object was based on the Physical Conveyor model used for this implementation, which was presented in Section 4.2.4.

The model was created by using the 3D modeling tools that Blender provides to create the shape of the conveyor and its coloring and texturing support to attribute the correct colors and textures to the appropriate conveyor's superficial area zones.

The constructed 3D model can be observed in Figure 4.12.

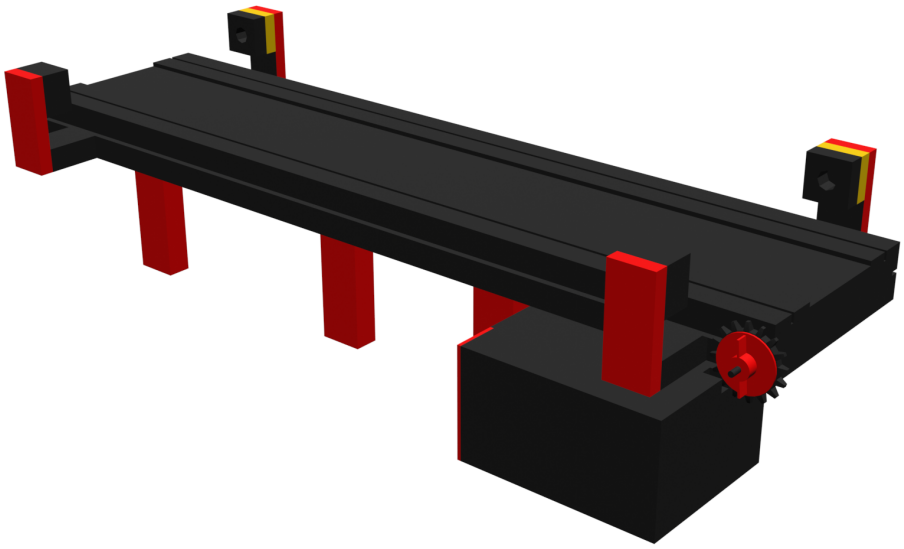


Figure 4.12: Virtual Conveyor - 3D model.

In addition to this, a model of a simple wooden box was also created, to allow the visualization of the product going through the conveyor. This box model is presented in Figure 4.13.

Figure 4.13: Box - 3D model.

Both these models were exported as FBX (Filmbox) files to later be imported into Unity.

4.5.2 Visualization Through Augmented Reality

This section explains how the visualization of the conveyor model was implemented using Augmented Reality (AR).

4.5.2.1 Unity's Augmented Reality Template Scene

Unity was used to visualize this model in AR. For this, a project was created with an AR template Scene that is already prepared to display a 3D cube in AR. This template Scene is composed of five GameObjects that are about to be introduced.

The *Directional Light* GameObject illuminates the environment, and the *Test Cube* GameObject is a simple 3D cube.

The *AR Session Origin* GameObject represents the position of the device used to visualize the AR objects, moving according to the device's real-world position and orientation.

Another GameObject is the *AR Camera*, a child of the *AR Session Origin*, which means that it is affected by every action applied to the *AR Session Origin* (linear and rotational movement for example). This *AR Camera* has the necessary Components to enable its appropriate behavior in an AR context.

Finally, the *AR Session* GameObject is responsible for managing the AR session by taking care of other important tasks like communication with the device, frame rate management, and others.

4.5.2.2 Visualizing the Virtual Conveyor on an Augmented Reality Marker

To facilitate the visualization of objects in a specific real-world location, an AR marker can used. An AR marker is a printed image that serves as a reference to a spatial location. The AR marker used was the Quick Response (QR) code presented in Figure 4.14.

Figure 4.14: Augmented Reality marker - QR code.

This AR marker has been imported into Unity and added to a *Reference Image Library*, which stores images to use later as a reference.

In addition to this, the Conveyor and the Box models that were exported from Blender as FBX files have been imported as Prefabs.

The *AR Session Origin*'s AR Tracked Image Manager and Tracked Image Prefab properties were respectively connected to the *Reference Image Library* and the Conveyor Prefab. This makes it so the application creates and displays an instance of the Conveyor Prefab when the device in which the application is running detects an image from the *Reference Image Library*.

It then becomes possible to use a printed image of the QR code as the AR marker. This marker can be placed in the location in which the Virtual Conveyor should be displayed to then be recognized by the device's camera when running the application. The application then makes the screen display the instance of the Virtual Conveyor in the desired real-world location (on top of the AR marker).

Figure 4.15 shows the visualization of the Virtual Conveyor in the real world through AR.

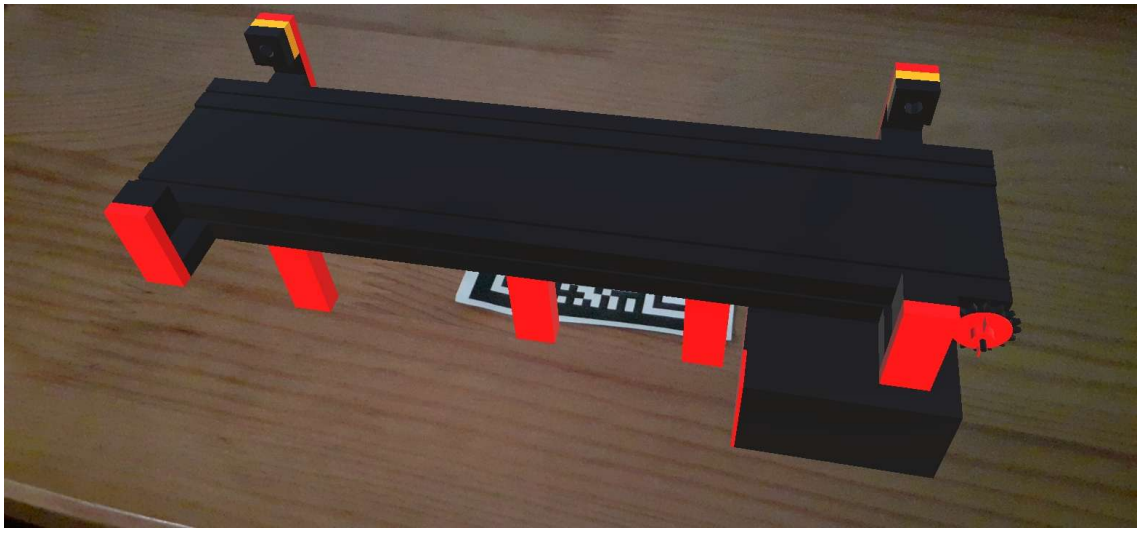


Figure 4.15: Virtual Conveyor displayed on the real world through Augmented Reality.

4.5.3 Virtual Conveyor Behavior

The Virtual Conveyor's behavior is managed by the class *ConveyorManager*, which is programmed in a script attached to a GameObject named *AR_Scripts_Manager*.

4.5.3.1 Conveyor Prefab

The Conveyor Prefab is composed of six GameObjects. These six GameObjects are the *Wheel*, *Belt*, *LeftSensor*, *RightSensor*, *SpawnPointLeft* and *SpawnPointRight*. From these GameObjects, only the last five are relevant to the implementation of the Virtual Conveyor's behavior.

The *Belt* has a Rigidbody Component attached to it for its physics to be simulated. It also has a BoxColider Component that allows the engine to detect collisions with other GameObjects. These Components will be necessary for the interaction between the conveyor's belt and the Box Prefab.

The *LeftSensor* and *RightSensor* GameObjects have the necessary BoxColider Components prepared to detect if a box is in the range of the sensors. It is then possible for these objects, through a dedicated script, to inform the *ConveyorManager* class when their sensor starts or stops detecting a box in front of it.

