Degree Project

Connected Digital Twin for Robotic Battery Assembly and Disassembly

Bachelor Degree Project in Industrial Engineering
First Cycle 30 credits
Spring term 2025

Student: Alberto Prats Ferrandis
Student: Selene Delgado Pastor
Supervisor: My Legendi
Examiner: Anna Syberfeldt

Abstract

This project presents the design and implementation of a digital twin-based user interface for the real-time control of a UR10e robotic arm in an industrial assembly context, specifically targeting robotic cell box assembly and disassembly. The user interface, developed using the PyQt6 framework and integrated through RoboDK, enables manual and automated control, speed and gripper adjustments, and visual feedback of the robotic process. This work aims to show the benefits of the implementation of digital twins, and ultimately demonstrates a practical and flexible solution for enhancing human-robot interaction in smart manufacturing, bringing the feasibility and effectiveness of combining digital twins with the industry, ensuring optimal control, advanced user interface design, and the ability to control complex systems.

Keywords: Digital Twin (DT), UR10e, Robotic Assembly, RoboDK, Simulation, Real-time Synchronization, Remote Robot Control, RobotiQ, Gripper, User Interface (UI), Process Optimization, Python, PyQt6, CasADi.

Certification

Certification
and
Declaration on the Use of Artificial Intelligence (AI) Tools during the degree project

This report has been submitted by *Selene Delgado Pastor and Alberto Prats Ferrandis* to the University of Skövde as a requirement for the degree of Bachelor of Science in Production Engineering.

The undersigned certifies that all the material in this report that is not our own has been properly acknowledged using accepted referencing practices and, further, that the report includes no material for which we have previously received academic credit.

As part of our commitment to academic integrity and transparency, we hereby declare the extent to which we have used Artificial Intelligence (AI) tools during the preparation and writing of this report. This declaration does not apply to the use of basic tools for checking grammar, spelling, references etc.

Please check all that apply:

☐ **No use of AI tools**
We have not used any AI-based tools or services in any part of the research, writing, analysis, or presentation of this report.

☑ **Language editing assistance only**
We have used AI tools exclusively for language correction, grammar checking, or stylistic improvements. No substantive content was generated or modified by AI.

☐ **Idea generation / brainstorming**
We used AI tools to help generate initial ideas, brainstorm possible research questions, or explore conceptual frameworks. The final work and conclusions are my own.

☐ **Methodology design**
We used AI tools to assist in designing the research methodology (e.g., structuring experiments, surveys, or models). The methodology has been critically reviewed and adapted by me.

☑ **Programming or code assistance**
We used AI-based coding assistants to help write or debug code. All code has been reviewed, tested, and modified by me.

☐ **Data analysis / processing support**
We used AI tools to assist with data analysis, visualization, or statistical processing. Interpretations and conclusions were drawn independently.

Certification

☐ **Content generation (text, figures, tables)**
AI tools were used to generate portions of text, diagrams, or tables that appear in this report. Such content has been critically reviewed, fact-checked, and appropriately cited where applicable.

☐ **Other (please specify):**

_____

_____

_____

Responsibility Statement
We confirm that all AI-assisted contributions have been used ethically and in accordance with institutional guidelines. We take full responsibility for the accuracy, originality, and academic integrity of the content presented in this report.

We understand that any misuse or misrepresentation of AI tool usage may be considered a breach of academic conduct and could result in disciplinary actions, including revocation of the report and/or the academic degree awarded.

Signature: 

Name: Selene Delgado Pastor

Date: 20/05/2025

Signature: 

Name: Alberto Prats Ferrandis

Date: 20/05/2025

Contents

List of Figures

List of Acronyms

AI Artificial Intelligence 4, 13–15

API Application Programming Interface 22, 32, 34, 41

BMS Battery Management Systems 12

CCTV Closed-circuit television 8

CPS Cyber-Physical Systems 8

CSV Comma-Separated Values 31, 36, 61

DSRM Design Science Research Methodology vii, 4, 16, 18, 20, 21, 46, 47

DT Digital Twin i, 1–4, 6–8, 10–18, 20–24, 30, 31, 33, 34, 36, 39, 41, 42, 45–53, 60, 65

DTE Digital Twin Environment 7

DTI Digital Twin Instance 7

DTP Digital Twin Prototype 7

GDPR General Data Protection Regulation 15

HMI Human-Machine Interface 13, 33

IDE Integrated Development Environment 57

IIoT Industrial Internet of Things 8

IoT Internet of Things 4, 7–9, 11, 14

IP Internet Protocol 8, 31, 34

ML Machine Learning 7, 13, 14

MQTT Message Queuing Telemetry Transport 4

NASA National Aeronautics and Space Administration 2

NLP Non-Linear Programming 35, 62

RFID Radio-frequency identification 8

ROS Robot Operating System 4, 22

SDG Sustainable Development Goal 50

SM Smart Manufacturing 8

SoC State of Charge 12

SoT State of Health 12

TCP Tool Center Point vii, 30, 34, 35, 37, 39, 40, 44, 63, 64

TCP Transmission Control Protocol 8

UI User Interface i, v, vii, 3, 17, 20, 30–34, 36, 37, 39–43, 46, 50–53, 60, 61, 63–67

UML Unified Modeling Language vii, 33, 60

UR Universal Robot 10, 22, 31, 57

USB Universal Serial Bus 60

VR Virtual Reality 12, 13, 15, 53

WSN Wireless Sensor Networks 8

1. Introduction

This report is structured to provide a comprehensive examination of the development and implementation of a real-time connected digital twin for a UR10e robotic system in battery assembly. It follows a logical progression, beginning with the background and theoretical foundations, moving through the design, development, and implementation phases, and concluding with an analysis of the results and future considerations.

1.1. Background

Since the 1990s, lithium batteries have been the most common form of rechargeable battery (Sparke, 2024). The way these batteries are made is constantly improving to increase their performance due to the high demand currently driven by electric vehicles for their environmental benefits and energy efficiency. The increase in their performance may be due to a decrease in their energy density or an extension of the cell's lifespan.

Batteries are traditionally governed by an imprecise reference table that indicates how much current is applied to a cell depending on temperature and state of charge (Stevens, 2025). The environmental conditions in which batteries are handled influence their performance and state, making their manufacturing a delicate process. For this reason, automated assembly ensures that it is highly precise as well as safe (A. Singh, 2025).

The assembly of batteries for electric vehicles is a process that requires precision and efficiency. The automation of the process in the industry is allowing them to be produced at a higher speed, with greater safety and quality, thanks to robotic technology (A. Singh, 2025). Also, the automation of the manufacturing process reduces material costs and improves consistency by reducing human error. However, errors and inconsistencies due to assembly tolerances still occur and must be resolved. Additionally, each battery requires specialization, as it has its own characteristics and specifications for assembly.

Due to high demand and according to A. Singh (2025), manufacturers have limited resources to meet it, even though they need precise control at every stage of production and real-time adjustments. The need for battery improvement requires intelligent solutions, as the process is becoming increasingly subtle and critical.

Based on M. Attaran and Celik (2023), to meet these needs a Digital Twin (DT) would serve as a powerful tool to track, optimize, and control products. Furthermore, due to data collection during production, the process could be optimized, as well as reduce the ecological footprint and material consumption without the need to stop production or invest expenses in it. In this way, the manufacturing industry would shift from being reactive to predictive, being able to identify machine performance and improve it, predict machine operation when it is wearing out or has detected a failure, and even redesign its mode of operation to make better use of it.

The concept of DTs emerged in the early 2000s when Grieves and Vickers (2016) introduced the first DT concept, initially known as 'mirrored spaces models,' for a product life cycle management course. In this, he addressed the real space, the virtual space, and the flow of information between them (Dubarry, Howey, and Wu, 2023).

However, as stated by M. Singh et al. (2021), National Aeronautics and Space Administration (NASA) used the term 'Digital Twin (DT)' for the first time in a preliminary version of the technology roadmap, at that time known as 'Virtual Digital Fleet Leader', defined as "an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin." Although its first mention dates back to the 1970s, when NASA was working on the Apollo program by simulating a spacecraft on the ground, allowing them to work with it by recreating problems and implementing possible solutions, thus replicating the real behavior of an original spacecraft. Since then, up to the present day, the concept continues to be discussed and modified by the industry.

1.2. Problem statement

In modern manufacturing environments, robotics plays a critical role in automating assembly processes, improving efficiency, and reducing human intervention in repetitive tasks. However, one of the primary challenges in robotic automation is the lack of real-time synchronization and predictive capabilities between physical robotic systems and their virtual representations. Offline programming and simulation techniques frequently struggle to accurately and dynamically reflect the real-world behavior of robots, resulting in challenges related to monitoring, control, and predictive maintenance.

Without real-time data exchange, manufacturing systems lack the ability to detect faults, optimize processes, and adjust operations dynamically (Tao et al., 2019). This gap results in increased downtime, inefficient programming, and difficulties in predictive maintenance, ultimately affecting production reliability and cost-effectiveness. Therefore an absence of a fully connected DT that enables bi-directional communication between the physical and virtual systems can be a major limitation in robotic assembly lines.

1.3. Aim and objectives

This project aims to develop a real-time connected DT for a UR10e robotic system performing battery assembly and disassembly tasks. The DT will enable live synchronization between physical and virtual models, providing real-time insights into robotic operations, predictive analytics, and system optimization. By integrating software control, the project seeks to enhance monitoring, control accuracy, and decision-making within robotic assembly and disassembly processes. In order to achieve this aim, the project must be divided into smaller objectives, which are to be tackled separately:

1. **System architecture**

   Define the real-time digital twin architecture by identifying and analyzing the necessary software and hardware. Relevant literature and research, along with technical specifications, will form the basis of the analysis.

2. **Virtual model**

   Develop an accurate 3D simulation of the robotic station and/or battery assembly process by taking measurements of the physical robot station. Establish the bi-directional connection between the physical robot and the virtual model. It is crucial to ensure that the virtual model accurately mirrors the physical robot.

3. **User interface**

   Design and produce an intuitive and functional graphical UI that allows for both the monitorization and control of the physical robot.

4. **Advanced computation**

   Carry out the proposed software data analysis to a satisfactory degree, this refers to both the trajectory refining and joint data extraction for maintenance determination, as well as any additional functionalities that might be thought to be relevant. The trajectory refining will give the option to select the algorithmically calculated trajectory in substitution to the predetermined trajectory method. The maintenance determination will use data extracted from the robot to help the user visualize the joint usage and identify abnormalities in motors, or help the user determine when maintenance should be carried out.

5. **Testing and validation**

   Certify the correct evaluation of the synchronization accuracy, latency in data exchange, and general functionality of the digital twin system.

6. **Analysis**

   Produce a conclusive thought to the project, documenting performance results, potential optimizations, as well as comparing the final conclusions with existing research and identifying areas for future improvements.

1.4. Limitations

The project is limited to the development and implementation of a DT for the UR10e robotic system within a controlled laboratory environment. The study will focus on a single robotic process (the battery assembly and disassembly line) without considering integrating previous and subsequent production lines. This controlled setting allows for detailed analysis and optimization, but it does not fully take into account production flow variations and how external processes' coordination may influence performance.

Additionally, the DT will be designed for process monitoring, control, and data processing, but it will not include advanced Artificial Intelligence (AI)-driven decision-making or autonomous adaptation beyond predefined parameters. While real-time data exchange will be implemented, alternative communication protocols such as Message Queuing Telemetry Transport (MQTT) or Robot Operating System (ROS)-based systems will not be explored in depth. Furthermore, the focus will remain on the robotic system itself, without extensive consideration of human-robot collaboration or its implications in an industrial workspace. The project will emphasize functional validation and short-term performance evaluation rather than long-term stress testing or deployment in a production environment.

1.5. Overview

Introduction (Chapter 1) introduces the topic of the project, outlines the background and industrial context, defines the research problem, and states the main aim and objectives. It also presents the limitations that define the scope of the study and concludes with this overview to guide the reader through the structure of the report.

Theoretical Basis and Frame of Reference (Chapter 2) presents the theoretical foundation of the project, introducing key concepts relevant to Digital Twin (DT) development. It covers topics such as Internet of Things (IoT), smart manufacturing, DT, and collaborative robots. These are explored in the context of their application to robotic systems, particularly in battery assembly, and provide the knowledge base on which the project is built.

Literature Review (Chapter 3) examines existing research on Industry 4.0 applications, digital twinning, and related technologies. It also examines the roles of virtual reality and artificial intelligence in digital twinning, identifying trends, opportunities, and limitations in current studies. This critical review positions the current project within the broader academic and industrial landscape.

Methodology (Chapter 4) outlines the research methodology used to structure and execute the project. Based on the DSRM, it describes the approach to problem identification, objective formulation, system design, testing, and validation. It ensures a structured, iterative, and scientifically sound process for developing the digital twin system.

The Implementation (Chapter 5) describes the technical development and system integration that enables the digital twin's functionality. It also describes the integration of the virtual and physical systems and the synchronization mechanisms between them.

The Results (Chapter 6) present the findings of the study, including performance evaluations and system validation. The outcomes are analyzed in relation to the project objectives to assess the success of the implementation.

The Discussion (Chapter 7) provides an in-depth discussion of the results, highlighting the strengths and limitations of the developed system. It connects the findings to existing research, suggests possible improvements, and outlines directions for future work.

The Conclusion (Chapter 8 summarizes the process and development of the whole thesis, providing a conclusive thought upon the project.

The Future Work (Chapter 9 dwells into the possibility of further development and extension of the project, stating the areas that are recommended to expand upon and opportunities to implement further functionalities.

This structure ensures a comprehensive and logical progression from conceptual foundations to practical implementation and evaluation.

2. Theoretical basis and frame of reference

The development of a real-time connected DT for a UR10e robotic system is grounded in several key technological and theoretical concepts that define its structure, functionality, and implementation. This section introduces the foundational principles that support the research, with detailed explorations of each concept. By providing an overview of these interconnected concepts, this section establishes the scientific foundation for the project, demonstrating how they collectively contribute to the development of an effective real-time digital twin. With this, the reader will have a better understanding of the technical concepts necessary to comprehend the thesis.

2.1. Digital Twin

DTs are cyber-physical systems that combine and connect real-time data captured by sensors with different physical models (Dubarry, Howey, and Wu, 2023). They can make accurate predictions, as well as optimal and intelligent decisions, thus representing the real behavior of a physical or virtual object in the digital world with precision (Dubarry, Howey, and Wu, 2023). The physical object must be transmitting its information to the virtual model to achieve a real-time representation, and viceversa, so that all changes made in the virtual world are received by the physical object. The exchange of data must be a transparent and standardized process in which stakeholders must respect confidentiality, thus obtaining feedback from the virtual world (Fu et al., 2022).

2.1.1 Digital Twin according to their level of integration

According to Kritzinger et al. (2018), DTs can be classified into three subcategories based on their level of integration, as can be seen in figure 2.1:

- Digital model: The exchange of information between objects is manual, and therefore, when a change is applied to physical objects, it is not directly reflected in the virtual object and vice versa.
- Digital shadow: The physical object's data is sent directly to the virtual object, but the information flow from the virtual object to the physical one remains manual, so the applied changes in the virtual object are not reflected in the physical one.
- Digital twin: The data flow between both objects, physical and virtual, is automatic and bidirectional, ensuring that changes in either object are directly reflected in the other.

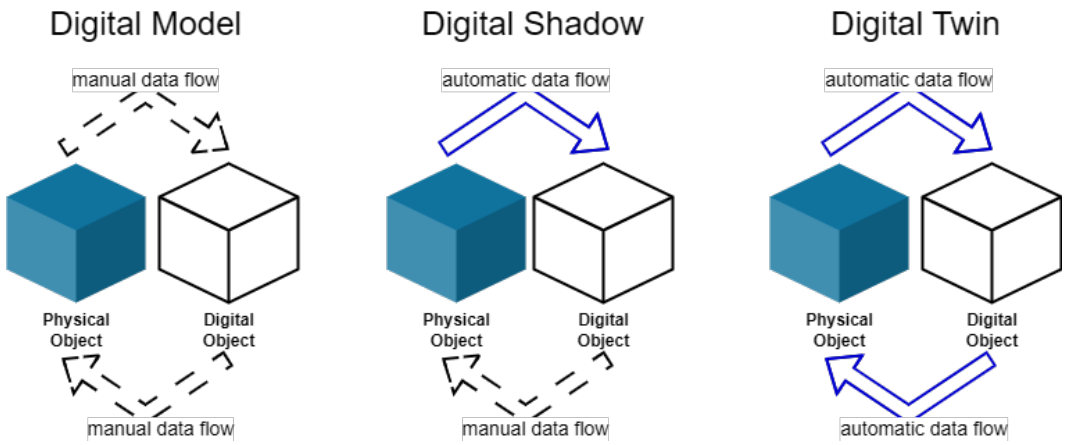


Figure 2.1: Types of Digital Twin inspired by(Andrade, 2024)

2.1.2 Digital Twin depending on how they were created

Grieves and Vickers (2017) consider that DTs can be classified into two types depending on how they were created within the Digital Twin Environment (DTE):

- Digital Twin Prototype (DTP): It gathers all the necessary information to construct the physical digital twin from the virtual one. The DTP includes models and designs and undergoes different tests where undesirable scenarios are identified to avoid or solve them. Once validated, the process of constructing the physical twin in the real world begins.
- Digital Twin Instance (DTI): The digital twins of both the real and virtual worlds have already been created and are monitored to observe the data flow between them and predict system behavior. Predicted and undesirable behaviors can be identified, verifying whether the outcome is as expected and allowing for any changes in one of the two objects, which will be directly transmitted to the other.