Finally, the *SpawnPointLeft* and the *SpawnPointRight* GameObjects are empty GameObjects that were simply located respectively above the leftmost and the rightmost extremities of the Conveyor. These objects serve as a reference to the location of the Scene in which the Box Prefab should appear when simulating its movement through the Conveyor.

4.5.3.2 Box Prefab

The Box Prefab uses the Rigidbody and BoxColider Components to aid the Virtual Conveyor's behavior. The Rigidbody Component simulates the physics of the object, including the gravity to which the Box should be subjected. The BoxColider Component allows the Box to detect its collision with the *Belt* and therefore not go through it but instead stand on top of it.

4.5.3.3 Conveyor Manager Code

In order to simulate the behavior of a Physical Conveyor, the *ConveyorManager* class starts by telling the WebSocket client (associated with the *WebSocketCleint* class) to send a message with the information that the conveyor's sensors are not detecting anything in front of them. In addition to this, the belt speed is set to zero since it shouldn't be moving as soon as the AR marker is detected.

Once per frame, the *ConveyorManager* class tries to find an instance of the Conveyor Prefab in the Scene to manage it afterward (the instance exists if the Virtual Conveyor has appeared due to the device's camera detection of the AR marker).

Once every 0.02 seconds, the *Belt's* movement is updated according to its speed. In order to affect the *Belt* correctly, its Rigidbody Component is moved according to the belt's speed, bringing any object that is on top of it with it, and then its location is directly set to the initial position, which means that it returns to the initial position without taking any object with it. This way, every 0.02 seconds, if a Box Prefab instance is on top of the *Belt*, it will be moved according to the belt's speed, but the *Belt* itself will remain stopped.

The *ConveyorManager* class is capable of interacting with the Virtual Conveyor to make it behave as a Physical Conveyor. To do this, it changes the belt's speed when it has to make the conveyor belt move in a certain direction or stop.

To manage the Virtual Conveyor's sensors, every time the conveyor's sensor state changes, their GameObjects' classes (*LeftSensor* and *RightSensor*) make the *ConveyorManager* class instruct the *WebSocketClient* class to send an appropriate message to the server (a message that informs it about the state of the Virtual Conveyor's sensors).

In addition to this, since the Virtual Conveyor cannot interact with the real box, the *ConveyorManager* class also has the capability of instantiating the Box Prefab on the leftmost or rightmost extremities of the Virtual Conveyor when there are no other boxes in the Scene, as well as the capability of deleting this instance of the Box Prefab.

4.5.4 WebSocket Client

In order to program the WebSocket client for the AR application, the *websocket-sharp* library was used. This library allows the creation and management of WebSocket clients and servers through the C# programming language.

The WebSocket client is the *WebSocketClient* class, programmed in the *WebSocketClient* script, which is a Component of the *AR_Scripts_Manager* GameObject. This class starts a WebSocket client that connects to the server's corresponding address and port, connecting specifically to endpoint */virtual*. This client can send messages to and receive messages from the server.

The client can send the *left_sensor_on* and the *right_sensor_on* messages, which are sent to inform the server, respectively, about the activation of the Virtual Conveyor's left and right sensor. Additionally, the messages *left_sensor_off* and *right_sensor_off* can also be sent to inform the server, respectively, about the deactivation of the Virtual Conveyor's left and right sensor.

Figure 4.16 represents, through a diagram, how the application reacts to changes in the Virtual Conveyor's sensor state.

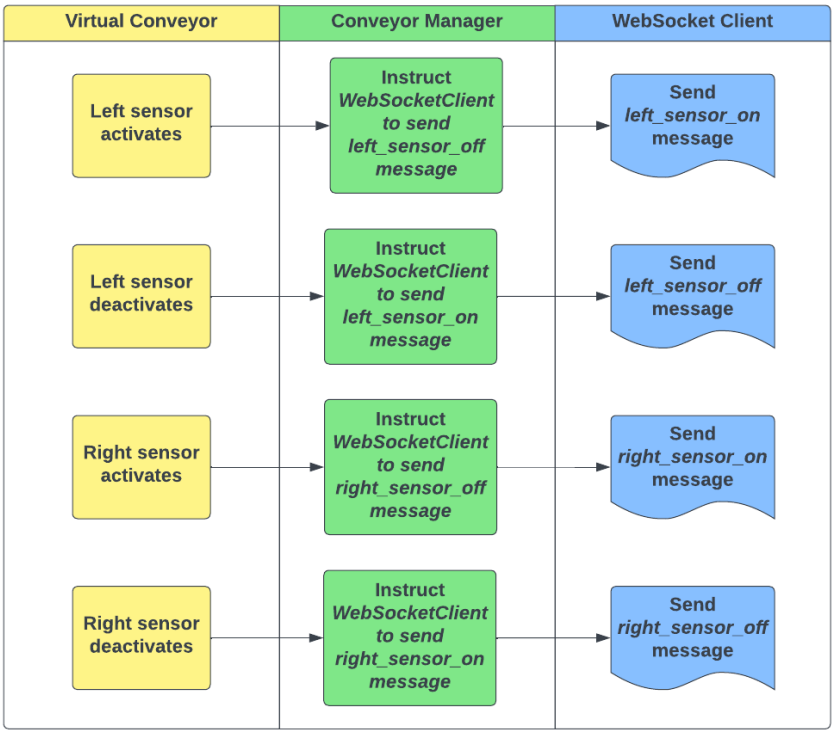


Figure 4.16: Diagram of the application's operation logic when the Virtual Conveyor's sensor state changes.

The client was also programmed to react in the intended way when receiving specific messages from the server. When the messages *move_left*, *move_right* or *stop* are received, the client instructs the *ConveyorManager* class to respectively, make the Virtual Conveyor start moving to the left, start moving to the right, or stop. In a similar way, when the messages *create_box_left*, *create_box_right* or *delete_box* are received, the *ConveyorManager* class is respectively instructed to create an instance of the Box Prefab at the leftmost extremity of the Virtual Conveyor, create an instance of the Box Prefab at the rightmost extremity of the Virtual Conveyor or delete the Box Prefab instance from the Scene.

Figure 4.17 represents, through a diagram, how the application operates when it receives messages from the WebSocket server.