2.2. Smart manufacturing

Based on Qi and Tao (2018), the increase in the digitization of manufacturing, composed of structured, semi-structured, and unstructured data, is creating opportunities for smart manufacturing. Smart manufacturing can be achieved thanks to big data analysis, which enables manufacturers to identify bottlenecks in their processes, understand the impact of their problems by finding their causes, and find solutions to them to enhance efficient and competitive manufacturing.

According to B. Wang et al. (2021), manufacturing has gone through different stages up to the present day. The first stage, before 2000, was digital manufacturing, where computers were used to support machines with some use of decision-tree expert systems. After the 2000s, the second stage arrived: smart manufacturing, where digital manufacturing leverages networks to adapt to a dynamic environment and customer needs, focusing on analysis and control. Later, after 2020, the third stage emerged: intelligent manufacturing, which uses IoT, Machine Learning (ML), and big data to better integrate human-machine systems. Depending on different perspectives, various

definitions of Smart Manufacturing (SM) have been made:

- Engineering view: it is the application of advanced smart technologies that enable the rapid and stable manufacturing of new products, dynamic response to customized product manufacturing, and real-time optimization of production.
- Networking view: it is the application of IoT, Industrial Internet of Things (IIoT), and Cyber-Physical Systems (CPS), enabled by sensors and communication technologies that capture data at all levels and stages of manufacturing.
- Decision-making view: it is the application of SM using the accessibility and ubiquity of the information domain to help companies improve their predictions, maintain their production processes, and enhance productivity. Based on big data analytics, it optimizes manufacturing processes.

Qi and Tao (2018) considers the combination of DT and big data a great tool for the development of smart manufacturing. DTs allow real-time management of physical objects and their virtual representation, creating a bridge for cyber-physical integration, which, along with the analysis and predictions enabled by big data through its large volume of collected data, would lead to more reasonable and precise manufacturing.

2.3. Internet of Things

According to Gokhale, O. Bhat, and S. Bhat (2018), the concept of a network of smart devices was discussed in 1982 when a Coca-Cola machine at Carnegie Mellon University was connected to Internet, being able to report on its inventory and whether the drinks were cold. However, the concept of 'the Internet of Things (IoT)' was invented by Kevin Ashton in 1999, who described it as a system where the internet is connected to the physical world through ubiquitous sensors.

The words 'Internet' and 'things' refer to an interconnected global network that uses the standard Internet protocol (Transmission Control Protocol (TCP)/Internet Protocol (IP)) based on sensors, communication, networks, and information processing. Wireless Sensor Networks (WSN), barcodes, intelligent sensing, and Radio-frequency identification (RFID) are, among others, the technologies involved in IoT (Gokhale, O. Bhat, and S. Bhat, 2018)(Madakam, Ramaswamy, and Tripathi, 2015).

The IoT transforms real-world objects into intelligent virtual objects. Thanks to IoT, all devices will be unified under the same infrastructure, which will allow for control as well as knowledge of the state of things (Madakam, Ramaswamy, and Tripathi, 2015). In addition, IoT is constantly updating the information of the already connected devices and increasing the volume of usable data (M. Attaran, S. Attaran, and Celik, 2023). Within IoT, physical and virtual things have their own identity and attributes and are capable of using intelligent interfaces, which are connected thanks to their unique identification (Gokhale, O. Bhat, and S. Bhat, 2018). To achieve a seamless IoT, three components are required according to Madakam, Ramaswamy, and Tripathi (2015):

- Hardware: composed of sensors, actuators, IP cameras, Closed-circuit television (CCTV), and embedded communication hardware.

- Middleware: on-demand storage and computing tools for data analytics with the cloud and Big Data Analytics.
- Presentation: visualization and interpretation tools that are easy to understand, possibly designed for different applications.

Based on Gokhale, O. Bhat, and S. Bhat (2018), a requirement for an IoT is that the 'things' must be interconnected to each other in the network. For this, the IoT architecture must be adaptive and ensure a bridge between the physical and virtual worlds, so that the 'things' can move physically while needing to interact with each other in real-time mode dynamically. The structure is composed of the following layers, as observed in the figure 2.2:

- Sensing Layer: composed of all available objects to detect their status.
- Network Layer: being the infrastructure that supports wireless and wired connections between things.
- Service Layer: responsible for creating and managing the services needed by users or applications.
- Interface Layer: which consists on the interaction methods with users or applications.

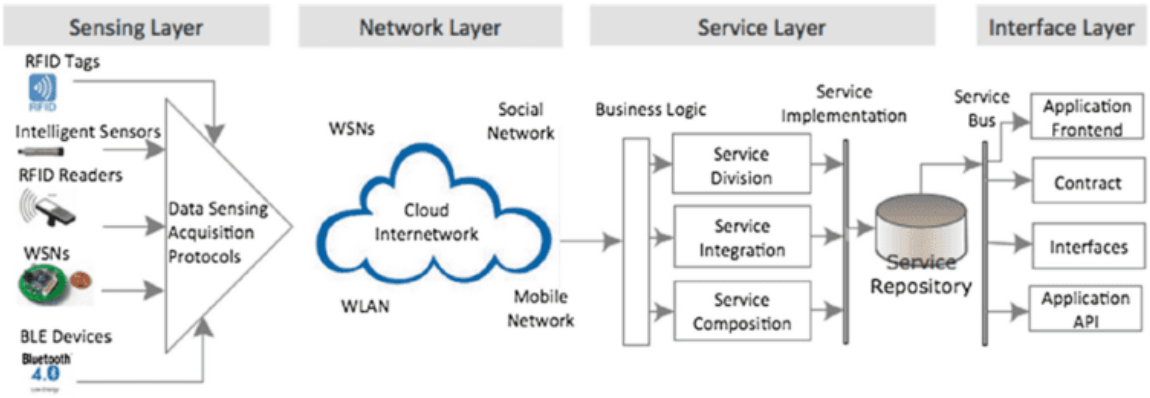


Figure 2.2: Arquitectural Layers of IoT (Pericherla, 2025)

2.4. Collaborative Robots

Based on Kragic et al. (2018), robotic technology has been evolving over the years since the first industrial robot was used in the 1960s. Over time, and even today, a solution is still being developed where production lines can be quickly restructured and adapted to a new product or a minimally modified version of the same one. Due to this, the integration of industrial robots, which are still sometimes reprogrammed and have little capacity to adapt to errors, along with humans in a shared space where they perform different tasks collaboratively, is being pursued.

Kragic et al. (2018) ensures in his study that collaborative robot systems allow a human to take control in tasks that are complex for the robot. Likewise, for human-

robot collaboration to be efficient, the robot must perform both active and passive parts of the interaction, just as a human would. The passive part occurs when the robot responds to the human without taking initiative when the human takes control. The active part occurs when it can plan and execute appropriate trajectories for objects, have information about the human, their internal state, and the constraints of the object it will manipulate. Additionally, the flexible use of multisensory data in real-time would also be necessary.

Sherwani, Asad, and Ibrahim (2020) also states that the purpose of collaborative robots is to perform tasks alongside humans while sharing the same workspace, thereby increasing mobility and flexibility. These machines are designed to be easily reprogrammed so that any person without programming knowledge can use them.

2.4.1 UR10e

Universal Robot (UR) is a collaborative robot brand that combines both long reach and high payload, offering endless automation possibilities thanks to its powerful system of Universal Robot (UR)+ components and kits (UR, 2025). Additionally, it is quick and easy to reprogram, allowing for flexible configuration changes at any time. It is characterized by its high productivity—eliminating errors by introducing automated procedures; its agility—creating new solutions and reusing programs for different tasks; and its employee support—giving them the freedom to focus on what they consider important while leaving the rest to the robot (UR, 2025).

2.5. Software

According to Amaya (2010), software consists of the detailed instructions that control the operation of a computational system, characterized by managing hardware resources, providing the necessary tools for this, and acting as an intermediary between organizations and stored information. For the development of this thesis, two software programs have been considered for implementation and to create the DT with them. These are:

1. The simulation software RoboDK (2025a), which is a powerful and effective simulator that allows to get the most out of industrial robots and robot programming. It doesn't need prior knowledge or programming skills due to its intuitive interface, which enables the worker to easily program any robot offline. Additionally, robots can be programmed outside the production environment directly from your computer.
2. The simulation software Simumatik (2025a), which is a cloud-based engineering software for innovation and creativity, that provides the ability to create any component with its respective geometry, physics, and behavior, as well as the capability to connect components to build a system.

3. Literature review

In this chapter, an explanation of the use of digital twins in different work areas is carried out. Thanks to the compilation of information provided by various papers and studies, the reader gains a better understanding of the current role of DTs.

3.1. Digital Twin in Industry 4.0

In Industry 4.0, based on Fuller et al. (2020) DTs are a tool originating from the aerospace industry, expected to revolutionize many industries due to the advantage they offer in efficiency, productivity, and competitiveness against their competitors. It says that a DT is the virtual representation of a physical model that mimics its behavior through a real-time data exchange. This exchange between the physical and virtual worlds provides an advantage in unpredictable scenarios with realistic measures. The physical and virtual connection facilitates seamless integration between the IoT and data analysis.

According to Grieves and Vickers (2017), due to real-time data exchange, as well as the transfer of historical and environmental data, it is possible to track the state of the DT, understand its current conditions, predict its behavior, or design new processes or products, among other functionalities. They also suggest using the DT to design the production of the next generation of a product once it is discarded. The lack of real-time data makes a model static due to the absence of movement, as well as the inability to generate new predictions until new information is received. This completely results in the loss of the essence of a DT, making it more of a previous version such as a Digital Model or a Digital Shadow.

As Javaid, Haleem, and Suman (2023) mentions, the use of DT in the industry provides a digital image of factory operations, the activities of a communication network, or the movement of assets through a logistics system. It is possible to observe how the assets behave in real-time in order to make intelligent decisions about them with confidence by placing sensors on the physical asset, collecting information and creating its virtual duplicate, and applying the benefits of machines.

Based on M. Attaran, S. Attaran, and Celik (2023), in Industry 4.0, DTs are used to maintain competitiveness, improve the manufacturing chain, and provide visibility to the supply chain due to their usefulness in creating better products, detecting failures in a short time, or predicting outcomes more accurately. Due to their high range of applications, DT technology is a useful tool that can create opportunities in the manufacturing industry, among others.

3.2. Digital Twin for battery assembly

Fuller et al. (2020) mentions that manufacturers are always looking for ways to track their products as simply as possible, aiming for cost-effective and fast solutions, cre-

ating new opportunities for the future of the manufacturing industry. One way to achieve this would be to implement DTs in the industry so that physical and virtual objects would be connected in real time. The connectivity of the physical object to the virtual one, as well as its bidirectional connection, benefits manufacturers by providing real-time information about their machines and feedback on their production. With this information, they could improve their production process and anticipate possible system failures, preventing them in advance.

Due to the high demand for lithium-ion battery cells, the use of efficient processes is necessary. Also, thanks to the digitalization of Industry 4.0, more and more production processes use Digital Twin (DT)s (A. Singh, 2025) (M. Attaran and Celik, 2023).

In the study by Husseini et al. (2022), the use of DT for battery cell production is analyzed. Due to the ability of DTs to map processes virtually, the commissioning of machines could be analyzed, thus reducing the high production cost involved in doing it in the real world, and the low risk along with the different optimization approaches that virtual implementation offers. Subsequently, the study was developed by modeling the important components of the continuously operating flexible cell stacking machine along with the material behavior. Thanks to this, they were able to simulate different commissioning scenarios.

In the study by W. Wang et al. (2021), the real-time update of the battery model enables tracking of the charging and discharging behavior of the lithium battery as well as its internal behavior. Thanks to this, a prediction for electric vehicle batteries is achieved, aiming to manage their energy and safety, extend their lifecycle, and increase their efficiency.

Naseri et al. (2023) gathers information on the real benefits of DT in batteries and when it is not beneficial to apply those improvements due to their high cost compared to their results. It considers that promising results can be achieved using DTs for battery State of Health (SoT) estimation, using advanced algorithms that reduce the battery lifecycle cost, battery reuse, which adds value to the system as a whole and not just to electric vehicles, and predictive maintenance. Meanwhile, for State of Charge (SoC) estimation, it is not effective due to the current results obtained in Battery Management Systems (BMS), which are already sufficiently good.

3.3. Virtual Reality with Digital Twins

According to Inamura (2023), DTs facilitate collaboration between humans and robots by enabling them to work cooperatively serving as a bridge of communication between both. The complexity lies in the human's understanding of the feedback sent by the robot, as they do not always know how to use the received information to improve their behavior, making human-robot collaboration inefficient.

DT were once descriptive and static, but they have gradually transformed into a physical system considered an independent entity, transitioning into an actionable and experimentable digital twin. In the study by Pérez et al. (2020), the use of sensors and 3D visualization technologies, such as Virtual Reality (VR), is employed to access a

realistic model of the process state so that the DT uses the information provided by the sensors, and the 3D environment offers real-time visualization of the virtual object through VR glasses.

The work proposed by Pérez et al. (2020) aims to create a DT of the manufacturing process in which different objects collaborate with an immersive interface to simulate and analyze the layout, avoiding latencies between actions and feedback. In addition to the robotic station where different robots are located, safety parameters, motion detection sensors, and a Human-Machine Interface (HMI) for communication are subsystems to consider when creating the DT that will connect with the control system linking the different subsystems.

In another study, Kuts et al. (2019) understands the usefulness of VR in various industrial fields, including manufacturing, robotics, and architecture, or as a sound visualization tool for users ranging from artists to disabled individuals. The need for high-quality equipment, such as a powerful computer, is required to avoid hardware-related issues and latency, as well as the need for a specialized and trained team to manage the VR field and integrate high-quality 3D models with human interactions, which involve thoughts and different ways of thinking.

The study focuses on creating a DT that enhances workspace understanding using software controlled directly from VR, in this case, the Unity3D game engine, to achieve a flexible and user-friendly environment. The developed tool is not only intended for viewing the robotic cell along with the manufacturing line in real scale, but it is also an interactive tool that allows real-time reprogramming. Additionally, they present the use of a collision prediction system as an alternative to physical sensors.

A study developed by Inamura (2023) shows how intelligent robot systems are considered a derivation of DT in which a human and a conventional object DT are combined. This system is used in healthcare and sports training to serve as a support tool for humans, helping them improve. Due to the complexity of integrating these two DTs, Inamura (2023) proposes the integration of DT with VR for assistive robotic systems to create a more immersive experience that can help humans in their daily lives rather than just being a data transfer. In this study, virtual and augmented reality are considered potential improvements in human-robot interaction by providing images that allow users to experience the received information instead of merely receiving a set of numbers, thus intervening in a more direct manner.

3.4. Artificial Intelligence in Digital Twins

M. Attaran, S. Attaran, and Celik (2023) informs about the combination of DTs with Artificial Intelligence (AI). While DT collect information, AI calculates the impact of possible changes and an optimal range of parameters, as well as deciphers processes and complex manufacturing systems from the provided data, making predictions and suggesting solutions to potential issues.

Rathore et al. (2021) tells us that AI is a digital replica of three human cognitive skills: learning, reasoning, and self-correction. At the same time, it teaches us the difference between AI and ML; which is an AI method that searches for historical data

to take a decision, and deep learning; a ML technique focused on collecting historical information iteratively, inspired by biological neural networks. As a result, their study is based on the relationship between AI/ML and DT.

The study of Rathore et al. (2021) shows how IoT devices collect information from their environment in real-time, thus achieving an easy representation of the physical environment in virtual space. Since it is a large set of data, traditional techniques cannot be used, so it recommends collecting big data from the environment using IoT. This data is later introduced into an AI model, which allows detecting, predicting, optimizing, and making decisions dynamically based on the data from the created DT, ultimately making it capable of optimizing processes in the industry. This entire relationship is represented in the figure 3.1.

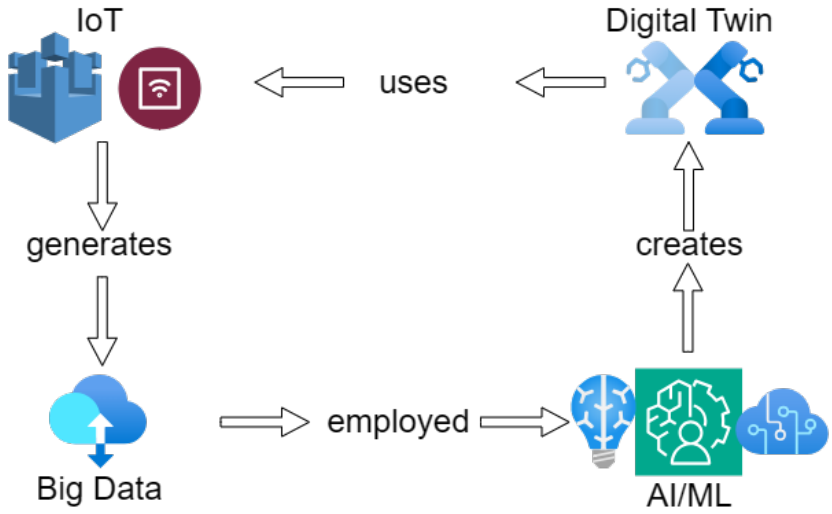


Figure 3.1: Relation IoT, BigData, DT and AI inspired by Rathore et al. (2021)

The study conducted by Lv and Xie (2022) gathers information obtained from various studies where DTs are combined with AI, observing their advantages in different environments. In the aerospace industry, how the use of DT of real aircraft simulates different flight situations, and by collecting that data, AI informs about the aircraft's problems and its current state. In autonomous driving, how the design of a DT that simulates a car and the driver performing various tests in a virtual environment ensures the safety of drivers, and together with algorithms created by AI, the development of an autonomous driving system is possible. In smart manufacturing, how applying AI in DT in manufacturing allows them to simulate, predict, optimize, and control different processes so that product quality is maximized and production costs are reduced. However, in all fields, we encounter challenges when simulating a large number of objects, as well as due to the lack of resources regarding the combination of both fields currently in the industry, its development requires a high investment.

Considerations to take into account when combining DT with AI include having a high-performance infrastructure in the form of updated software and hardware. This infrastructure should be capable of executing AI algorithms after the data capture performed by the DT Fuller et al. (2020). The collected data and information need to be clear and of high quality to integrate into AI algorithms.

Since it is a recent technology, this data collected and processed by AI does not have clear regulations regarding privacy. In Europe, the governing regulation is the General Data Protection Regulation (GDPR), which ensures the privacy and security of personal data (Fuller et al., 2020).

3.5. Analysis of the literature review

To sum up, although there are currently not many studies linking the use of DTs to battery assembly, a promising future is visible thanks to battery DT studies. Advancements are already being made there and are helping to increase battery production due to high demand, along with an improvement in their composition thanks to battery DTs, which allow for various tests to be conducted to identify weaknesses and strengthen their qualities.

All of this leads to the digitalization of industrial processes, which can be benefited by the use of DTs. With them, an advancement in manufacturing would be made as the state of the DT is tracked, achieving real-time data exchange, which is what is sought in this thesis. Although the benefits, such as creating better products, detecting failures, or predicting outcomes more accurately, are very appealing, this technology is under development and still has some weaknesses.

Moreover, the rapid evolution of Industry 4.0 and the increasingly frequent implementation of AI, as well as VR in the industry, are making the virtual world more extensive. Due to this, the combination of DTs with these two technologies is becoming more common, and the use of all three as a set is more and more applied in manufacturing due to their benefits, such as detecting errors, improving processes, transmitting information between the real and virtual worlds, among others.

4. Methodology

According to Creswell (2014), methodology refers to the overarching strategy that guides the research process, ensuring coherence between the research questions, data collection, and analysis methods. In the context of this project, the methodology is essential for guiding the development of a real-time DT for a UR10e robotic system, ensuring that each phase is carried out logically and efficiently.

This project must ensure to maintain a goal-driven, iterative, and structured approach, which allows for continuous refinement and validation of the DT system (Tao et al., 2019). Without a well-defined methodology, research efforts can become disorganized, inefficient, and difficult to validate, ultimately undermining the reliability and applicability of the developed solution (Hevner et al., 2004). Thus, a clear and well-documented methodology is crucial for achieving the project's objectives and ensuring its contributions to real-time robotics and smart manufacturing.

This project follows a DSRM. The DSRM methodology, as proposed by Peffers et al. (2007), is a structured approach used in engineering and technology research to develop, implement, and evaluate an artifact that solves a specific problem. This structured approach ensures that the development of the DT system is goal-driven, iterative, and continuously improved based on testing and validation.

This methodology is designed to develop and evaluate an artifact - in this case, a real-time connected DT - to address the problem of enhancing monitoring, control, and predictive analysis of a UR10e robotic system in a battery assembly process.

The DSRM consists of six key steps; problem identification and motivation, solution objectives definition, design and development, demonstration, evaluation, and communication, as seen in Figure 4.1.

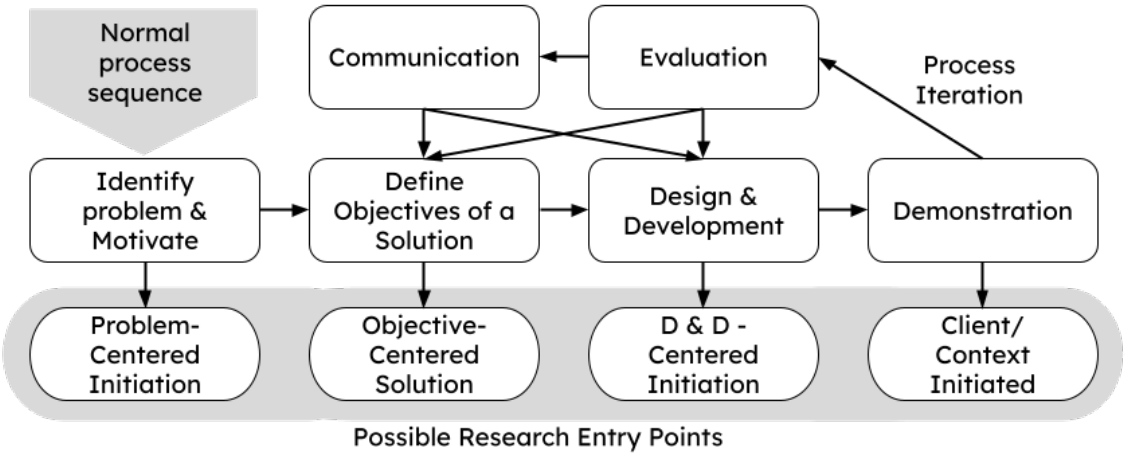


Figure 4.1: DSRM Process Model (Peffers et al., 2007)

4.1. Problem Identification and Motivation

This initial step involves recognizing and defining the core issue that the research aims to address (Peffers et al., 2007). It establishes the context, significance, and impact of the problem in the field of study. This is of high importance as it ensures that the research is relevant and solves a real-world problem, as well as helping justify the need for the proposed artifact.

The identified problem is the lack of real-time connectivity and predictive analytics in robotic assembly and disassembly processes. This limitation affects efficiency, monitoring, and control, needing a DT system for improved operations.

As the increasing complexity of modern manufacturing systems needs the implementation of real-time monitoring and predictive control. Robotic systems used in assembly lines often encounter issues such as unexpected interruptions in operation, inefficient programming processes, and the absence of seamless interaction between their physical components and virtual environments. A real-time connected DT provides a solution by mirroring the physical system and allowing for remote monitoring, data-driven decision-making, and predictive maintenance.

4.2. Solution Objectives Definition

This phase translates the problem statement into specific measurable objectives that the solution must achieve (Peffers et al., 2007). The objectives can be qualitative or quantitative. This provides a clear direction for design and implementation, helps to define the scope of the research and the criteria for evaluating its success, and ensures that the project remains goal-oriented and structured. The main objectives within this project include creating a DT architecture, ensuring real-time data exchange, developing a 3D simulation model, and integrating a UI for intuitive control.

4.3. Design and Development

This step involves the actual creation of the artifact, including the selection of tools, technologies, and frameworks that will be used to implement the solution (Peffers et al., 2007). It involves designing models, simulations, and software components. This translates the theoretical concepts into functional prototypes, defines the architecture, structure, and working mechanisms of the artifact, and ensures that all technical and operational aspects are considered before testing.

In this project, the DT system is developed using a software solution of choice, which for this project will be RoboDK. The architecture consists of a physical layer, a virtual model, data exchange, and user interface to ease the real-time monitoring and control.

4.4. Demonstration

This phase tests whether the developed artifact solves the problem under controlled conditions (Peffers et al., 2007). The artifact is implemented in a simulated or real-world environment, and its functionality is observed. This provides empirical evidence of the system's performance, it helps identify gaps or inefficiencies in the design that need refinement, and ensures that the solution is practical, functional, and aligned with the objectives. For this, the DT is connected to the physical UR10e robot, where the real-time data exchange, movement synchronization with the virtual model, and data analysis are tested separately to ensure that their functionalities work as intended.

4.5. Evaluation

This step measures the performance of the artifact, comparing the tests carried out in the demonstration phase against the defined objectives. It involves collecting quantitative and qualitative data, such as latency times, accuracy rates, and system robustness. This ensures that the solution is effective and meets the expectations, it also identifies any weakness or areas for improvement before its full deployment. Additionally, it provides scientific validation for the developed artifact. The evaluation of the DT will be based on the synchronization accuracy between the virtual and physical robot, the latency in data communication, the charts extracted from maintenance determination, and the user interface and control robustness.

4.6. Communication

The final step involves documenting and sharing the research findings through reports, academic papers, or industry presentations. It explains the problem, methodology, design, implementation, results, and conclusions. Allowing the border scientific and engineering community to learn from the research enables future improvements or adaptations in other projects, justifying the impact and relevance of the research in the field. Once the DT has been finished, the findings, including the implementation challenges, test results, and optimization strategies, are to be documented, and future research directions and other potential applications are to be discussed.

4.7. Theoretical Framework

The theoretical framework diagram visually maps the project objectives onto the stages of the DSRM (diagram 4.2). It illustrates the structured progression through the different phases, showing how each component of the DT is methodically planned and implemented within the research framework. Said theoretical framework diagram is displayed below.

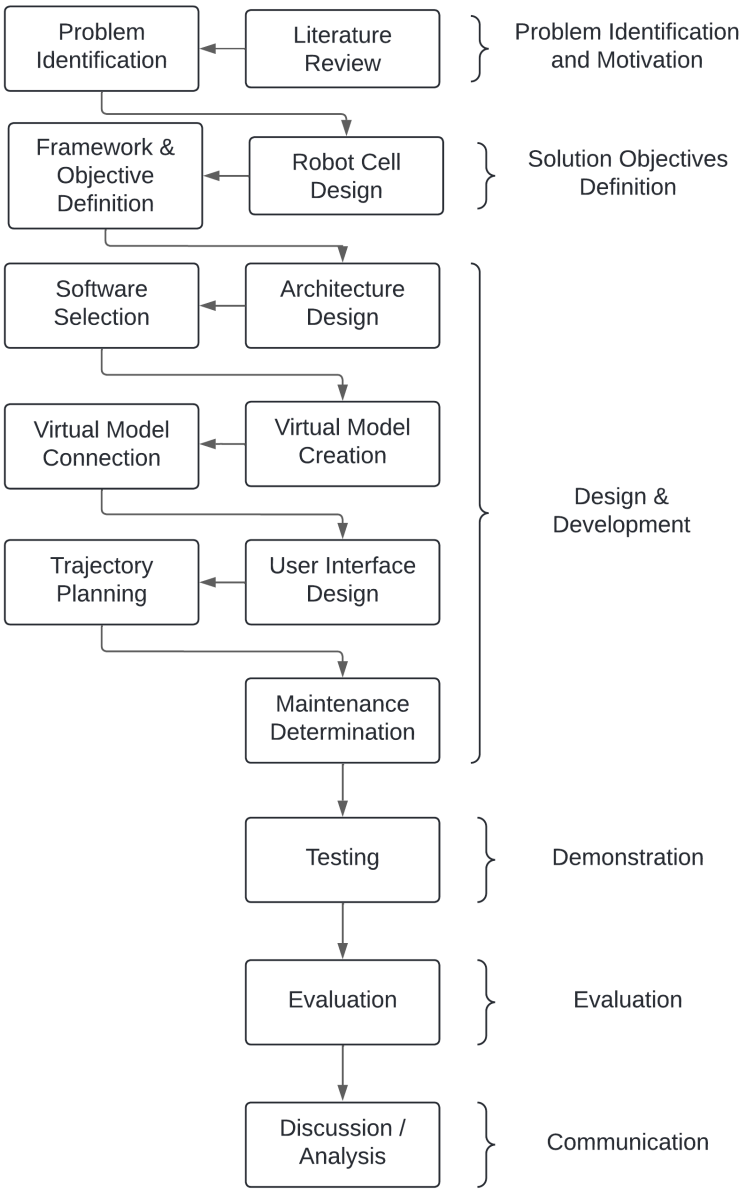


Figure 4.2: Theoretical framework diagram

5. Implementation

This section presents the practical realization of the DT system developed for the UR10e robotic assembly and disassembly process. It describes the step-by-step development workflow, including the selection and integration of key technologies such as the software solution and the UI. This section details how the components of the system (both physical and virtual) were designed and connected. By documenting the full implementation process, this chapter provides a clear account of how the theoretical objectives and architectural designs were translated into a functioning real-time DT. This implementation section embarks the design & development phase of the DSRM, and took place once the phases of problem identification and motivation, and solution objectives definition were considered finished. The DT will consist of a virtual model that mirrors the entirety of the physical robotic station, and will be governed by a UI.

5.1. Assembly and disassembly process

Before dwelling into the implementation of the DT system, an overview of the process for which the system is created must be done. The DT system aims to digitalize and control the process of assembly and disassembly of battery cells developed by Bolivar Perez and Sagarna Zabala (2025). Said process is constituted by a set of physical objects that must be arranged in a given position in a certain order. The assembly of battery cells regards the positioning of four cells in a closed box. The process of assembly requires the robot to take a box from the box shelf and position it in the assembly zone, then take four cells from the cell shelf and insert them into the four slots of the box. Once the box is full of cells, it is moved to the final product deposit shelf, where it waits for the lid. The lid is taken from the box shelf, positioned momentarily on the assembly zone to reposition the grip of the robot, and then placed on the box that has been moved to the final product deposit shelf. Once the lid is located on its box, the assembly process is finished. The disassembly process is composed by the same steps as the assembly process, but executed in reverse. The disassembly process finishes whenever the box, lid and cells are in their respective shelves, in the positions they were taken from. In the robotic station there are three shelves, (box, cell and final product deposit), an assembly zone, the UR10e robotic arm equipped with the RobotiQ 2F-85 gripper, and a webcam. In the box shelf there are three different boxes with their respective lids, grouped in pairs by level. In the cell shelf there are twelve cells positioned horizontally, having one cell situated in each level.

5.2. Initial Steps

The initial state of the project involved the organization and preparation of the approach to be taken, planification of the methods to achieve the goals, as well as the software selection and its justification. The first thing that had to be done was the literature review, investigating and comprehending the existing research on DTs and

related topics. Once the framework of the project was understood and correctly delimited, the proposed robotic station solution was analyzed, the individual aims and objectives were set, ensuring them to be reasonable and leaving space for some extra additional desired functionalities. This process constituted the phases of problem identification and motivation, and solution objectives definition of the DSRM.

Lastly, with the knowledge obtained from the literature review, the system architecture was devised, being the first step in the design and development phase. The initial architecture was designed with the simplicity of just the UR10e connected via Ethernet to a local computer, which would execute the required software solution. However, after discussing with both the supervisor and the examiner, and taking into account the other projects being carried out within the same robotic cell, a different architecture was designed. This architecture was composed of the UR10e, a local computer, a router, and a webcam. The UR10e would be connected via Ethernet to the router, to which the computer would be connected wirelessly to access the robot. This wireless connection enables any authorized user to connect to the robot, as well as the versatility of the positioning of the robot cell, as the computer monitoring the robot does not need to be physically connected to the robot.