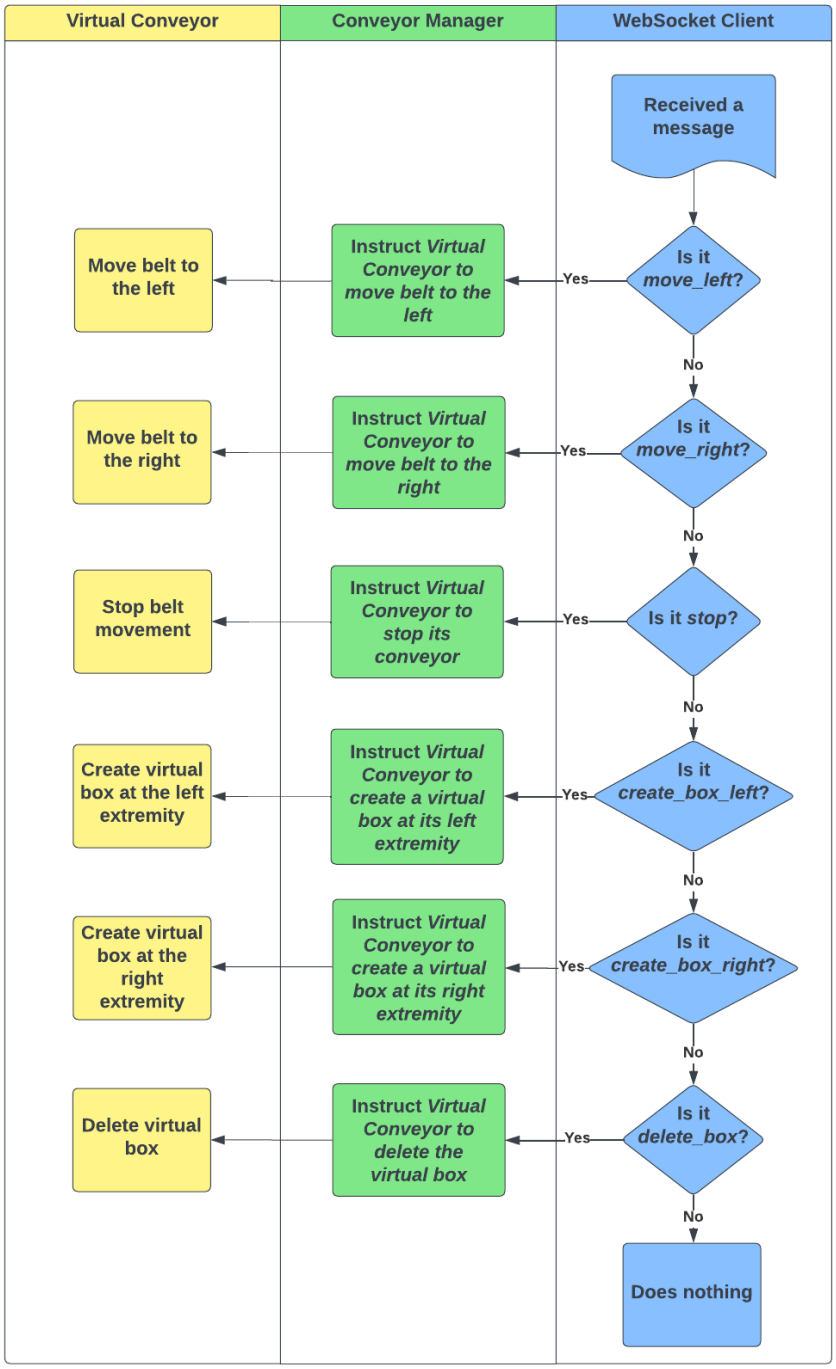


Figure 4.17: Diagram of the application's operation logic when receiving messages from the WebSocket server.

Finally, the client connects to the server as soon as the application starts, and this connection is closed when the application is terminated.

4.5.5 Augmented Reality Application Implementation Logic

Figure 4.18 presents a diagram that allows a more general visualization of the AR Application's code logic and its resulting operation.

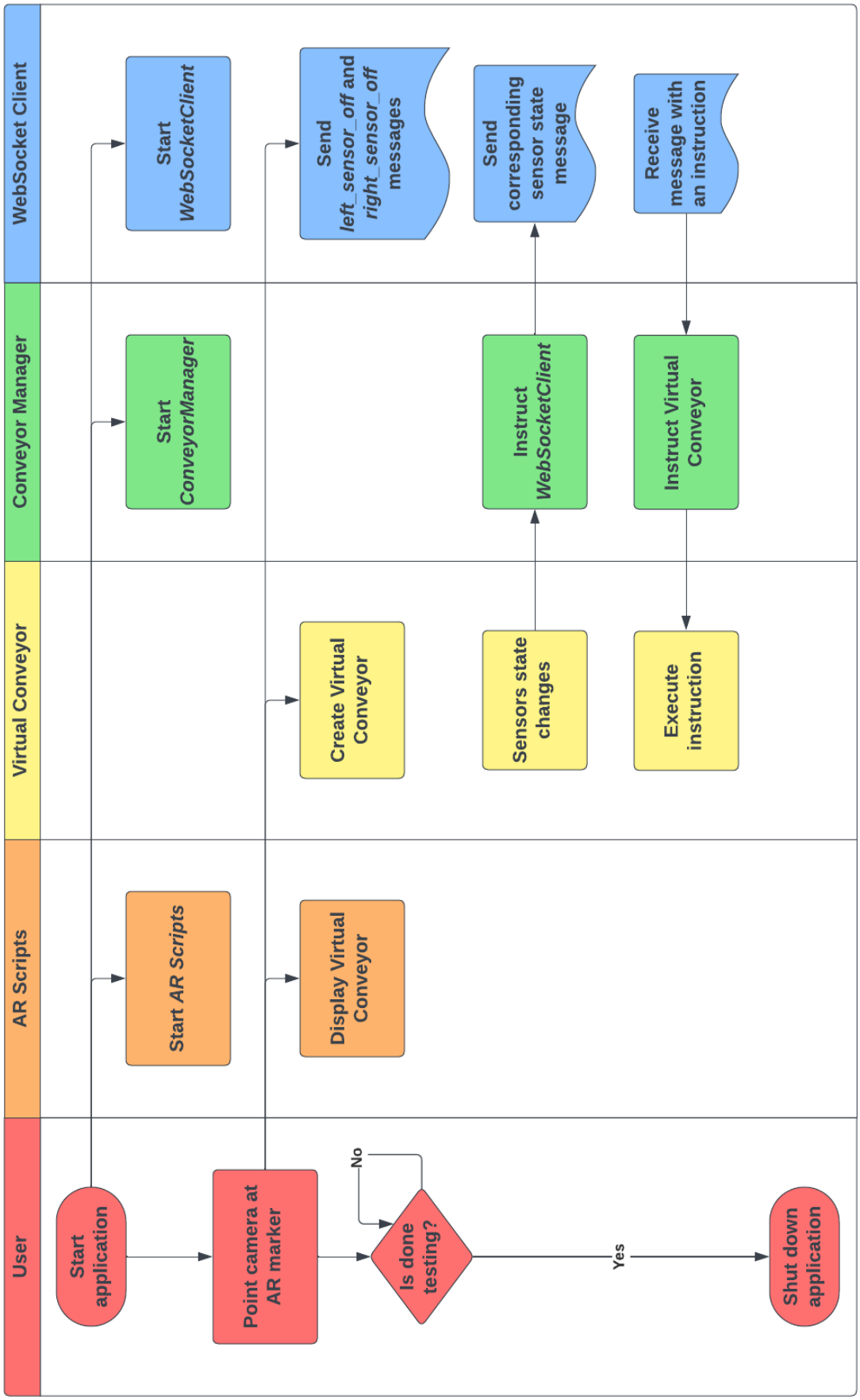


Figure 4.18: Diagram of the Augmented Reality Application's implementation logic.

4.6 Summary

This chapter explained how the main components of the architecture presented in Chapter 3 were implemented.

The implementation of the System Controller was done using the Java Application DEvelopment Framework (JADE) for the creation of a Multi-Agent System (MAS) that executes the logic of this controller, while the *Java WebSocket* library was used for the programming of the Local Area Network's (LAN) server-side code.

The Input/Output Controller (I/O Controller) was implemented by using the RevPi Core 3+ Industrial Personal Computer (IPC) to program the controller's logic using the pre-installed Node-RED software in conjunction with a RevPi DIO module, responsible for connecting the IPC to Physical Conveyors.

Finally, the Augmented Reality (AR) Application was implemented by using Blender for the 3D modeling of a Virtual Conveyor, as well as using Unity for the display of this Virtual Conveyor using AR, for the programming of its behavior logic, and for the programming of a WebSocket client using the *websocket-sharp* library.

The system must now be tested and validated in order to make sure that it is capable of fulfilling the project's objectives successfully.

5

Tests and Validation

This chapter tests and validates the implemented project. It starts by explaining what tests the project has to be subjected to and presenting how each test was prepared and executed. It continues by presenting the results of each test by describing what was visualized during their execution. The chapter then uses the tests' results to validate the implemented project, while concluding by verifying that this project is a solution to the initially introduced problem.