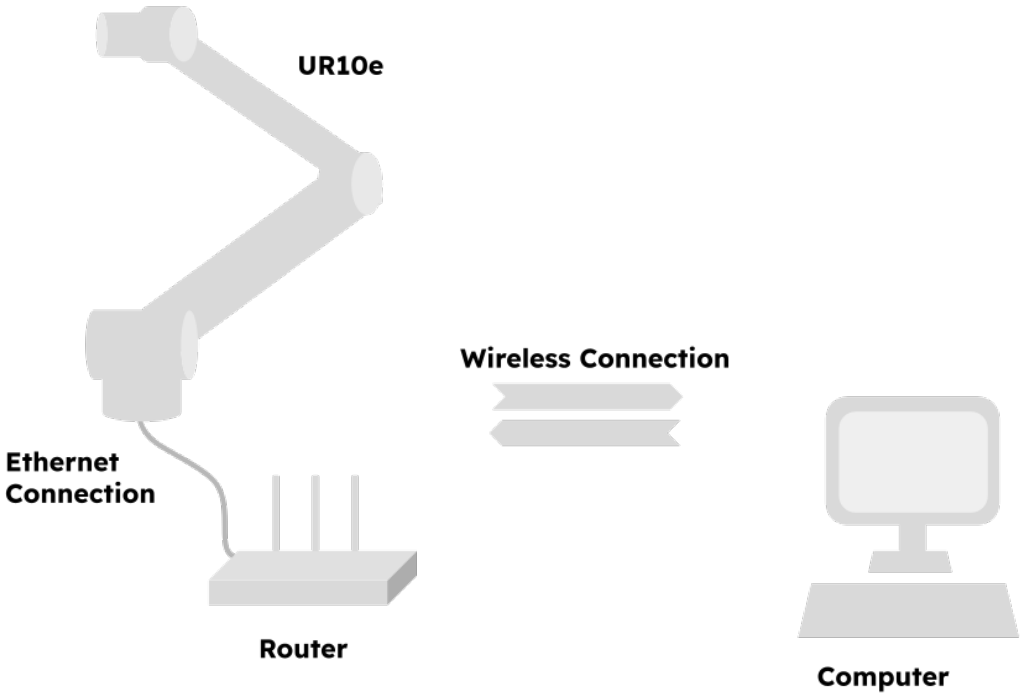


Figure 5.1: System architecture diagram

With all of the initial steps performed, the development of the DT itself could start.

5.3. Software Selection

In the following section of the report, a structured comparison of both RoboDK and Simumatik is provided, evaluating their suitability based on their ease of use, connectivity, implementation, data analytics potential and remote control capabilities. Said key factors must be considered to ensure a successful and efficient deployment, and each comparison plays a crucial role in determining the best tool for the implementation of the DT. The main objective here is to justify the selection of the most efficient and suitable software for said implementation. Other softwares such as Gazebo (via ROS), Tecnomatix, ThingWorx or Unity have also been investigated, but were quickly discarded due to incompatibilities with this project or for being overly complex when compared to RoboDK and Simumatik. Both softwares were evaluated through brief testing (due to a lack of license at the time), and through the study of the software documentation; RoboDK (2025c) and Simumatik (2025b). This section corresponds with the second step in the design & development phase.

The first characteristic to compare is the ease of use, this proves to be one of the initial considerations, as a software that is easy to learn and use allows for a faster implementation, reducing the time needed for training and troubleshooting. An intuitive environment and a simpler interface leads to fewer errors and enables a higher focus on the implementation itself rather than requiring expertise to manipulate the tools in an efficient manner.

Taking a look at RoboDK, it provides an intuitive, graphical interface that simplifies robot programming. It requires minimal configuration, enabling rapid deployment of simulations and real-time connections to the physical robot. Simumatik, on the other hand, while powerful for system-wide automation modeling, has a steeper learning curve. It requires a more detailed setup when compared to RoboDK, such as the robot controller itself. Therefore RoboDK is deemed to be more user-friendly and allows for a faster setup, making it the preferred choice for a robot-focused task implementation.

The second key factor to take into account is the connectivity with the physical robot. A DT must communicate in real-time with the physical robot to accurately mirror and represent its movements and operations. Therefore a poor or complicated connectivity could cause delays, mismatching in the models, or even system failures, which can then lead to inefficiencies or even safety risks. A direct connectivity ensures that the simulation is realistic and reliable, making it a valuable tool for testing and debugging before running and operating the robot in the live production environment.

Regarding connectivity, RoboDK offers built-in connectivity with UR through its native drivers and Python Application Programming Interface (API). It allows real-time communication, enabling direct control, monitoring, and program execution on the physical robot. Similarly, Simumatik also offers an immediate direct connection with the physical robot, but lacks simplicity in connection setup compared to RoboDK, as the controller in Simumatik must be configured for the connection to be successful. Both options offer a real-time direct connection, with RoboDK having a slightly more seamless connection.

The third characteristic that has to be taken into account is the implementation

of the DT itself. The main purpose of the DT is to accurately replicate the physical robot and allow for simulation and analysis, therefore a well-integrated DT reduces the need for physical prototyping, reducing the costs and saving time, and plus, if the simulation does not match the physical robot's behaviour accurately, the entire system would be unreliable. Additionally, included tools and add-ons greatly aid in the implementation of the DT, enabling the focus to be set on the optimization itself.

For the implementation of the DT itself, since RoboDK specializes in robotic DTs, it provides a realistic simulation environment with accurate kinematic modeling and collision detection, where the virtual model can be directly synchronized with the physical robot for real-time feedback. Additionally, RoboDK has many tools and add-ons that can aid in both the remote monitorization and the data analytics. On the other hand, Simumatik is designed for system-wide automation simulations, it allows the modelling of conveyors, sensors and other control logic units, but limits the capabilities of robot simulation, requiring external integrations such as add-ons and tools. Therefore RoboDK is deemed best in this aspect.

The fourth key factor to take into account is the data analytics and monitoring capabilities of the programs. Collecting and analyzing the data from the physical robot allows for joint data processing for the maintenance determination, which helps prevent failures and improve efficiency. With good data analytics, it enables the detection of performance issues before they lead to possibly costly downtimes. Additionally, monitoring real-time performance helps optimize cycle times, energy consumption, and other overall productivity factors.

Taking into account that the UR10e has a built-in data logging tool to keep track of data such as the current consumption, the main capability of the program has to be the extraction of said data and the analysis of the data itself to make a decision. RoboDK includes built-in data tracking tools, enabling even more data to be taken in such as cycle time. RoboDK also supports Python and MATLAB integration, which in turn allows for customized data visualization and advanced analytics. On the other hand, Simumatik does not offer a native robot monitoring tool, and therefore extracting real-time data from the physical robot would require external logging methods and additional software configuration, making the process more complex. That is why RoboDK has also been selected as superior when regarding this key factor.

Finally, the last characteristic to be considered for the selection of the software is the capability and ease of remote control and real-time feedback. Being able to control the robot remotely allows for a greater flexibility, making it possible to monitor and adjust operations without physically being present. They are useful in troubleshooting and updating programs, as well as being essential for validating performance and making on-the-fly adjustments.

Both RoboDK and Simumatik support remote operation of the physical robot from the locally connected computer where they are being run on. Local network communication enables the physical robot and the program to establish a two-way connection. However, RoboDK counts with additional tools and add-ons that can aid in the connection, making it simpler and more intuitive. Therefore RoboDK is also recommended as per this key factor.

In this comparative analysis, RoboDK and Simumatik have been evaluated based on several key factors and desired characteristics. After a thorough comparison, RoboDK has been selected as the optimal solution for this project. Its intuitive interface, direct robot connectivity, built-in analytics, and remote control capabilities make it the most efficient and practical choice for developing the DT. While Simumatik offers a broader system-wide simulation environment, it introduces unnecessary complexity for a project that primarily focuses on robotic monitoring and control.

By choosing RoboDK, a faster implementation, an accurate real-time synchronization, and the ability to analyze and optimize the robot's performance efficiently is ensured. Thus RoboDK is the most suitable software from those considered for this digital twin project.

5.4. RoboDK

The next step, the third one in the design & development phase, was the creation of the entire virtual space of the DT. For a fully functional development of the robotic station, two key aspects of RoboDK software were taken into account. These were the creation and/or import of objects as well as the cell itself, and the use of functions that allow to simulate a realistic and smooth behavior of the gripper used, which in this case was the RobotiQ 2F-85 gripper.

5.4.1 Virtual objects

Shelves

The designs of the three shelves in the cell, which are the box and cells shelf, the assembly zone, and the final product deposit shelf, were based on their real-life measurements. To model them in 3D, two aspects were considered: the objects designed and 3D printed by Bolivar Perez and Sagarna Zabala (2025), which were inserted into the station, and the objects that were purchased, which were modeled using SolidWorks. After this, each of the necessary pieces to build the physical shelves were imported into RoboDK and modeled directly within RoboDK, as shown in Figures 5.2 and 5.3.

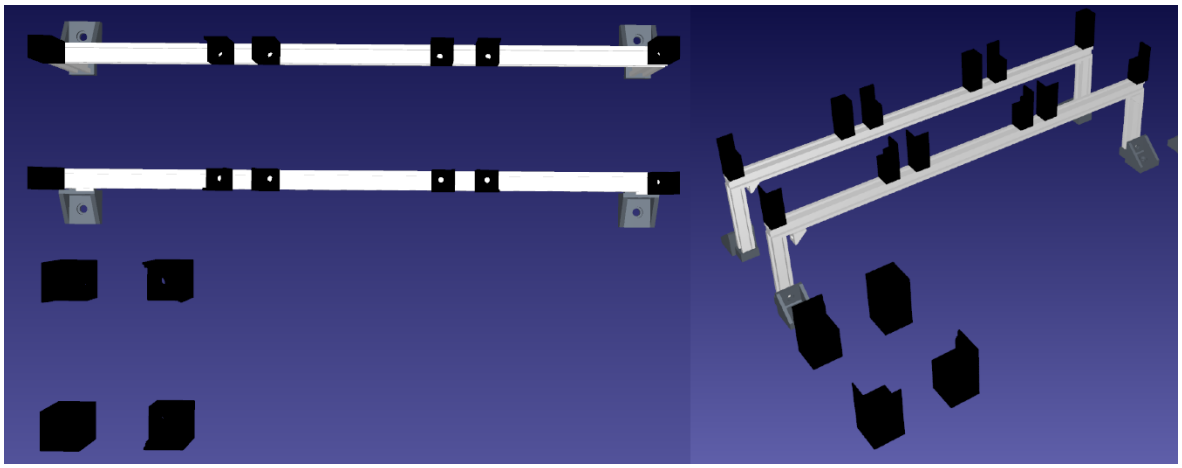


Figure 5.2: Assembly zone and final product deposit shelf

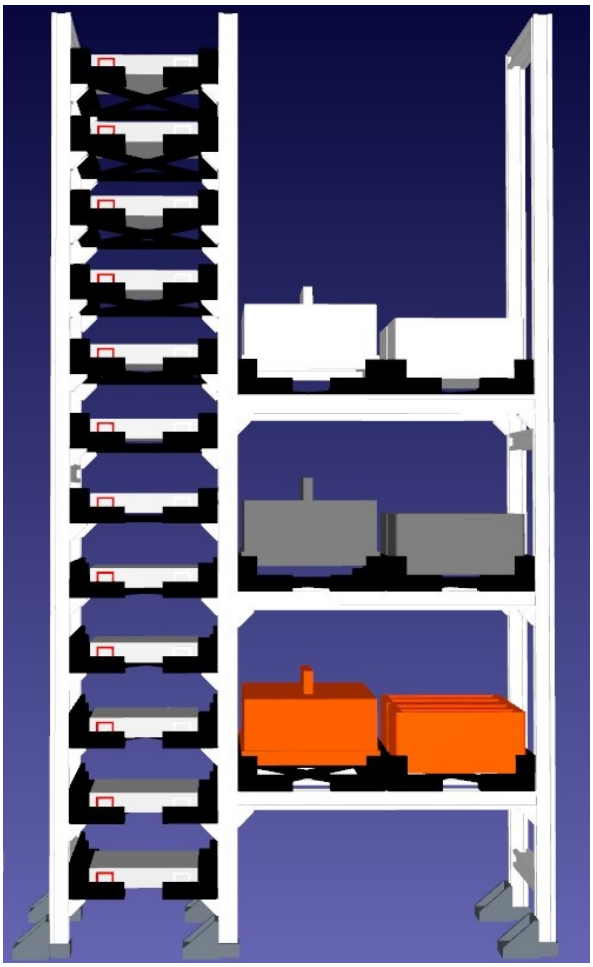


Figure 5.3: Boxes and cells shelf in RoboDK

Building the virtual objects as the union of all the individual pieces of the shelves ensures modularity and enables any necessary adjustments to be easily made. The dimensions of the different shelves in the station are shown in Figures 5.4, 5.5 and 5.6.