5.1 Tests

It is necessary to verify that the project is working in the intended way. To do this, the objectives introduced in Section 1.3 should be correctly fulfilled during the tests done throughout this chapter. These objectives are, in the context of Modular Production Systems (MPS):

- Testing the controller code for a new module without having to purchase it.
- Visualizing the integration of a new module into an existing real system by watching it operate with that system in its environment.

For the fulfillment of these objectives, three things must be verified during the afterward presented tests.

The virtual module should be correctly commissioned, which can be confirmed by verifying if it is behaving as expected and interacting correctly with the remaining system modules.

Additionally, this module should also be able to be visualized through a mobile device during its operation, which can be confirmed by seeing if the module is correctly displayed on the device's monitor throughout the tests.

Finally, since this project wasn't limited to the creation of the virtual module, but instead involved the development of the whole MPS, it is also necessary to verify that it is behaving as a MPS. This is necessary since, in case the system doesn't act as a MPS, then

it isn't possible to prove that the solution presented throughout this document allows the commissioning of MPSs.

Since the system is a Modular Production System (MPS), its modules should be able to be easily added or removed from the system and be arranged in any order. In order to verify this, three system arrangements will be tested. First of all, the Virtual Conveyor will be tested on its own. The second arrangement of conveyors to be tested will be composed of, from left to right, a Physical Conveyor, followed by a Virtual Conveyor, and then the remaining Physical Conveyor. Lastly, the final arrangement to be tested will be composed of, from left to right, the two Physical Conveyors and the Virtual Conveyor.

For the system to be considered a MPS, its original modules and the added virtual module should be able to execute any of their associated skills in any necessary order instead of working together as a system that only provides a single function. To verify the system modules' skill individualization, three tasks were attributed to the system for each of the previously mentioned conveyor arrangements.

The three tasks tested for each arrangement were the following:

- Make a product go from left to right and then from right to left through the conveyor sequence.
- Make a product go from right to left and then from left to right through the conveyor sequence.
- Make a product go from left to right twice through the conveyor sequence.

The first task allows the testing of all of the system's different skills, while the second task does the same while additionally confirming that the product can not only be placed at the left of the conveyor arrangement but also at the right. Finally, the last task allows the verification that the project can operate correctly more than once, as intended.

5.1.1 Tests Preparation

This section explains how the project was modified to allow the execution of the previously mentioned tests.

Left to Right and Back:

As mentioned in Section 4.3.1, the System Controller's application was programmed to generate a skill list that the system should execute for the next product to be ready.

From the conveyor order submitted by the user, the application generates the skill list that allows a product to be transported through the conveyor sequence from left to right and back, from right to left.

The application is therefore already ready to test the task that was first presented above, which is making a product go from left to right and then from right to left through the conveyor sequence.

Right to Left and Back:

In order to test the second task, the System Controller's application code was modified to create a skill list that, when executed by the system, makes the next product go from right to left and then back, from left to right.

It is important to reference that the code was modified to, for the correct execution of this task, have the user input the arrangement of the conveyors from right to left instead of left to right.

This was done because this task starts by moving the product to the left, so it is more intuitive to input the order of the conveyors from the right to the left.

Left to Right Twice:

Equivalently, to test the third task, the System Controller's application code was modified to create a skill list that the system must execute to transport a product from left to right twice.

For this task, the user should input the conveyor order as normal, from left to right.

5.1.2 Testing the Virtual Conveyor

In order to test the Virtual Conveyor by itself, the user should, after preparing the System Controller application to test the appropriate task as mentioned before, begin by respectively starting the System Controller's application and the Augmented Reality (AR) application. The camera of the AR application's device should then be pointed at the AR marker for the Virtual Conveyor to become visible. The user should then only select the Virtual Conveyor in the System Controller's Graphical User Interface (GUI) to submit the correct conveyor arrangement, and the test will start.

It is possible to see the Virtual Conveyor in operation in Figure 5.1.

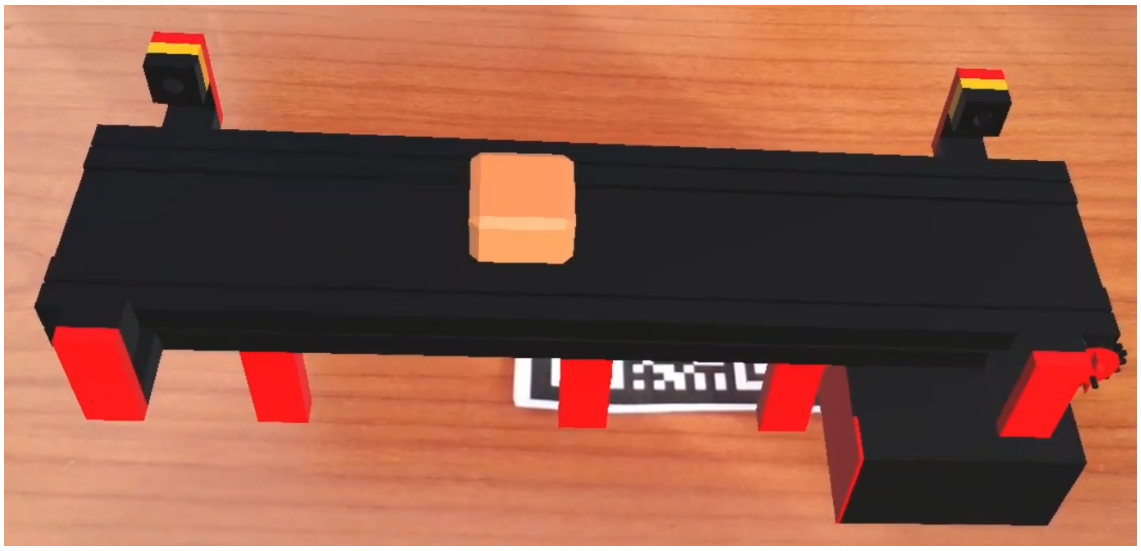


Figure 5.1: Virtual Conveyor in operation.

After the test has been concluded, the user shuts down the applications in whichever order they prefer.

5.1.2.1 Tests execution

Since for the Virtual Conveyor tests, all of the system's actions occur in the virtual world, the user only has to observe the conveyor's behavior throughout the execution of the tested task. This applies to all of the three previously presented tasks.

5.1.3 Testing the Virtual Conveyor Between the Physical Conveyors

In order to test the system with the Virtual Conveyor between the two Physical Conveyors, the user should also prepare the System Controller application to test the appropriate task as mentioned before. They can then start the System Controller's application and continue by starting the AR application and turning on the Physical Conveyors and the Input/Output Controller (I/O Controller). The camera of the AR application's device should then also be pointed at the AR marker for the Virtual Conveyor to appear on the device's monitor.

The user should then use the System Controller's application GUI to select the three conveyors in their arrangement order and submit this order. The test execution will then start.

It should be noted that the conveyor order selected depends on the task to be executed. For the "right to left and back" task the order should be selected from right to left, which means the Physical Conveyor located at the right extremity should be selected first, which should be followed by the Virtual Conveyor, and then by the Physical Conveyor located at the left extremity. On the other hand, for the remaining tasks, the order should be selected from left to right, which means the order should be inverted.

The tested conveyor arrangement can be seen in operation in Figure 5.2.

Figure 5.2: Conveyor arrangement with the Virtual Conveyor positioned between the Physical Conveyors in operation.

After the test has been concluded, the user shuts down the applications and turns off the I/O Controller and the Physical Conveyors in whichever order they prefer.

5.1.3.1 Left to Right and Back

At the start of the execution of this task, the user has to start by placing the product on the leftmost conveyor's left extremity.

The product should be picked up every time it transitions from one of the Physical Conveyors to the Virtual Conveyor because a real product cannot be transported by the Virtual Conveyor.

Additionally, the product has to be placed at the next Physical Conveyor's correct extremity when it transitions from the Virtual Conveyor to a Physical Conveyor.

Finally, the user should remove the product from the system when the task finishes its execution.

5.1.3.2 Right to Left and Back

After the task starts its execution, the user must place the product on the rightmost conveyor's right extremity.

As in the execution of the last mentioned task, the product should also be picked up every time it transitions from one of the Physical Conveyors to the Virtual Conveyor and be placed at the next Physical Conveyor's correct extremity when it transitions from the Virtual Conveyor to a Physical Conveyor.

Finally, the user should also remove the product from the system when the task execution concludes.

5.1.3.3 Left to Right Twice

At the start of this task's execution, the user must place the product on the leftmost conveyor's left extremity.

As in the execution of the other two mentioned tasks, the product also has to be picked up every time it moves from one of the Physical Conveyors to the Virtual Conveyor and be placed at the next Physical Conveyor's correct extremity when it moves from the Virtual Conveyor to a Physical Conveyor.

Additionally, when the product reaches the right extremity of the rightmost conveyor it must be removed from the system and placed one more time at the left extremity of the leftmost conveyor for it to be transported to the right again.

The user should also remove the product from the system at the end of the task's execution.

5.1.4 Testing Virtual Conveyor After the Physical Conveyors

To test the system with the Virtual Conveyor positioned after the two Physical Conveyors, the user also has to prepare the System Controller application to test the appropriate task, start the System Controller's application, continue by starting the AR application, and turn on the Physical Conveyors and the I/O Controller, just like for the last mentioned

tests. The camera of the AR application's device should then also be pointed at the AR marker for the Virtual Conveyor to be visualized through the device's monitor.

As for the last mentioned test, the user should also select the three conveyors in their arrangement order and submit this order by using the System Controller's application GUI. The test will then start its execution.

Just as for the last mentioned test, the conveyor order selected depends on the task to be executed. For the "right to left and back" task the order should be selected from right to left, which means the Virtual Conveyor located at the rightmost position should be selected first, which should be followed by the middle Physical Conveyor, and then by the leftmost Physical Conveyor. On the other hand, for the remaining tasks, the order should be selected from left to right, which means it should be inverted.

The tested conveyor arrangement can be seen in operation in Figure 5.3.

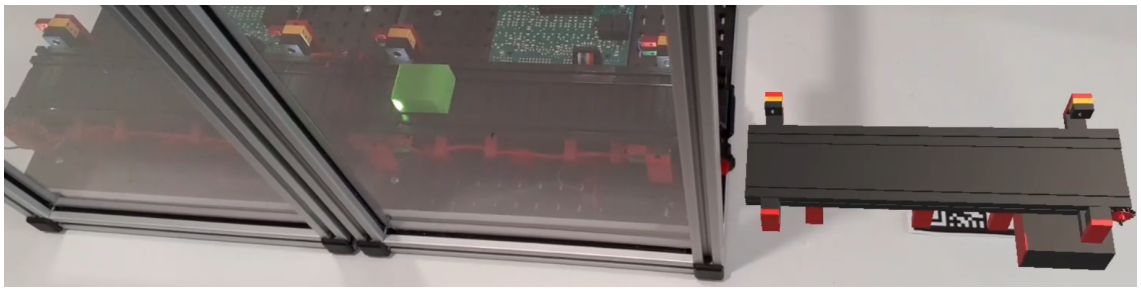


Figure 5.3: Conveyor arrangement with the Virtual Conveyor positioned after the Physical Conveyors in operation.

After the test has been executed, the user must shut down the applications and turn off the I/O Controller and the Physical Conveyors in any order.

5.1.4.1 Left to Right and Back

When this task starts its execution, the user must start by placing the product on the leftmost conveyor's left extremity.

The product should then be picked up every time it moves from one of the Physical Conveyors to the Virtual Conveyor because a real product cannot be moved by the Virtual Conveyor.

Additionally, the product has to be placed at the next Physical Conveyor's correct extremity when it moves from the Virtual Conveyor to a Physical Conveyor.

Finally, the user should pick up the product from the system when the task finishes executing.

5.1.4.2 Right to Left and Back

After the task starts its execution, the product doesn't have to be placed immediately on the system because it starts its movement on the Virtual Conveyor.

As in the execution of the last mentioned task, the product should also be placed at the next Physical Conveyor's correct extremity when it transitions from the Virtual Conveyor to a Physical Conveyor and picked up every time it transitions from one of the Physical Conveyors to the Virtual Conveyor.

Finally, the user doesn't have to remove the product from the system when the task execution concludes since the last conveyor the product goes through is the Virtual Conveyor.

5.1.4.3 Left to Right Twice

At the start of this task's execution, the user must place the product on the leftmost conveyor's left extremity.

As in the execution of the other two mentioned tasks, the product also has to be picked up every time it transitions from one of the Physical Conveyors to the Virtual Conveyor and be placed at the next Physical Conveyor's correct extremity when it transitions from the Virtual Conveyor to a Physical Conveyor.

Additionally, when the virtual product reaches the right extremity of the Virtual Conveyor, it must be placed one more time at the left extremity of the leftmost conveyor to be transported to the right again.

The user doesn't have to remove the product from the system at the end of the task's execution because the last conveyor that it goes through is the Virtual Conveyor.

5.2 Results

This section presents the results of the tests done on the project. Since the verification of the system's correct operation can only be done by evaluating its observable behavior, the results of the tests presented throughout the last section are a description of the system's behavior throughout them.

5.2.1 Virtual Conveyor

This section describes the results of the three tests done on the Virtual Conveyor.

5.2.1.1 Left to Right and Back

At the start of this test, a virtual product appeared at the left extremity of the Virtual Conveyor, in front of its left sensor. The product was then transported to the right until it reached the right sensor of the conveyor.

The conveyor then stopped moving and the product was deleted from the virtual environment and reappeared at the same position, in front of the right sensor of the Virtual Conveyor.

Finally, the product was transported back to the left until it reached the conveyor's left sensor, where the conveyor was stopped and the product was deleted once again.

5.2.1.2 Right to Left and Back

The results of this test were symmetric to the ones presented in the last section. The product appeared at the right extremity of the Virtual Conveyor, starting by moving to the left until it reached the left sensor and then returning to the right sensor, where the conveyor was stopped and the product was deleted once again, finishing the task's execution.

5.2.1.3 Left to Right Twice

In this test, the virtual product appeared at the left extremity of the conveyor and it started by being moved to the right until it reached the right sensor.

The conveyor was then stopped, and the product was deleted and recreated at the conveyor's left extremity. It was then once again transported to the right until it reached the right sensor, where the conveyor stopped and the product was deleted, ending the task's execution.

5.2.2 Virtual Conveyor Between the Physical Conveyors

Next, the results of the tests done on the system when it had the Virtual Conveyor between the two Physical Conveyors are described.

5.2.2.1 Left to Right and Back

For this test, the product started at the left extremity of the conveyor sequence and was moved to the right until it reached the end of the first Physical Conveyor (where it was removed by the user). A virtual product then appeared at the Virtual Conveyor's left extremity, resulting in the interruption of the Physical Conveyor's movement.

From this position, the product was also moved to the right, starting the next Physical Conveyor's belt movement when detected by the Virtual Conveyor's right sensor. This virtual product was then deleted and the real product reached the left sensor of the second Physical Conveyor (being placed by the user), resulting in the interruption of the Virtual Conveyor's movement.

The product was then transported to the right until it reached the Physical Conveyor's right sensor, being then transported back through the entire conveyor system in the same way but from the right to the left. The execution of the task ended when the product reached the left sensor of the leftmost conveyor.