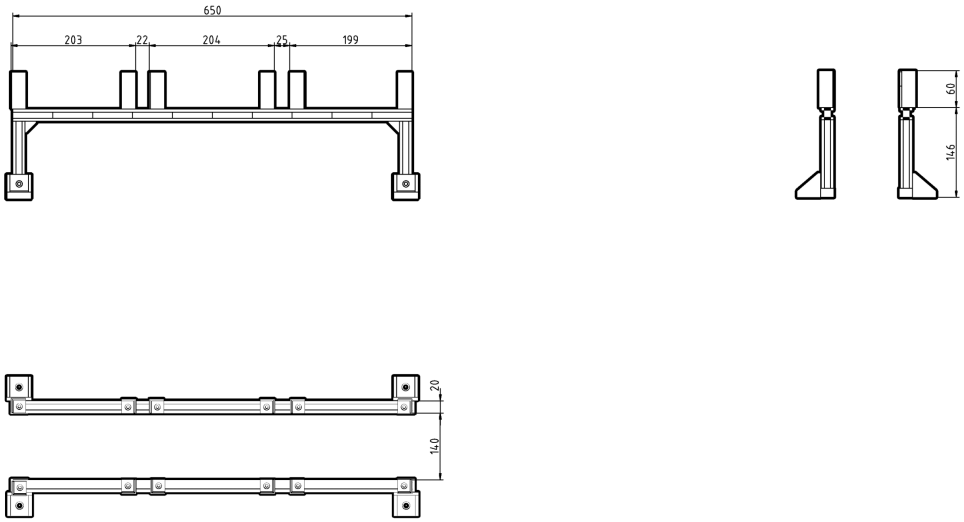


Figure 5.4: Final product deposit shelf dimensions

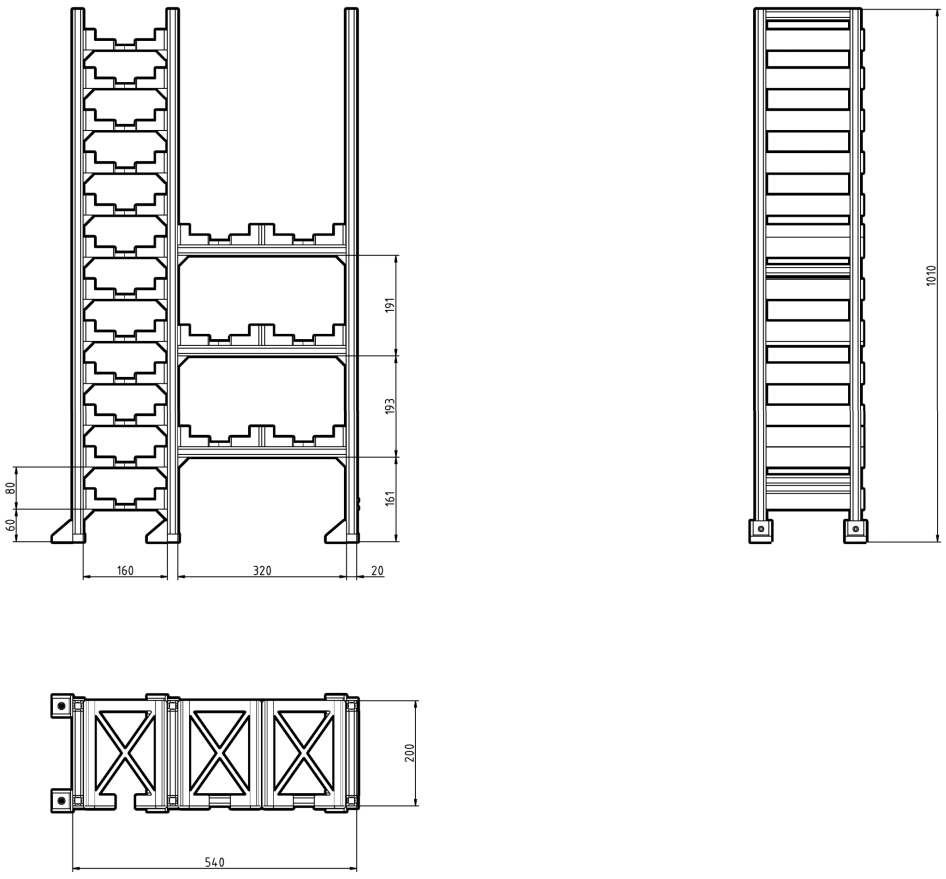


Figure 5.5: Cells shelf (left) and Boxes shelf (right) dimensions

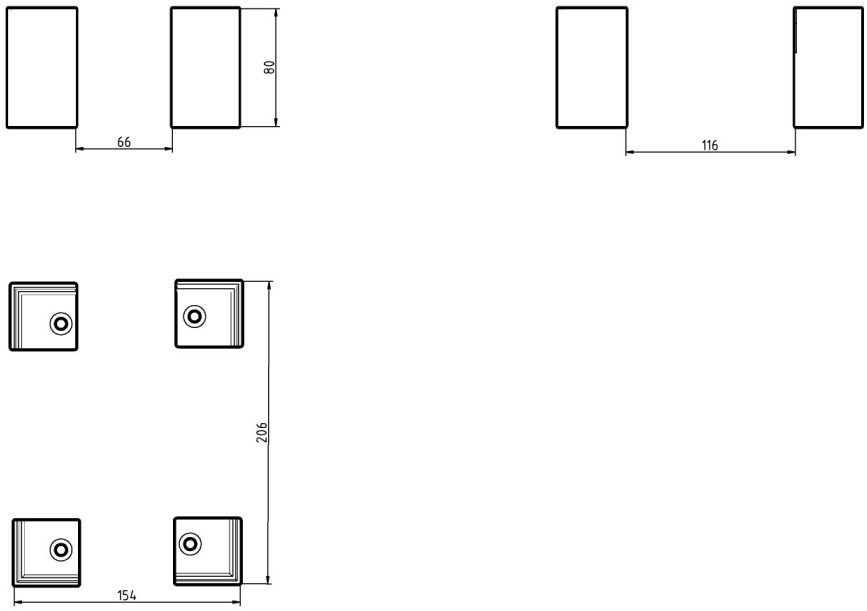


Figure 5.6: Assembly zone dimensions

Robotic Station

The robotic station's main structure was purchased from the company Vention and assembled. The virtual model of the main structure was already available when the project was initialized and therefore was of great use. Said model can be seen in Figure 5.7.

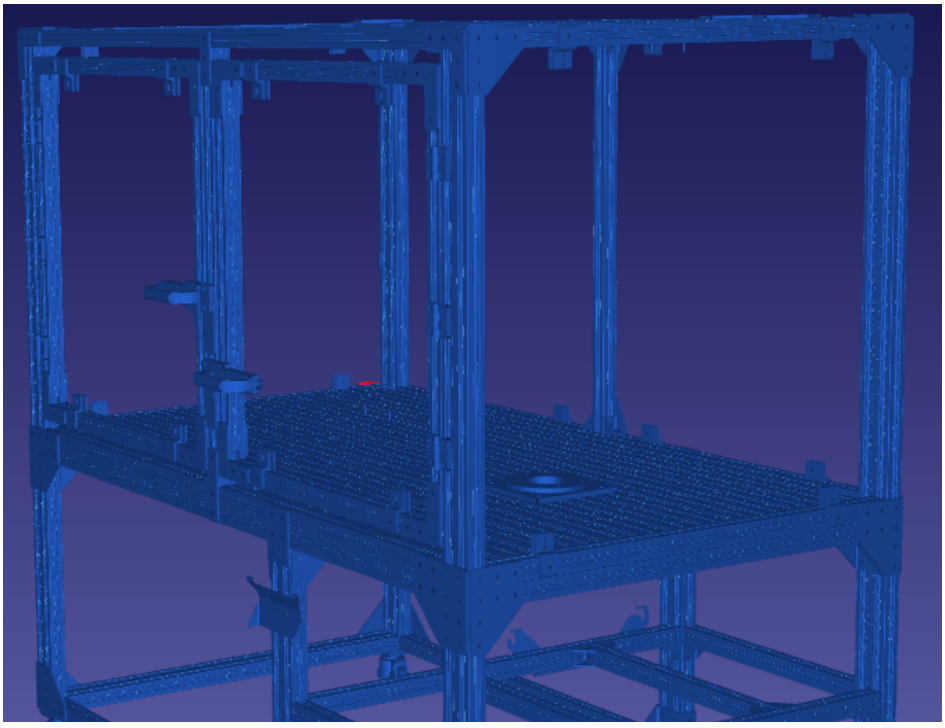


Figure 5.7: Empty station

However, Rhinoceros 3D was used to modify the structure model to ensure compatibility with this project. The objects within the station were separated into blocks with the goal of creating a modular station that could be easily modified if the robot's position needed to be changed. Due to this, the station was separated into two main sections: the full station on one hand, and the robot base on the other, in case it needed to be moved. This allows repositioning directly from RoboDK. The Figures 5.8 and 5.9 show how the actual robotic station looks with all the objects contained in it, while Figure 5.10 shows the position of the various objects present in the station.

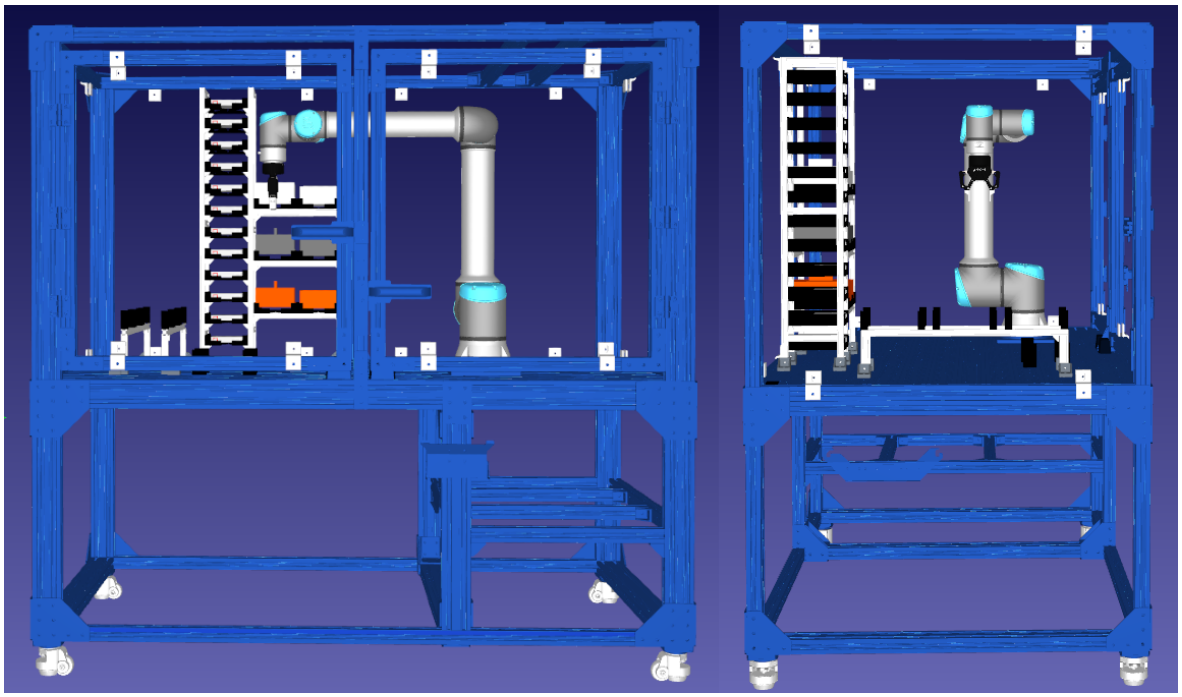


Figure 5.8: Elevation and side view of the robotic station

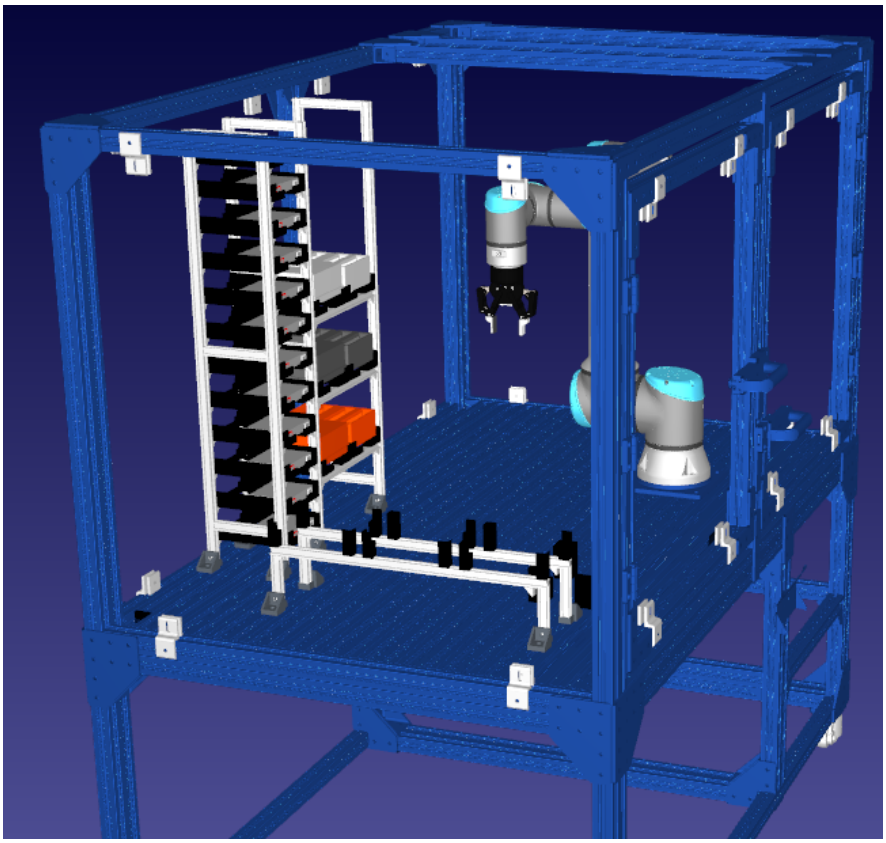


Figure 5.9: Angular view of the robotic station

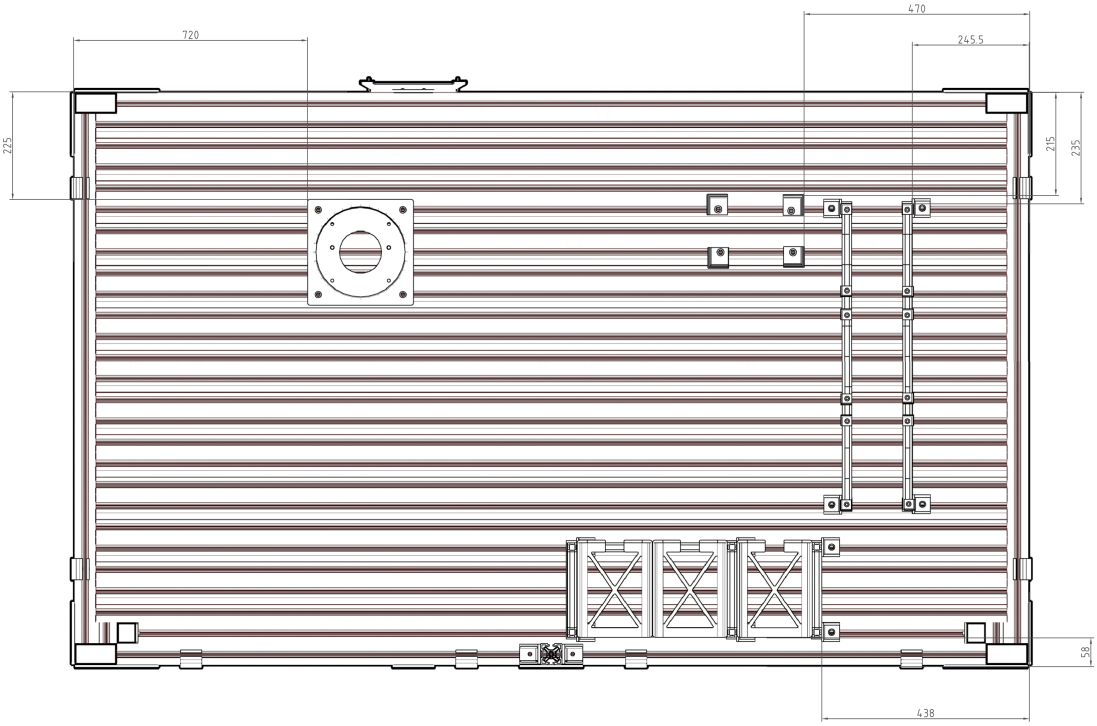


Figure 5.10: Top view of the station

Boxes and Cells

The design of the boxes, lids and cells were obtained from the work of Bolivar Perez and Sagarna Zabala (2025), due to their work being the physical implementation of the robotic station, the files to be 3D printed and used in the physical environment could be also used for this virtual model. For this reason, modeling them in RoboDK only required importing and joining them.

On the other hand, the cell dimensions were taken and modeled using SolidWorks.

5.4.2 Functions

Having created the virtual model, auxiliary functions are needed to ensure desired functionalities. Scripts such as the one needed to perform the attachment of the virtual objects to the robot TCP in order to move them in the virtual model, are implemented and included in the RoboDK object tree. The functions, which can be seen below, are located in the RoboDK environment but are called from within the User Interface (UI) program, executing them as explained in Appendix A.5.4.

- Attach: This function allows the virtual robot to grab a virtual object and then manipulate it by enabling its displacement. To do this, a single object is attached to the robot tool TCP, this being the gripper. One function per object that the robot can grab has been implemented.
- Detach: This function allows the robot to release the objects it had previously grabbed. This function in RoboDK releases the object, or objects, that the selected TCP tool is holding and attaches them to the global reference frame of the robot station. This function attaches all of the previously attached objects on the TCP, to the global reference, therefore only one *'Detach'* function is needed for the entirety of the objects in the station.
- Replace: This function is intended to be used once the final object has been correctly assembled and is to be removed from the cell. When this happens, the objects used in the process return to their initial position, and the process continues without interruptions.
- Replace all: This final function is not used during the execution of the path, but rather implemented as a useful tool if a complete reset of the cell is desired. This function returns all objects to their initial position. It is intended to be used when performing tests and then restoring everything to its initial state. This occurs during the initialization of the UI to reset the virtual environment and place all objects in their defined starting positions, or if an unexpected power outage occurs and a reset is needed to restart the station.

5.5. Preconnection

Once the virtual model is created, the next step is to install everything necessary to create and run the DT without issues. This would compose the fourth step in the design & development phase. After installing the RoboDK software and Visual Studio Code in case it has not been previously installed, several considerations must be taken

into account to establish a successful connection between the virtual model in RoboDK and the real robot. All of the necessary files are found in the digital twin folder, which contains the robotic station, UI Python files, gripper functions, reset cell image and joint Comma-Separated Values (CSV) files.

To enable a connection between RoboDK and the physical robot, an exception in the Windows Defender Firewall must be created in order to allow said connection, otherwise remote control of the robot will not be allowed. For this, both an inbound and an outbound rule must be created, these rules can be set up as program exceptions for RoboDk and its driver "apiur", or port exceptions for ports 30000-30002, which are the range of ports in which RoboDK usually operates with UR devices.

After opening the robotic station RoboDK file, which contains the complete robotic station with the virtual objects set in their initial positions, and apart from ensuring to have an active and valid license, three things must be ensured before executing and starting the DT:

- The robot in the station, in this case the UR10e, is using the *Universal Robots RobotiQ* post-processor. To check this, the UR10e robot must be located in the tree, and when selecting the option *'Select post-processor'*, said one should already be selected for the RobotiQ. This is necessary for RoboDK to know which tool the robot is using, allowing it to process the related commands.
- The UI is opened and run in Visual Studio Code. It must be ensured that the libraries and extensions mentioned in Appendix A.2 are installed for a correct code execution and to improve code comprehension and writing, respectively.
- The physical robot must be set to remote control mode from the teach pendant, allowing an external connection between the physical and virtual robot to be established.

The connection between the virtual and physical robots is performed by establishing the robot's IP address (retrieved from the teach pendant) and the port `30002` in the *'connect to robot'* window of RoboDK. It must be assured that the architecture devised in 5.1 is applied correctly for the connection to work.

5.6. Gripper RobotiQ 2F-85

With the robot and the tool imported, a way to control the gripper movement remotely had to be devised. In order to achieve control of the tool with the Universal Robot, several tests with different methods had to be carried out until full control of the Robotiq 2F-85 gripper was achieved. For this, the RoboDK (2025b) website provides two methods that can be used to achieve external control of a tool and its robot. With one of these methods called *'Send Program to Robot'*, it was possible to move the gripper externally, as will be further explored in the first bullet written below. However, full functionality of the tool was not achieved as intended, and the program provided by the RoboDK software had to be adapted, as explained in the second bullet, also mentioned below.

- Send Program to Robot: The RobotiQ 2F-85 gripper, being a RobotiQ tool,

RoboDK includes its own mechanism as well as a specialized post-processor for RobotiQ devices. With the use of this post-processor, it is possible to generate program calls that can operate the RobotiQ gripper (RoboDK, 2025b). Doing this generates a program that moves the gripper. However, even though sending the program to the robot executes the action correctly, it instantly disconnects the robot, so it was necessary to manually reconnect the virtual model to the physical robot every time the user desires to operate the gripper, something far from optimal. Said reconnection could not be done automatically from the UI code.

- Socket: Due to the near-success achieved by sending the program with the option generated by RoboDK, an attempt was made to establish communication through socket. With the use of a socket connection, the UI can connect itself to the physical robot and, once the programs generated in RoboDK are extracted, it is possible to send the scripts generated from these programs to the physical robot to operate the gripper. After moving the gripper, an automated reconnection to the RoboDK robot can be programmed from the UI. To carry out this method of gripper control correctly, two things must be considered: the setup that extracts the programs from RoboDK to make them functional, and the different necessary functions that perform the gripper movements.

It must be noted that since the implemented tool has its own movement, in RoboDK it is treated as a mechanism and therefore considered a robot. For this reason, the commands that move the physical and virtual gripper are not the same. While the virtual gripper is moved using a move joint function (`MoveJ()`), the command sent via socket to move the physical gripper is `rq_move()`, as shown in the RoboDK (2015-2025) API.

Therefore, in order for the gripper to be controlled, the programs created in RoboDK that make the gripper move in the physical environment are extracted and read. If the programs do not exist locally, they are extracted from RoboDK and saved in the local digital twin folder. It then ensures that the programs have been opened and saved correctly, and then the content of the programs are read and stored as commands to be used later. All of this process is performed within the execution of the code. Due to cross dependencies with other projects, and the collaborative nature of this project with the one performed by Bolivar Perez and Sagarna Zabala (2025), both the virtual and physical gripper control code has been encapsulated in a separate file `gripper.py` to allow a distinctive use of the gripper control within the project work of Bolivar Perez and Sagarna Zabala (2025) This can be seen in Appendix A.5.

5.7. User Interface

The fifth step in the design & development phase is the UI design. For this, the entirety of the implementation and programming of the robot and its functionalities, including the advanced computation and the UI have been developed on top of a previously existing python file. Said file is found in the work of Bolivar Perez and Sagarna Zabala (2025), a python program containing three classes; `Vision`, in charge of the visual identification of battery cells, lids and boxes for the assembly through a vision system,

`Path`, containing the program logic, robot path and target order of the assembly and disassembly process, and finally `Robot` containing the instances of the previous classes and the one in charge of connecting to the robot. Therefore in this project, the aim of the UI is to further develop this existing file by importing it to a new file and adding the desired functionalities and methods as additional classes, and substituting the `Robot` class for a much advanced, intuitive and user-friendly `UI` class.

To add simplicity and modularity, all of the execution of this DT system has been done in a single python file. Due to different contextual sections of the logical flow of the system having to be running at the same time and requiring real-time communication, (for example having the vision system or the robot movement running at the same time as the UI) two options were considered. The first option was to have each contextual section in a different python file, (`vision.py, maintenance.py, paths.py, ui.py`, etc), then executing each one simultaneously through multiprocessing and communicating each process through tunneling. This would separate each section completely, and would cause the need to use consumption-heavy methods for each section such as multiprocessing and tunneling. Having such a relatively high amount of sections to be executed parallely provokes a high usage of the hardware, and would hinder the communication between the DT and the physical robot, possibly slowing the connection and reducing accuracy of the virtual model.

The second option, on the other hand, was to include every section in the same file. By importing the previous code developed by Bolivar Perez and Sagarna Zabala (2025), modifications can be performed and new functionalities can be added without manipulating the original file. Three files exist in the DT system: the `gripper.py` file that contains the code and functionalities to connect and control both the physical and virtual grippers, `robotcode.py` developed by Bolivar Perez and Sagarna Zabala (2025), that contains the vision system and the assembly and disassembly logic processes, this file also imports and uses the `gripper.py` file created in this project to control the grippers, finally the `ui.py` file that contains the entirety of the UI, this file imports the `robotcode.py` file in order to modify some of its class methods and add additional classes and functionalities.

Having each section represented as a class, creating instances of each class and through inheritance, communication between the sections can be achieved seamlessly. For this second option, multithreading was considered as a lighter alternative to multiprocessing. Five contextual sections were devised, each being represented as a class: `RobotControlUI` (in `ui.py`) being the class that embarks the entirety of the UI and its control methods, `Path` (in `ui.py` imported from `robotcode.py`) being the class that includes all of the predesigned robot programs and trajectory calculation methods, `Maintenance` (in `ui.py`) being the class that reads and processes the joint data to determine each individual joint usage and processes that data to visualize it accordingly, `Gripper` (in `gripper.py`) being the class in charge of both the physical and virtual gripper control, as well as the attach and detach of the virtual objects, and finally `Vision` (in `robotcode.py`) being the class where the identification and comprehension of the physical environment takes place. The interaction between classes can be visualized by using the Unified Modeling Language (UML) diagram in A.5.

To visually control the whole system, a HMI is needed. The HMI provides both

remote control and a visual comprehension of what the control capabilities enable. Therefore, a UI was implemented. It was done as an external Python script outside RoboDK using the PyQt6 framework. The goal of the interface is to provide intuitive and versatile control over the UR10e robot and the industrial assembly and disassembly environment, both for real-time manual movements and the execution of optimized trajectories and robot programs. The UI integrates real-time robot communication, dynamic path generation, and advanced optimization algorithms via CasADi to enable a robust and extensible system for both teaching and industrial use cases. The implementation process evolved through multiple iterative improvements, each based on functional testing, user interaction, and system performance evaluations. The finished UI exists as a python file named `ui.py` and is located within the DT folder.

5.7.1 UI system design

The UI is structured around a central PyQt6 application, which is encapsulated within a `RobotControlUI` class, which is responsible for initializing and managing all widgets, timers, connections, and control logic. This ensures modularity, responsiveness, and real-time interaction. RoboDK's Python API (robolink and robomath modules) was used for all robot control functions, including retrieving robot items, setting pose frames and tools, moving along trajectories, and interfacing with the physical robot. A complementary `Path` class handles path planning, interpolation, and execution, including interaction with the CasADi optimization framework. Three additional classes also handle the vision system, maintenance, and gripper functions respectively. The separation of UI logic, path execution, vision system, gripper functions, and maintenance handling facilitates maintainability and scalability of the system.

Upon initialization, the UI opens the RoboDK application and connects the RoboDK file containing the virtual station and a user prompt appears via `ItemUserPick()` to allow manual selection of the UR10e robot model. An emergent window containing an image of the initial reset state of the virtual cell prompts the user to ensure that the physical cell is in the same starting condition as the virtual model, this ensures that the starting point of the process, as well as the synchronization of the virtual and physical objects are always correct, even after unexpected situations such as power outages. It then verifies the availability of the DT and attempts to connect to both the physical robot and the gripper over a local network using an IP address. In case either one is unavailable, the pertinent error messages are displayed, if the connection fails, the user is prompted to retry the connection through an emergent window. This connection procedure was designed for failure handling and improved to ensure robust startup behavior.

Once connected to the robot, the UI appears as a windowed application. Two buttons on the top enable switching between manual and automatic mode, two sets of six directional buttons enable the manual movement of the robot TCP, a home button that will move the robot to its home position, three buttons give the user control over the predefined robot program, enabling the user to execute, pause, and resume the program, five situational buttons enable a manual control of the gripper, two maintenance buttons to view and reset the joint usage data, and finally a slider to manually select the speed of the robot.

5.8. Advanced Computation

As stated in the objectives, two main advanced computation additions were desired to be implemented, each corresponding to one of the final stages of the design & development phase. On one hand the trajectory refining, which would enable the user to choose between a selection of algorithmical methods from which to calculate the trajectory the path would take. And on the other hand the maintenance determination, which would enable the user to visually comprehend the usage of each robot joint and determine when to carry out a maintenance. Both of these additions were developed and implemented in the `ui.py` file.

Trajectory refining was implemented as additional functions into the existing `Path` class, whilst maintenance was developed as a whole separate and independent class `Maintenance`

5.8.1 Trajectory Refining

Trajectory refining, or trajectory planning, determines the path that the TCP will take in order to reach a given robot joint position (kept as a target) from another starting target. This is done automatically by the `MoveJ` or `MoveL` functions, however, it is limited to a single movement regarding cartesian axis (for `MoveL`) or joint movements (for `MoveJ`) and does not take into account further optimizations such as obstacle avoidance, time minimization or energy consumption. Therefore, three different methods of calculating the trajectory (or path) have been implemented as options. The selection of the method of calculation resides in a function `perform_path` in the `Path` class.

The first method of trajectory calculation is the predetermined `MoveJ` function, as simplicity is often preferred, the basic function has been left as the predetermined method of calculation. The second method of trajectory calculation is through interpolation, where the path to follow between the origin target and the destiny target is separated into a set number of individual smaller paths that are then concatenated and executed, this does not provide the trajectory with obstacle avoidance or any parameter minimization, but makes the path carry out in smaller increments, enabling a steadier approach.

Lastly, the third method of trajectory calculation involves the use of CasADi, here the trajectory is calculated through a solver; a limit to the maximum translation per step is set for smoothness, then the Non-Linear Programming (NLP) problem is built, an initial guess is performed and finally the optimization is solved and the trajectory then extracted and executed iteratively from the solution. Additional features can be selected to be used; obstacle avoidance is carried out by ensuring each point of the trajectory remains at least a minimum distance away from the obstacle center via distance constraints in the optimization problem. Time minimization is carried out by including a symbolic variable to minimize total time, adding a lower bound to ensure the motion is not instantaneous. Finally, energy minimization is carried out by penalizing acceleration through a cost function that minimizes the second-order difference between adjacent steps, providing a smoother movement. These features can be enabled by adding their respective parameters, or can be left blank to perform a simple trajectory calculation.

Being CasADi the most complex method of trajectory calculation, it offers incomparable features; however, the execution time hinders the whole robotic station, as each time the script is executed, CasADi takes a few seconds to carry out the calculations before performing the whole script.

5.8.2 Maintenance

In order to try to determine when maintenance should be carried out, the user must know the wear and use of the joints. This can be induced and determined by the average acceleration, velocity and use of each joint. For this, a tracking of the different joints has to be made for the user to comprehend the usage of each individual joint and identify potential anomalies in the joint usages. To obtain said information, each robot movement has been appended to a script, from which the joint data has been extracted to a CSV file with `InstructionListJoints`. The data extracted through this method provides the positions, steps, speeds, accelerations and times of each joint throughout the execution of the script. Once the data is extracted, graphs that represent the range, usage, position over time, speed and acceleration for each joint in the execution of the current script are created and shown. This enables the user to easily visualize the data and identify possible anomalies amongst the joints, as well as identifying which joints are being used the most.

Additionally, the average usage, speed and acceleration for each joint is saved to a separate CSV file in order to have a historical log of how this data changes throughout time. This data is visualized alongside the other joint graphs. The data in the historical log are updated with each execution of the script and saved between instances of the UI.

Having access to all of this data enables the user to identify any anomalies in joint usage, as well as giving insight on which joints are being used the most and therefore will most likely be the first ones to fail or require maintenance.

5.9. Evaluation

To evaluate the UI and the general DT system, latency and positional accuracy will be measured between the virtual and physical robots throughout the execution of the assembly and disassembly processes. Positional accuracy will be evaluated by stopping the robot in ten different positions along the execution of the assembly and disassembly processes. The physical robot will then be disconnected and the joint values of the physical and virtual robots will be noted and compared using standard deviation. Latency in the DT system will be evaluated by measuring the time between the sending of a command and the detection of the command execution for both the robot and the gripper movement. In other words, for the robot movement, a timer is started when the UI sends the `MoveJ`, and it is stopped whenever the UI detects that the robot has started to move. For the gripper movement, a timer is started whenever the movement command is called, and two times are measured: when the virtual gripper responds, and when the physical gripper finishes its movement and the robot reconnects successfully. 20 latency measurements will be taken for the robot movement, whilst 10

will be taken for the gripper. All of the time-related measurements are taken using the `perf_counter()` function from the `time` python library.

Additionally, tests upon the safety measures implemented will also be carried out to ensure that the safety measures work as intended. Trajectory refining will be evaluated by establishing a set test path to perform with the different optimizations and execution time will be measured for ten iterations of each optimization. Additionally, obstacle avoidance will be evaluated by locating a virtual obstacle in the form of a target to avoid, and comparing the path carried out by the TCP with and without the obstacle avoidance for the test path.

Maintenance will be evaluated by comparing the graphs created by the implemented functions, to the graphs created by the RoboDK add-in RoboCharts (2024) for the test path. The historical graphs will be evaluated by executing the test path ten times and ensuring their linearity. The interactive components of UI will be evaluated by ensuring they respond accordingly to their contextual situation. The evaluation performed on the interactive components will be the following:

- **Manual and automatic mode buttons:** They must switch the mode when they are pressed: the manual button must deactivate automatic mode and activate manual mode, whilst the automatic button must deactivate manual mode and activate automatic mode. The manual control buttons must not be responsive once the automatic mode button is pressed, and vice versa. The automatic mode button must be responsive whenever it is pressed, however, the manual mode button must only activate the manual mode whenever the robot is not moving or executing the program.
- **Manual movement and rotation buttons:** They must only be responsive when manual mode is activated and the robot is not moving. One single press of the button must displace/rotate the TCP by the step established in the code. Keeping the button pressed must produce a constant displacement/rotation of the TCP in steps whilst the button remains pressed. Pressing a button whilst the robot is moving must not perform the intended move, and must return a warning message regarding the busy status of the robot.
- **Maintenance buttons:** The maintenance buttons must only be responsive during automatic mode. The button labeled *Charts* must open and visualize the developed charts and graphs calculated by the maintenance determination in 5.8.2, whilst the button labeled *Clr Charts* must delete the history script in RoboDk and create a new, empty one. Pressing *Charts* after pressing *Clr Charts* with no robot movement between said presses must open and visualize the charts, but empty (with exception of the history graphs that must remain with the previous saved data).
- **Manual speed slider:** The slider must only be responsive during manual mode. Moving the slider must result in a change in robot speed only in the manual mode.
- **Manual gripper movement buttons:** The manual gripper movement buttons must only be responsive during manual mode, and when neither the robot nor the gripper are moving. Pressing on one of the buttons must result in the positioning of the gripper indicated by the button label, in a range of a few seconds.

- **Move to Home button:** This button must only be responsive during the manual mode, and whilst the robot is not moving. When pressed, the robot must return to its home position.
- **Execute Program button:** This button must only be responsive during automatic mode, and whilst the robot is not moving or already executing the program. Pressing the button must trigger the assembly mode selection and its subsequent emergent windows.
- **Pause Program button:** This button must only be responsive during automatic mode, and whilst the robot is executing the script. Once pressed, the robot must stop at the end of its current movement path, and not immediately, ensuring a safe stop.
- **Resume Program button:** This button must only be responsive during automatic mode and only after the *Pause Program* button has been pressed. Once pressed, this button must resume the execution of the current robot program, continuing with the assembly or disassembly process previously selected.

6. Results

This chapter presents a comprehensive analysis of the developed system for connected DT control in robotic battery assembly and disassembly, as well as an overview of the implemented functionalities, tying with both the demonstration and the evaluation phase of the thesis. Additionally, an evaluation of the performance and integration of each component within the broader architecture will be presented, ensuring that the system meets its design goals in terms of functionality, responsiveness, automation, and safety.