5.2.2.2 Right to Left and Back

The results of this test were symmetric to the ones presented in the last section. The product started at the right extremity of the conveyor sequence, from where it was moved to the left, going through, respectively, the rightmost Physical Conveyor, the Virtual Conveyor, and the leftmost Physical Conveyor until reaching this last conveyor's left sensor.

The product then returned to the rightmost conveyor's right sensor by being transported back to the right through the entire conveyor sequence and ending the task's execution.

5.2.2.3 Left to Right Twice

In this test, the product started at the left extremity of the conveyor sequence and was moved to the rightmost extremity of it in the same way as in the "left to right and back" test, explained in Section 5.2.2.1.

When the product reached the right sensor of the rightmost Physical Conveyor of the sequence, this conveyor was stopped. The product was then placed back at the left extremity of the conveyor sequence (by the user) and was once again moved, in the same way, to the rightmost extremity of the conveyor sequence.

When the product reached the right sensor of the rightmost Physical Conveyor the second time, this conveyor was stopped once again and the task finished its execution.

5.2.3 Virtual Conveyor After the Physical Conveyors

Finally, a description of the results of the tests done on the system when it had the Virtual Conveyor placed after the two Physical Conveyors is given.

5.2.3.1 Left to Right and Back

For this test, the product started at the left extremity of the leftmost Physical Conveyor. The product was moved to the right through the two Physical Conveyors, starting the second conveyor's belt movement when detected by the first conveyor's right sensor and stopping the first conveyor's belt movement when detected by the second conveyor's left sensor.

When the product reached the right sensor of the second Physical Conveyor, the virtual product appeared on the left extremity of the Virtual Conveyor, being detected by its left sensor and stopping the second Physical Conveyor's belt movement. This virtual product was then transported to the right until it reached the Virtual Conveyor's right sensor, where it was deleted.

This virtual product then reappeared at the right extremity of the Virtual Conveyor and was transported to the left until it reached the Virtual Conveyor's left sensor. This product was then deleted and the belt of the Physical Conveyor located in the middle of the conveyor sequence started its movement to the left.

The product was then placed at the middle conveyor's right extremity (by the user), from where it was transported back to the left extremity of the conveyor sequence through the two Physical Conveyors in the same way it was previously transported to the right. The task's execution ended when the product reached the left sensor of the leftmost conveyor.

5.2.3.2 Right to Left and Back

In this test, the product started by appearing virtually on the Virtual Conveyor's right extremity, from where it was moved to the left extremity of the conveyor sequence by being transported through all the conveyors in the same way as in the last explained test.

When the product reached the left sensor of the leftmost conveyor, it was transported back to the rightmost extremity of the conveyor sequence, also in the same way as in the last explained test. The task execution ended when the product was deleted after reaching the Virtual Conveyor's right sensor.

5.2.3.3 Left to Right Twice

In the test of the "left to right twice" task, the product started at the left extremity of the conveyor sequence. Just like at the start of the "left to right and back" task, the product was transported to the right through all the system's conveyors until it reached the right sensor of the Virtual Conveyor.

The virtual product was then deleted, and the product was once again placed at the left extremity of the conveyor sequence (by the user). In the same way, the product was then transported to the right one more time until it reached the right sensor of the Virtual Conveyor once more. Finally, the product was deleted and the task's execution was complete.

5.2.4 Virtual Conveyor Visualization Results

Throughout every test, it was possible to clearly visualize the Virtual Conveyor through the mobile device's monitor when the Augmented Reality marker was clearly visible by the device's camera and this device was relatively stable. The visualization correctly displayed the Virtual Conveyor's behavior when operating by itself and with the remaining system modules.

5.3 Validation

This section analyzes the tests' results and draws conclusions about the project's validity, taking into account its various purposes.

By analyzing the previous results, it is possible to affirm that the Virtual Conveyor has behaved as expected and has interacted correctly with the remaining system modules (the Physical Conveyors). This indicates that the software code responsible for the control of this module can be correctly tested with this project, without the necessity to acquire a physical module, or in other words, this project allows the module to be commissioned through Virtual Commissioning.

It is also possible to confirm that the Virtual Conveyor was able to be clearly visualized in operation by itself and with the remaining system through the mobile device's monitor

as long as the device was relatively stable and the Augmented Reality marker was clearly visible by its camera. This indicates that the project allows good visualization of the module in the system's environment, operating by itself or with the remaining system modules.

Since the system behaved correctly for the different conveyor arrangements, it is possible to observe that modules can easily be added to the system, removed from the system, and have their order rearranged.

Additionally, because the three tasks were successful for the three tested arrangements, it was also possible to confirm that the system's modules can execute their associated skills in any necessary order, which means the system's skills are individualized.

It is therefore possible to conclude that the system developed for this project is a Modular Production System (MPS).

These conclusions allow the validation of the developed project since it is capable of executing all of its intended functions. The project's validated functions were the commissioning of the Virtual Conveyor without the necessity of the acquisition of its real module and the correct visualization of this module in operation, interacting with the remaining system. It was also possible to validate the system developed in the project's context since it was demonstrated that it is a MPS.

5.4 Summary

This chapter allowed the validation of the developed project. As concluded in the last section, the project allows, in the context of MPSs, the commissioning of the Virtual Conveyor without the acquisition of its corresponding real module (through Virtual Commissioning) and the visualization of this module in interaction with the remaining conveyors on their environment.

Because of this validation, it is possible to affirm that the project is a solution to the problem proposed by this document since it fulfills the two objectives introduced in Section 1.3 in the context of MPSs. Once again, these objectives are:

- Testing the controller code for a new module without having to purchase it.
- Visualizing the integration of a new module into an existing real system by watching it operate with that system in its environment.

The Virtual Commissioning of the Virtual Conveyor satisfies the first objective since it tests the control code of a new module of a MPS and this module's acquisition isn't necessary.

The visualization of this conveyor's behavior and interaction with the other conveyors in their environment fulfills the second objective since it allows the visualization of the integration of a new module into a MPS.

6

Conclusions and Future Work

This chapter concludes the document by summarizing the work done throughout it and presenting suggestions for future work that can be done in its context.

6.1 Conclusions

Modular Production Systems (MPS) are becoming more and more important in the modern industry. Although Virtual Commissioning (VC) is used to reduce the time, risks, and costs associated with the commissioning of industrial systems, there is no solution that uses VC to commission modules that a manufacturer wants to add to a MPS. Such a solution would be relevant since MPSs are flexible, so it is common for a manufacturer to want to add modules to it, remove them from it, or rearrange them.

Additionally, the VC of systems with a good visualization allows a manufacturer to have a good perception of how the final system will look and behave, allowing them to be more certain about the acquisition of the system and therefore many times reducing costs and risks associated with the purchase of the system.

This document proposes a solution for this problem that uses VC to commission a module that is to be added to a MPS, while allowing the visualization of this module in the Real Environment (RE) through Augmented Reality (AR) during its commissioning, in order for the manufacturer to more easily visualize what is happening to the simulated module and to, as mentioned before, reduce the risks and costs associated with the future module's purchase.

This solution is composed of a System Controller that runs on a Personal Computer (PC), an Input/Output Controller (I/O Controller) that runs on an Industrial Personal Computer (IPC), and an AR Application that runs on a mobile device (smartphone for example).

The System Controller controls the entire manufacturing system through a Multi-Agent approach, to allow the system to be modular. The I/O Controller receives instructions from the System Controller and commands the Physical Modules, while also informing the System Controller of the modules' sensor states.

The AR Application has a similar function, receiving instructions from the System Controller and commanding a simulation of the module that the manufacturer wants to add to the system, while also informing the System Controller of this module's sensor state.