The project combines various subsystems: a simulation and execution backend using RoboDK, a communication framework and architecture through RoboDK, and a UI. Each component was evaluated independently and then tested in an integrated environment to verify interoperability. Testing scenarios included both simulated execution and real-time robot control with the UR10e robotic arm, under varying conditions such as manual operation, automated sequences, and obstacle avoidance.

The following sections present the results in three stages: first, an overview of the functionalities, then an overall system-level assessment, and finally, a more detailed analysis of each individual component, including manual control, program execution, gripper and speed interface, trajectory optimization, and DT synchronization.

6.1. Functionalities

Although an overview of the functionalities can be found here, the logic of the code, including a breakdown of each method of each class, along with its functionality and how they interact with each other can be found in the Appendix A.5.

- **Manual and automatic modes:** A central part of the UI is its dual-mode functionality, it was designed this way to ensure safety, as there are two different contextual types of control, a manual control where the robot is moved manually, and an automatic mode where a predefined script is set to execute on the robot. It would be a safety risk to enable manual control whilst executing a predefined script, as the robot could perform two different movements at the same time, making it unpredictable and unsafe. Therefore the implementation of two distinguished and exclusive modes was implemented, ensuring that only one can be active at a time, disabling the functionalities related to one mode when the other is selected.

  On one hand, the manual mode enables the user to send the robot to a home position, change the manual speed using a slider, and control the gripper and the robot's TCP position and orientation using intuitive directional buttons. The movement of both the physical and virtual robot occurs in small increments (set to a default of 10 mm or 3°), updated continuously through `QTimer` signals. The timers allow motion commands to be triggered at intervals, enabling smooth movement. On the other hand, automatic mode disables direct control inputs

and allows execution of predefined paths or programs, enabling the execution, pause and resume of the main robot program.

- **Script execution:** The UI can trigger the execution of a multi-step complex robotic path, defined in the `Path` class. Instead of relying on RoboDK's internal run mode, the function `script_execution` that iterates the robot movement through a list of targets was modified to enable a pause and resume of the robot movement between targets. This ensures full control of the movement and the ability to pause and resume execution. The class was also modified to include different methods of trajectory calculation, enabling the selection of the method of calculation only in the code, and not by the user. More on the trajectory refining can be found in the section 5.8.1.

  When the script execution is called through the UI button, an emergent window will prompt the user with the selection of the mode of execution of the script. The user can choose between three execution modes: automatic assembly, manual assembly, and disassembly. Once selected the method of execution, the `Path` method `main_script` is called as a thread to maintain the UI responsiveness, and the robot carries out a full assembly of the selection. The order of the path to be followed, as well as the targets, logical flow and iteration of the paths are determined by a series of functions previously developed by Bolivar Perez and Sagarna Zabala (2025).

- **Execution modes:** The three selectable execution modes offer different functionalities, each assigned to one of the assembling processes.

  - **Automatic assembly:** where the vision system detects and determines what boxes and cells to assemble depending on availability. This mode performs an automatic assembly of the first available box with the first four available cells, not needing further user prompts than the one that starts this mode.

  - **Manual assembly:** where the user is then prompted with emergent windows to choose the box and cells to be assembled. This mode performs a manual assembly, enabling the user to select what box and which cells to use for the assembly, the availability of the selected objects is checked with the vision system to ensure that the assembly process will be done correctly.

  - **Disassembly:** where the user will select through an emergent window what box to disassemble and return its components to their original placements. This mode performs the disassembly of the box that the user selects, it ensures that the selected box has indeed been assembled beforehand, and returns the box, lid and cells to the position they were retrieved from.

- **Manual movement:** The user can control the TCP position manually via directional buttons. Six buttons are assigned for the displacement axis and another six are assigned for rotational. Movement is implemented by applying incremental translation and rotational transformations (`transl`, `rotx`, `roty`, `rotz`) to the current pose, therefore performing only TCP reference movements. These movements are sent as non-blocking commands to ensure UI responsiveness.

To prevent UI blocking during robot motion, threading was introduced for non-blocking calls. This allows the `QTimers` and signal slots to remain responsive even during movement commands.

- **Manual speed control:** A horizontal slider allows manual adjustment of robot speed between 1 and 100 mm/s. The selected speed is then applied globally. The slider is only enabled in manual mode to prevent interference with automated execution.
- **Gripper movement:** A series of contextual buttons call the gripper control functions found in the `Gripper` class to control the gripper's position. The buttons enable the positioning of the gripper in five different stances: open, open after dropping a cell, grabbing a box, grabbing a lid, and grabbing a cell. By establishing these five positions, it allows precise control, ensuring repeatability and ease of use, and limits the complexity of the manual gripper control. As with speed, gripper control widgets are restricted to manual mode.
- **Maintenance:** As an additional feature, two maintenance buttons enable the visualization and deletion of the current historical joint positions, speeds, and accelerations throughout the performed paths. More details on maintenance handling can be found in the Section 5.8.2.
- **Error handling:** Various error handling features, such as ensuring that the manual control does not take the robot off-limits, or skipping targets if they cannot be found, have been implemented to ensure both the safety of the users and the correct functionality of the cell. Information, warning and error messages regarding general states, connectivity issues, robot movement and such are displayed through the terminal to ensure that the user has complete comprehension of the current state of the process and general program flow.

6.2. System-level analysis

This analysis aims to evaluate the system as a whole, observing global characteristics such as latency, positional accuracy, and safety measures.

6.2.1 Integration and Interoperability

The system was evaluated for seamless communication between the UI, RoboDK simulation environment, and physical- robot execution via the RoboDK API. Special attention was paid to thread synchronization, real-time command delivery, and data consistency between virtual and physical states. The UI reliably sent commands to RoboDK without noticeable delays and real-time data was accurately reflected in the virtual model, ensuring synchronization with the DT. Finally, multi-threading mechanisms allowed concurrent reading of sliders, mode switches, and robot motion without blocking or crashes.

6.2.2 Latency and positional accuracy

System latency was measured in three primary areas: UI input to robot movement response, UI input to virtual gripper response, and physical gripper execution and robot reconnection. Data extracted and processed are located in the figures found in Appendix A.6.

Positional accuracy was measured by comparing the physical and virtual joints of ten positions during the execution of the assembly and disassembly process. Said comparison is also found in the Figures of appendix A.6.

- Average UI robot movement input to physical robot movement latency: 79ms.
- Average UI gripper movement input to virtual gripper movement latency: 234ms.
- Average UI gripper movement input to physical robot reconnection: 2.9s.
- Average virtual-physical position difference standard deviation: 0.00462323 deg.

6.2.3 Safety and Error Handling

Safety protocols were implemented in the form of mode restrictions (manual vs automatic), UI blocking, and confirmation buttons. The system was also tested for resilience against unexpected inputs or communication delays. Manual mode prevented accidental execution of full programs, automatic mode prevented any interference with the predefined script, and no system crashes were observed during simulated communication dropouts or invalid pose inputs.

6.3. Component functional analysis

On the other hand, this analysis compares the expected outcomes of each individual component of the DT system, to the actual developed functionalities.

6.3.1 Virtual model

Accurately mirroring the physical station, the virtual model ensures that all possible aspects of the physical station are correctly represented. All virtual objects mimic their physical counterparts, and ensure a virtual representation, from the dimensions and movements of the UR10e robot arm, to the different colors of the boxes and lids.

6.3.2 UI

Designed for an intuitive and robust interaction, modularity and real-time control, the testing performed showed responsive, well labeled controls and dynamic mode switching, without any contextual or logical errors. The UI, including its visualization in both manual and automatic modes can be observed in Figure 6.1.

Figure 6.1: UI, including all contextual button visualization for both manual mode (left) and automatic mode (right).

6.3.3 Manual robot control

Executing the manual movement in both the virtual and physical robots demonstrated a precise and adaptable approach for the manual positioning of the robot in the robotic cell.

6.3.4 Gripper and speed control

The implemented buttons that enabled positional gripper movement translated into a simple experience when controlling the gripper, removing unnecessary complexity, whilst the slider controlled speed modification enabled the user to have an overall full control of the robot's movement when used alongside the manual control.

6.3.5 Script execution

Enabling the assembly process mode selection and giving the user the ability to pause and resume said execution provided flexibility and transparency. The assembly processes executed without interruption or command overlap, along with the consistent time measurements and transitions between program states were proved throughout the tests and evaluation.

6.3.6 Trajectory calculation

Applying the CasADi framework for the calculation of the paths translated to a higher computation time, resulting in a delay of two to three seconds before the robot started moving. However, time-optimized paths were faster but less smooth, energy consumption-optimized trajectories showed a visible reduction of joint acceleration but remained slower, and path-optimized solutions offered a smoother and more visually appealing motion. Generated trajectories were feasible without any manual corrections and were all successfully executed in simulation. The average times of execution for each step in the test path, as well as all executed trajectory times can be found in Appendix A.7.

Additionally, safe navigation in constrained environments was tested with the use of the obstacle avoidance implemented into the trajectory calculation, making the robot follow a path that curved smoothly around the artificial obstacle, successfully avoiding collisions. This test was only carried out for the test path and can be seen in the comparison between the trajectory carried out by the TCP during a normal execution of the test path and an execution of the test path including an obstacle at the end of the third step. Said comparison is found below in Figure 6.2.

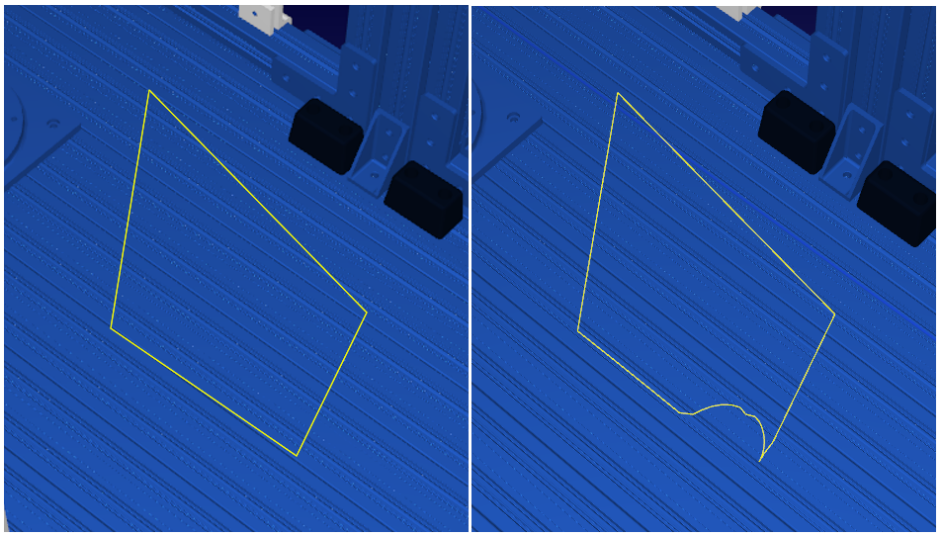


Figure 6.2: Side to side comparison of the trajectory carried out by the TCP during a normal execution of the test path (left) and the one carried out during an obstacle avoidance optimization of the test path, with the obstacle located at the end of the third step (right).

6.3.7 Maintenance

Extracting the data from the history script to later process and visualize it provided five graphs plus an additional three historical data graphs. Comparing the generated graphs to those provided by the RoboCharts add-in resulted in minor differences assigned to the different sampling rates of the graphs, and therefore ensured the validity of the graphs. Despite the maintenance determination generating a total of eight graphs, the RoboCharts add-in only generated three, and therefore only three graphs could be compared. However, all graphs can be observed in Appendix A.8.

6.3.8 Digital twin synchronization

The synchronization of the DT proved to be accurate and reliable, reflecting the behavior of the physical robot under the tested conditions, providing minor deviations attributed to the difference in real-world motor dynamics under load or acceleration and two-way connection delay. Any malfunctions encountered were minor and inconsistent, and were determined to be caused by hardware quality. The developed DT served as a valid predictive tool for previsualizing robot tasks and validating motion plans, as well as monitoring and controlling the robotic functions carried out in the cell.

7. Discussion

This section, assigned to the communication phase, will discuss and evaluate the process and method through which the objectives were achieved. This project has created a DT of a robotic battery assembly and disassembly, achieving the synchronization of the physical and virtual robots. To begin, extensive research about DTs was carried out, in which the different types of DTs and their various classifications were studied in section 2.1, together with the analysis of the research in the literature review, which helped improve the understanding of the benefits of DTs, their uses and functions, and their possible combinations with other technologies to enhance them in future research.

Following this, the implementation phase took place, where the need to synchronize the virtual model with the physical model to obtain a fully functional DT presented several challenges that led the development process to evolve and adapt to the obstacles encountered in order to achieve the objectives. Finally, several tests were conducted to verify the full functionality of the DT, successfully obtaining a developed DT that meets the planned goals.

7.1. Method

The development of the connected DT for robotic battery assembly was systematically structured following the DSRM, as presented in chapter 4. This methodology played a fundamental role in ensuring an iterative, goal-oriented, and empirically validated approach, allowing the project to not only meet its technical objectives but also maintain methodological coherence throughout the phases.

The identified problem focused on the lack of real-time synchronization and predictive capabilities between a physical robotic system and its virtual counterpart, which is critical in the automated industry. This motivation materialized during the early stages of the implementation, including the design of a simple but scalable system architecture connecting the UR10e robot, a local computer, and a user interface via local network connection.

The problem was then broken down into clear and measurable objectives, which became the foundation for the implementation. These included creating an accurate 3D model of the robotic assembly cell, ensuring real-time bidirectional synchronization between the physical and virtual models, designing an intuitive UI to control the robot, and finally implementing advanced computation modules such as trajectory optimization. These objectives directly influenced the software architecture, with specific implementation of functionalities aligned to each one; having to create the model in RoboDK, developing the UI in a python script, and such.

The design and development, being the heart of the implementation, contained the software selection, the modeling of the full physical station and all its related objects, the modular structuring and development of the python UI, and the integration of the trajectory refining and joint data visualization as the advanced computation.

With this, the theoretical models were effectively transformed into a fully functional and interactive DT system.

The artifact was demonstrated under controlled evaluative conditions using both the physical UR10e robot and its developed DT. The execution of the predefined assembly process and real-time bidirectional synchronization of poses and commands were evaluated as successful. The validation of communication between the virtual and physical environment, and the functionality of the advanced computation modules were all confirmed to work as intended. This ensured that the problem identified at the start of the project had been correctly addressed. The synchronization accuracy and latency between the physical and virtual robots was assessed, with addition of the trajectory refining methods being evaluated for performance.

Finally, through the comprehensive documentation of the research process in this report carried out in the discussion and analysis, the communication phase was fulfilled, where every component; from the problem statement and methodology, to implementation, testing and future work, was explained in a level of detail that ensured transparency, replicability, and potential further development. Therefore it can be assured that the development process was aligned with the scientific research standards provided with the DSRM, ensuring that the project has remained structured, iterative, and aligned with its purpose.

7.2. Project Content, Development, and Results

The project focused on achieving the set objectives, which aimed at obtaining a DT of a UR10e whose function is based on the assembly and disassembly of batteries with a gripper. Until the successful creation of the DT, several challenges were encountered, most of which were resolved, as well as changes in the design of the cell's functionality, which presented obstacles that were overcome.

- **Physical and virtual gripper model mismatch:** To import the virtual model of the gripper, the RoboDK libraries were used, and the RobotiQ 2F-85 gripper mechanism was imported directly. However, the mechanism is not identical to the physical model of the gripper, as the fingers of the physical one are slightly longer. When searching on the Robotiq (n.d.) website, the difference between the fingers of the RobotiQ 2F gripper for 85mm and 140mm openings was observed, as seen in the left side of the Figure 7.1. This led to the conclusion that the physical gripper used at the station is a combination of the 85mm opening with the fingers of the 140mm gripper, as shown on the right side of the Figure 7.1. Due to this, there is no virtual model with movement identical to the physical model. However, although this makes the visualization not identical to reality, it does not pose any serious inconvenience for the model.

Figure 7.1: RobotiQ 2F-85 and 2F-140 gripper (left) versus RobotiQ 2F-85 gripper used in the physical station(right)

- **Gripper delay:** During the process of obtaining external control of the RobotiQ 2F-85 gripper, several limitations were encountered, as reasoned in section 5.6. Among them, it is observed that the method to control the tool remotely, (by sending the command line that executes the script via socket connection), causes the physical and virtual models to disconnect. For this reason, after sending the program through socket, the reconnection between both models is performed. This reconnection causes a delay of approximately two seconds every time the gripper is moved. This slows down the assembly process being executed due to the time taken to complete the reconnection and have the robot become available again.