Additionally, the AR Application is responsible for allowing the manufacturer to visualize a 3D module of the module (a Virtual Module) that they want to add to the system, even during its operation and interaction with the remaining system modules. The visualization is done by pointing the mobile device's camera at an AR marker, that should be placed at the new module's intended location, and looking at the device's monitor, in which the Virtual Module will be displayed on top of the AR marker.

The System Controller, the I/O Controller, and the AR Application communicate with each other through a Local Area Network (LAN).

The solution was implemented in a manufacturing system composed of two Real Conveyors, to which a Virtual Conveyor is to be added. For this implementation, the System Controller application was programmed in JADE (Java Agent DEvelopment Framework), the I/O Controller logic was controlled by the Node-RED software, and the AR Application was programmed in Unity. The protocol used for the LAN communication between the project's devices was the WebSocket Protocol.

The proposed solution was then tested and validated, and it was possible to conclude that it solves the presented problem since it allows a reduction of the risks, delays, and costs associated with the commissioning of a module that is to be added to a MPS, while also reducing its purchase risks and costs by allowing the visualization of this module in operation in the same environment as the remaining system.

6.2 Future Work

Even though the proposed solution solves the presented problem, there is still future work that can be done to modify and improve it.

The proposed solution tests the control logic of the new module. Still, it can be improved to also test the code responsible for communicating between the Input/Output Controller (I/O Controller) and this module. This could be done by programming the I/O Controller to interact with a third module, but instead of sending a signal to its actuators, sending messages to the AR Application. This way, the I/O Controller's code would also be tested, and the code responsible for receiving instructions from the System Controller, commanding the new conveyor, and informing the System Controller about the conveyor's sensor state would also be tested.

The method of visualizing the module through AR also has room for improvement since sometimes, when the device gets too far from the AR marker or is shaken, the Virtual Module moves unexpectedly. Sometimes, this unexpected movement can occur when the product is being transported, and if in this case, the module is tilted, the product might fall off it, resulting in the need to restart the commissioning test.

The project can also be modified to be expandable, allowing the addition of more 3D models of other modules to be commissioned.

Additionally, the project can improve in terms of error management, allowing the system to inform the user when an unexpected situation occurs.

Since the manufacturer is the one moving the product between physical modules during the commissioning tests, there are some cases in which the project would have to be re-adapted due to the products being too heavy or the physical modules being unreachable by the manufacturer. For these cases, a physical transportation module, such as an Automated Guided Vehicle (AGV), would have to be added to the system and programmed as another Agent of the System Controller's Multi-Agent System (MAS) code. This Agent would take care of the transportation of the product between stations during the operation of the virtual module or modules.

Finally, the presented solution has to be tested in different modules, in order to verify its correct behavior for different MPSs. This includes tests done in actual industrial environments, using large-scale modules.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] H. Lasi et al. "Industry 4.0". In: *Business & Information Systems Engineering* 6 (2014), pp. 239–242. URL: https://api.semanticscholar.org/CorpusID:207433314 (cit. on p. 1).

[3] S. Weyer et al. "Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems". In: *IFAC-PapersOnLine* 48 (2015), pp. 579–584. URL: https://api.semanticscholar.org/CorpusID:112118276 (cit. on p. 1).

[4] G. Reinhart and G. Wünsch. "Economic application of virtual commissioning to mechatronic production systems". In: *Production Engineering* 1 (2007), pp. 371–379. URL: https://api.semanticscholar.org/CorpusID:9797203 (cit. on p. 2).

[5] R. Skýpala and R. Ruarovský. "The Role of a Behavioural Model for the Virtual Commissioning of Robotic Manufacturing Systems". In: *Research Papers Faculty of Materials Science and Technology Slovak University of Technology* 30 (2022), pp. 45 –52. URL: https://api.semanticscholar.org/CorpusID:251744549 (cit. on pp. 2, 18, 20).

[6] M. Soori, B. Arezoo, and R. Dastres. "Advanced virtual manufacturing systems: A review". In: *Journal of Advanced Manufacturing Science and Technology* (2023) (cit. on pp. 2, 18, 20).

[7] S. J. Oks et al. "Cyber-Physical Systems in the Context of Industry 4.0: A Review, Categorization and Outlook". In: *Information Systems Frontiers* (2022) (cit. on pp. 6, 9).

[8] L. Xu, E. L. Xu, and L. X. Li. "Industry 4.0: state of the art and future trends". In: *International Journal of Production Research* 56 (2018), pp. 2941 –2962 (cit. on pp. 6, 7).

[9] EPSI. 2013. URL: https://www.etsi.org/ (cit. on p. 7).

[10] K Pertz. *The Next Evolution of the Internet*. 2013. URL: http://theinstitute.ieee.org/technology-focus/technology-topic/the-next-evolution-of-the-internet (cit. on p. 7).

[11] L. Xu, W. He, and S. Li. "Internet of Things in Industries: A Survey". In: *IEEE Transactions on Industrial Informatics* 10 (2014), pp. 2233–2243 (cit. on p. 7).

[12] K. Ashton. "That 'Internet of Things' Thing". In: 1999 (cit. on p. 7).

[13] Q. Zhang, L. Cheng, and R. Boutaba. "Cloud computing: state-of-the-art and research challenges". In: *Journal of Internet Services and Applications* 1 (2010), pp. 7–18 (cit. on pp. 7, 8).

[14] P. Mell and T. Grance. "The NIST Definition of Cloud Computing". In: 2011 (cit. on p. 8).

[15] C. Hildebrandt et al. "Ontology Building for Cyber–Physical Systems: Application in the Manufacturing Domain". In: *IEEE Transactions on Automation Science and Engineering* 17 (2020), pp. 1266–1282 (cit. on p. 8).

[16] H. Schlingloff, H. Stubert, and W. Jamroga. "Collaborative embedded systems - a case study". In: *2016 3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)* (2016), pp. 17–22 (cit. on p. 8).

[17] Q. Zhang et al. "Diversity-by-Design for Dependable and Secure Cyber-Physical Systems: A Survey". In: *IEEE Transactions on Network and Service Management* 19 (2020), pp. 706–728 (cit. on p. 9).

[18] A. W. Colombo et al. "Industrial Cyberphysical Systems: A Backbone of the Fourth Industrial Revolution". In: *IEEE Industrial Electronics Magazine* 11 (2017), pp. 6–16 (cit. on p. 9).

[19] L. Monostori. "Cyber-physical production systems: Roots, expectations and R&D challenges". In: *Procedia CIRP* 17 (2014), pp. 9–13 (cit. on p. 9).

[20] W. de Paula Ferreira, F. Armellini, and L. A. de Santa-Eulalia. "Simulation in industry 4.0: A state-of-the-art review". In: *Comput. Ind. Eng.* 149 (2020), p. 106868 (cit. on pp. 9, 25).

[21] A. P. G. Scheidegger et al. "An introductory guide for hybrid simulation modelers on the primary simulation methods in industrial engineering identified through a systematic review of the literature". In: *Comput. Ind. Eng.* 124 (2018), pp. 474–492 (cit. on pp. 9, 10, 14–17).

[22] L. Rabelo et al. "Enterprise simulation: a hybrid system approach". In: *International Journal of Computer Integrated Manufacturing* 18 (2005), pp. 498 –508 (cit. on pp. 10–12).

[23] M. El-Gafy and T. S. Abdelhamid. "Using System Dynamics Modeling as a Lean Construction Work Structuring Tool". In: 2008 (cit. on p. 10).

[24] M. Hybinette et al. "SASSY: A Design for a Scalable Agent-Based Simulation System using a Distributed Discrete Event Infrastructure". In: *Proceedings of the 2006 Winter Simulation Conference* (2006), pp. 926–933 (cit. on p. 10).

[25] W. Ross, M. Ulieru, and A. Gorod. "A multi-paradigm modelling & simulation approach for system of systems engineering: A case study". In: *2014 9th International Conference on System of Systems Engineering (SOSE)* (2014), pp. 183–188 (cit. on p. 10).

[26] M. Barbati, G. Bruno, and A. Genovese. "Applications of agent-based models for optimization problems: A literature review". In: *Expert Syst. Appl.* 39 (2012), pp. 6020–6028 (cit. on p. 12).

[27] B. J. Malani and A. B. M. Sultan. "An introductory of a content provider agent in higher learning institutions". In: *International Journal of Computer Science Issues (IJCSI)* 8.4 (2011), p. 48 (cit. on p. 12).

[28] S. Abar et al. "Agent Based Modelling and Simulation tools: A review of the state-of-art software". In: *Comput. Sci. Rev.* 24 (2017), pp. 13–33 (cit. on p. 12).

[29] S. C. Brailsford et al. "Hybrid simulation modelling in operational research: A state-of-the-art review". In: *Eur. J. Oper. Res.* 278 (2019), pp. 721–737. URL: https://api.semanticscholar.org/CorpusID:125919858 (cit. on p. 13).

[30] N. Mustafee et al. "Purpose and benefits of hybrid simulation: Contributing to the convergence of its definition". In: *2017 Winter Simulation Conference (WSC)* (2017), pp. 1631–1645. URL: https://api.semanticscholar.org/CorpusID:13806879 (cit. on p. 13).

[31] Z. Huang et al. "A Survey on AI-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics". In: *Sensors (Basel, Switzerland)* 21 (2021) (cit. on p. 17).

[32] E. H. Glaessgen and D. Stargel. "The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles". In: 2012 (cit. on p. 17).

[33] A. A. Neto et al. "Digital twins in manufacturing: An assessment of key features". In: *Procedia CIRP* 97 (2021), pp. 178–183 (cit. on p. 17).

[34] M. Grieves. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. 2014 (cit. on p. 17).

[35] M. Grieves and J. Vickers. "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems". In: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Ed. by F.-J. Kahlen, S. Flumerfelt, and A. Alves. Cham: Springer International Publishing, 2017, pp. 85–113. ISBN: 978-3-319-38756-7. DOI: 10.1007/978-3-319-38756-7_4. URL: https://doi.org/10.1007/978-3-319-38756-7_4 (cit. on p. 17).

[36] M. Kopek, G. Krianová, and R. Ruarovský. "A Brief Survey of Creating a Digital Twin Architecture for Virtual Commissioning on the Machine Level". In: *Research Papers Faculty of Materials Science and Technology Slovak University of Technology* 30 (2022), pp. 71 –80. URL: https://api.semanticscholar.org/CorpusID:251744486 (cit. on pp. 17, 18).

[37] C.-G. Lee and S. C. Park. "Survey on the virtual commissioning of manufacturing systems". In: *J. Comput. Des. Eng.* 1 (2014), pp. 213–222. URL: https://api.semanticscholar.org/CorpusID:26494868 (cit. on p. 17).

[38] T. Lechler et al. "Virtual Commissioning – Scientific review and exploratory use cases in advanced production systems". In: *Procedia CIRP* (2019). URL: https://api.semanticscholar.org/CorpusID:198344402 (cit. on pp. 17–20).

[39] P. Hoffmann et al. "Virtual commissioning of manufacturing systems a review and new approaches for simplification." In: *ECMS* (2010), pp. 175–181 (cit. on p. 17).

[40] M. Ahrens et al. "Novel approach to establish model-based development and virtual commissioning in practice". In: *Engineering with Computers* 35 (2019), pp. 741–754 (cit. on p. 17).

[41] P. Milgram and F. Kishino. "A Taxonomy of Mixed Reality Visual Displays". In: *IEICE Transactions on Information and Systems* 77 (1994), pp. 1321–1329 (cit. on p. 21).

[42] D. W. F. van Krevelen and R. Poelman. "A Survey of Augmented Reality Technologies, Applications and Limitations". In: *Int. J. Virtual Real.* 9 (2010), pp. 1–20 (cit. on pp. 21, 24).

[43] P. Fraga-Lamas et al. "A Review on Industrial Augmented Reality Systems for the Industry 4.0 Shipyard". In: *IEEE Access* 6 (2018), pp. 13358–13375 (cit. on p. 22).

[44] H. Regenbrecht et al. "Using Augmented Virtuality for Remote Collaboration". In: *Presence: Teleoperators & Virtual Environments* 13 (2004), pp. 338–354 (cit. on pp. 22, 24).

[45] L. P. Berg and J. M. Vance. "Industry use of virtual reality in product design and manufacturing: a survey". In: *Virtual Reality* 21 (2017), pp. 1–17 (cit. on pp. 22, 24).

[46] M. K. Bekele et al. "A Survey of Augmented, Virtual, and Mixed Reality for Cultural Heritage". In: *Journal on Computing and Cultural Heritage (JOCCH)* 11 (2018), pp. 1 –36 (cit. on p. 22).

[47] I. Karlsson et al. "Combining augmented reality and simulation-based optimization for decision support in manufacturing". In: *2017 Winter Simulation Conference (WSC)* (2017), pp. 3988–3999. URL: https://api.semanticscholar.org/CorpusID:12652996 (cit. on pp. 27, 29).

[48] A. Kokkas and G.-C. Vosniakos. "An Augmented Reality approach to factory layout design embedding operation simulation". In: *International Journal on Interactive Design and Manufacturing (IJIDeM)* (2019), pp. 1–11. URL: https://api.semanticscholar.org/CorpusID:150080147 (cit. on pp. 27, 29).

[49] O. Havlíek. "Seamless integration of augmented reality into digital workflow of virtual commissioning". In: 2019. URL: https://api.semanticscholar.org/CorpusID:195462428 (cit. on pp. 27, 29).

[50] M. Schumann et al. "Augmented Reality Support for Commissioning and Monitoring of Electromechanical Multipoint Die Cushion". In: *The 28th Saxon Conference on Forming Technology SFU and the 7th International Conference on Accuracy in Forming Technology ICAFT* (2022). URL: https://api.semanticscholar.org/CorpusID:253442062 (cit. on pp. 28, 29).

[51] T. J. Team. *JADE (Java Agent DEvelopment Framework)*. URL: https://jade.tilab.com/ (cit. on p. 44).

[52] *Tutorial 1: JADE Architecture Overview*. URL: https://jade.tilab.com/documentation/tutorials-guides/jade-administration-tutorial/architecture-overview/ (cit. on p. 45).

[53] G. Caire. *JADE TUTORIAL - JADE PROGRAMMING FOR BEGINNERS*. 2009. URL: https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf (cit. on p. 45).

[54] F. Bellifemine et al. *JADE PROGRAMMER'S GUIDE*. 2010. URL: https://jade.tilab.com/doc/programmersguide.pdf (cit. on p. 46).

[55] K. GmbH. *RevPi Cores Datasheet*. URL: https://revolutionpi.com/wp-content/uploads/manuell/datenblatt/Datasheet_RevPi_Cores.pdf (cit. on p. 47).

[56] K. GmbH. *RevPi DIO Datasheet*. URL: https://revolutionpi.com/wp-content/uploads/manuell/datenblatt/Datasheet_RevPi_DIO.pdf (cit. on p. 47).

[57] *Open Source IPC*. URL: https://revolutionpi.com/revpi-core (cit. on p. 47).

[58] *Digital IO Modules*. URL: https://revolutionpi.com/io-modules (cit. on p. 47).

[59] *Conveyor Belt 24v*. URL: https://www.fischertechnik.biz/conveyor-belt-24-1 (cit. on p. 49).