  On the other hand, because the communication between the physical and virtual grippers is limited, they do not share the same programs or methods for their movement, as argued in section 5.6. This causes a slight further delay since, to execute the movement on the virtual gripper, the mode in which the commands are being executed must be changed, and after moving the virtual gripper, the mode that executes commands on the physical robot must be restored (more about this process can be seen in Appendix A.5).

  These two small delays lead to a wait of two or three seconds, as explained in subsection 6.2.2, which visually makes the assembly process appear slower. However, these delays ensure that the tool functions perfectly and guarantee that the objects have been properly gripped and delicately released in the correct positions.

- **Occasional DT malfunction:** Although the developed DT system is considered successful, occasional malfunctions occur. These malfunctions involve the virtual robot momentarily appearing in the zeroed configuration, completely stretched out. An extensive investigation of the causes of this malfunction has been carried out, and the root of the problem has been determined. The series of commands that take place in the `Gripper` class for the gripper control are the main cause of this malfunction. The reason is due to the succession of the commands, which perform the following: change the simulation method on RoboDK to execute

on the simulation only (to be able to move the virtual gripper and perform the attach scripts), move the virtual gripper, execute the required attach script virtually on RoboDK, change the execution method back to run on the robot, disconnect from the robot to be able to send the physical gripper command via socket connection, send the command line via socket, and finally reconnect to the physical robot through RoboDK. This series of changing execution mode, virtual script execution, and reconnection, with such a high frequency, causes the RoboDK app to overload and malfunction.

During said malfunction, when the virtual robot reads the position it has to move to, the malfunction causes it to perform an invalid movement, that sends the virtual robot immediately to its zeroed position. It is noteworthy to know that the virtual robot only remains in the zeroed position for a split second, as it quickly re-reads the position and correctly replicates it. However, this malfunction causes a visual interruption in the smooth virtual assembly process. It is of great importance to note that this malfunction does not affect the physical robot whatsoever, as it performs the assembly and disassembly processes without any issues.

Malfunction encounters only happen occasionally, but with the execution of the gripper control functions being so frequent, the probability of encountering one during the complete execution of the assembly and disassembly process is considerable. The noted malfunction depends on the fluidity of the RoboDk app, and varies from hardware used to execute the DT, ranging from one or two malfunctions per whole assembly process, to having no malfunctions at all in some computers. The DT system has been tested in three different computers, ranging from low-end to high-end gaming computers. Malfunction encounters vary greatly depending on hardware quality.

Equally, low hardware quality also causes mismatching in the synchronization between the physical and virtual robots, showing a complete synchronization in high-end computers, whilst having some virtual-physical speed mismatch on lower-end computers.

- **Uneven trajectory refined movement:** The trajectory refining functions are designed to perform advanced trajectory optimization, `joint_path()` using simple interpolation, and `casadi_path()` using the CasADi symbolic framework. These optimizations result in a well-structured trajectory composed of discrete positions that, when plotted, form a continuous and smooth path from the robot's current pose to the desired target pose. However, despite the quality of the computed path, the actual execution is constrained by how the individual steps are transmitted to the robot. Specifically, the path is executed with each interpolated waypoint being sent in series as a separate `MoveJ` command, and due to the synchronous nature of these commands, the robot must finish executing the current `MoveJ` before the next `MoveJ` command can be executed.

  This inherent and inevitable waiting period causes the robot to momentarily decelerate and pause between each movement, breaking the continuity of motion and preventing true trajectory streamlining. As a result, the robot cannot execute the path as one seamless, fluid gesture, but instead performs a sequence

of discrete movements. Nevertheless, this limitation does not lead to erratic or janky behavior; the robot transitions between the points in a stable and controlled manner, with only brief pauses that slightly slow down the overall execution. Executing the movements from the teach pendant could solve this issue, but that option is not viable as per this system's architecture.

Despite the challenges encountered throughout the development of the DT, it should be noted that these did not cause its development to result incomplete or to underperform. These obstacles were either solved or mitigated, making the DT fully functional and successfully achieving all the proposed objectives. These setbacks have been of great use in learning and development for the participants, managing to overcome adverse situations and demonstrating their problem-solving abilities.

7.3. Sustainable Development

The results obtained make it possible to address environmental aspects. DTs create a digital version of a physical object or process, and for this reason, it is possible to simulate the design of a new product or change the configuration of a process, thus reducing the energy consumption produced (Zhang et al., 2024). Similarly, it allows real-time monitoring of the robot's consumption, showing the user where it is most inefficient and enabling them to identify opportunities to reduce unnecessary consumption and/or waste.

Moreover, according to the United Nations' 2030 Agenda for Sustainable Development, this project contributes to the realization of several of the Sustainable Development Goals (SDGs). Specifically, it targets four large objectives: Infrastructure (Objective 9), by promoting the use of digital twins and robotic automation in manufacturing; Economic Growth (Objective 8), by raising productivity and worker safety; Climate Action (Objective 13), by conserving energy and waste in industry processes; and Responsible Consumption and Production (Objective 12), by optimizing production processes and minimizing waste. For more information on this, see Appendix A.9.

In terms of ethical and social aspects, the communication and needs of the user with the DT were taken into account. Regarding the UI, its design was created with ease of use and intuitiveness in mind. On the other hand, the existence of a virtual replica of the physical station allows the user to conduct tests, thereby reducing the potential risks involved in working with a robot, increasing the user's safety when working remotely.

From an economic perspective, by enabling various tests to find the most optimal process or create a new product, the use of the actual robot is significantly reduced, thus reducing electricity and resource consumption, and extending the object's useful life as its wear will be reduced.

8. Conclusion

The conclusion of this project marks the creation and development of a real-time connected DT of a UR10e. The main objective of this project was to obtain a virtual copy of a robotic station connected in real-time and capable of bidirectional communication. In addition to this general purpose, several other objectives proposed at the beginning of the thesis were achieved:

- **System architecture:** a research on the possible software to use was conducted, and it was concluded that RoboDK met most of the necessary features to develop the project. Furthermore, the system architecture was designed according to the resources and needs of the project and prior research to the development of the project was carried out to have a theoretical framework on which to base it.
- **Virtual model:** an exact representation of the physical station was created in RoboDK by importing and modeling the various objects of the station, as well as creating the necessary functions to represent in the simulation the actions carried out by the robot. The real-time connection between the physical and virtual robots was established by connecting the UR10e via Ethernet to the router, in order to connect the computer wirelessly to the robot to access it.
- **User interface:** an intuitive UI was designed, which allows the user to monitor and manually control the robot, in addition to selecting the assembly and disassembly modes, featuring all the desired functionalities and necessary interactive components to control the robot as desired.
- **Advanced computation:** two advanced computing additions were implemented: trajectory refining, that enabled the user to select the method of calculation of the path to follow along the assembly process, and maintenance determination, where the user receives the data regarding the robot's joint usage in a visual and interactive manner that allows them to detect any abnormal joint behavior and determine when maintenance should be carried out.
- **Testing and validation:** an evaluation was carried out, in which the file containing the DT was opened on a third-party computer external to the development of this project, to ensure its usability on the average computer, as well as to confirm that all the previously mentioned objectives were successfully achieved.
- **Analysis:** after validating the obtained results, an analysis was conducted to identify possible areas for improvement, as mentioned in Chapter 9. Additionally, due to the information studied in the literature review, it was confirmed that a DT is a tool with great potential in Industry 4.0, but still requires further study and integration with other emerging technologies.

The realization of the proposed objectives demonstrates that the balanced combination of an adequate literature review, a suitable methodology, and a practical implementation, along with the feedback obtained after each test carried out, has ensured that the work completed was successful.

9. Future work

Considering the results obtained and the objectives reached in this project regarding the use of DT for robotic battery assembly and disassembly, several future areas of work can be identified. Said areas would allow an expansion upon the advancements achieved in this study. These areas include:

- **Editing current assembled state:** Currently, in the situation of a power outage or a condition that causes the UI to close unexpectedly, the data regarding the assembled objects is lost. This translates into a required reset and repositioning of all the objects in the station each time the UI is started, enforcing the initial state of the robotic station to be the same every time the UI is started. It could be desired to start the assembly process at a point in which the objects are not in their original positions, for example having one box successfully assembled already. To enable this, an option can be implemented into the UI where the user introduces which boxes have been assembled with which cells, enabling a starting point that is not the complete resetted station. This could be evaluated with the vision system to ensure that the input given by the user is correct. This would also require the need to change the attach scripts in the virtual model in order to attach the closest item instead of a specific one.
- **External object manipulation:** The current assembly and disassembly logic assumes that the state of the objects inside the station remains constant, without any interference from outside the station. This ensures that the logic of the station only regards the displacement of the objects performed by the robot, and does not take into account any human interaction with the assembly process. Through this logic, the external manipulation of any object will cause a mismatch in the process' position logic, causing a system malfunction. Therefore, if external manipulation is desired to be implemented, the UI must be modified to include an option to indicate that an external displacement of an object has been carried out. This could be for example the manual disassembly of an assembled box, or the repositioning of a cell in the shelf.
- **Expand gripper user control:** The user control on the gripper during manual mode is limited to 5 different positions of the gripper. This reduces complexity and gives the manual gripper control repeatability and accuracy, as the user does not need to select the exact angle of aperture for the gripper every time the gripper is to be moved. However, in further implementations of this robotic station, being able to manually choose the gripper's aperture degree might be desired. This would enrich the UI panel, allowing the user not only to have buttons for controlling the gripper in the precise aperture angles to grab the station objects, but also to have a slider with which to control the gripper's full motion.
- **Error handling using logs:** Currently, error handling is primary achieved through a combination of try-except blocks and direct `print()` statements, which

while functional, lack the consistency and traceability for a higher-scale method of error handling, as undetected malfunctions can negatively affect functionalities. Therefore, if this project were to be expanded, an implementation of error log handling is highly suggested, as by introducing a logging mechanism, each method could record its operational status, warnings and exceptions in a unified format. Including timestamps, severity levels, and enhancing visibility, the user and the possible developers could diagnose and resolve the issues more efficiently. Integrating a centralized error logging system to this program would provide a structured and persistent mechanism for capturing and analyzing the UI runtime behavior, as logs can differentiate between expected operational anomalies and critical faults, making maintainability and scalability easier.

- **VR DT visualization:** While designed with the goal to be intuitive and user-friendly, the visualization of the DT can be taken a step further by implementing VR. RoboDK contains the necessary drivers and connectivity to allow a rather seamless connection with VR glasses such as the Meta Quest 3. If there were to be any VR glasses available for use, it is highly recommended to try the connection with RoboDK, as this would allow the visualization of the station through said glasses, as can be seen in the RoboDK (2015-2025), where the process of setup and connection is explained.

By addressing the areas mentioned in future research, a more flexible DT could be obtained, allowing the user to modify the way the work is developed, as well as an extended User Interface (UI). These improvements would enhance the benefits of using DTs in the manufacturing industry due to the improvements they would deliver to this field.

Bibliography

Amaya, Jairo Amaya (2010). *Sistemas de información gerenciales: Hardware, software, redes, Internet, diseño*. Ecoe ediciones. ISBN: 978-958-648-635-4.

Andrade, Andre (2024). *The 4 Levels of the Digital Twin Technology*. URL: https://vidyatec.com/blog/the-4-levels-of-the-digital-twin-technology/.

Attaran, Mohsen, Sharmin Attaran, and Bilge Gokhan Celik (June 2023). "The impact of digital twins on the evolution of intelligent manufacturing and Industry 4.0". In: *Advances in Computational Intelligence* 3.3. ISSN: 2730-7808. DOI: 10.1007/s43674-023-00058-y.

Attaran, Mohsen and Bilge Gokhan Celik (Mar. 2023). "Digital Twin: Benefits, use cases, challenges, and opportunities". In: *Decision Analytics Journal* 6, p. 100165. ISSN: 2772-6622. DOI: 10.1016/j.dajour.2023.100165.

Bolivar Perez, Paula and Aitor Sagarna Zabala (2025). *Implementation of a robotic cell for high precision battery assembly and disassembly*. Skövde: Högskolan i Skövde.

Creswell, J. W. (2014). *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE Publications. ISBN: ISBN 978-1-4129-6557-6 (pbk.) URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.ucg.ac.me/skladiste/blog_609332/objava_105202/fajlovi/Creswell.pdf.

Dubarry, Matthieu, David Howey, and Billy Wu (June 2023). "Enabling battery digital twins at the industrial scale". In: *Joule* 7.6, pp. 1134–1144. ISSN: 2542-4351. DOI: 10.1016/j.joule.2023.05.005.

Fu, Yang et al. (June 2022). "Digital Twin for Integration of Design-Manufacturing-Maintenance: An Overview". In: *Chinese Journal of Mechanical Engineering* 35.1. ISSN: 2192-8258. DOI: 10.1186/s10033-022-00760-x.

Fuller, Aidan et al. (2020). "Digital Twin: Enabling Technologies, Challenges and Open Research". In: *IEEE Access* 8, pp. 108952–108971. ISSN: 2169-3536. DOI: 10.1109/access.2020.2998358.

Gokhale, Pradyumna, Omkar Bhat, and Sagar Bhat (2018). "Introduction to IOT". In: *International Advanced Research Journal in Science, Engineering and Technology* 5.1, pp. 41–44. ISSN: ISSN (Online) 2393-8021 ISSN (Print) 2394-1588. URL: https://www.researchgate.net/profile/Omkar-Bhat/publication/330114646_Introduction_to_IOT/links/5c2e31cf299bf12be3ab21eb/Introduction-to-IOT.pdf.

Grieves, Michael and John Vickers (Dec. 2016). "Origins of the Digital Twin Concept". In: *Florida Institute of Technology*. DOI: 10.13140/RG.2.2.26367.61609.

— (Aug. 2017). "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems". In: *Transdisciplinary Perspectives on Complex Systems*. Springer International Publishing, pp. 85–113. ISBN: 9783319387567. DOI: 10.1007/978-3-319-38756-7_4.

Hevner et al. (2004). "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1, p. 75. ISSN: 0276-7783. DOI: 10.2307/25148625.

Husseini, K. et al. (2022). *Development of a Digital Twin for Improved Ramp-Up Processes in the Context of Li-Ion-Battery-Cell-Stack-Formation*. 9th CIRP Conference

on Assembly Technology and Systems. URL: https://www.sciencedirect.com/science/article/pii/S2212827122001512.

Inamura, Tetsunari (May 2023). "Digital Twin of Experience for Human–Robot Collaboration Through Virtual Reality". In: *International Journal of Automation Technology* 17.3, pp. 284–291. ISSN: 1881-7629. DOI: 10.20965/ijat.2023.p0284.

Javaid, Mohd, Abid Haleem, and Rajiv Suman (2023). "Digital Twin applications toward Industry 4.0: A Review". In: *Cognitive Robotics* 3, pp. 71–92. ISSN: 2667-2413. DOI: 10.1016/j.cogr.2023.04.003.

Kragic, Danica et al. (July 2018). "Interactive, Collaborative Robots: Challenges and Opportunities". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. IJCAI-2018. International Joint Conferences on Artificial Intelligence Organization, pp. 18–25. DOI: 10.24963/ijcai.2018/3.

Kritzinger, Werner et al. (2018). "Digital Twin in manufacturing: A categorical literature review and classification". In: *IFAC-PapersOnLine* 51.11, pp. 1016–1022. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.08.474.

Kuts, Vladimir et al. (Feb. 2019). "Digital twin based synchronised control and simulation of the industrial robotic cell using virtual reality". In: *Journal of Machine Engineering* 19.1, pp. 128–144. ISSN: 2391-8071. DOI: 10.5604/01.3001.0013.0464.

Lv, Zhihan and Shuxuan Xie (Nov. 2022). "Artificial intelligence in the digital twins: State of the art, challenges, and future research topics". In: *Digital Twin* 1, p. 12. ISSN: 2752-5783. DOI: 10.12688/digitaltwin.17524.2.

Madakam, Somayya, R. Ramaswamy, and Siddharth Tripathi (2015). "Internet of Things (IoT): A Literature Review". In: *Journal of Computer and Communications* 03.05, pp. 164–173. ISSN: 2327-5227. DOI: 10.4236/jcc.2015.35021.

Naseri, F. et al. (June 2023). "Digital twin of electric vehicle battery systems: Comprehensive review of the use cases, requirements, and platforms". In: *Renewable and Sustainable Energy Reviews* 179, p. 113280. ISSN: 1364-0321. DOI: 10.1016/j.rser.2023.113280.

Peffers, Ken et al. (Dec. 2007). "A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3, pp. 45–77. ISSN: 1557-928X. DOI: 10.2753/mis0742-1222240302.

Pérez, Luis et al. (May 2020). "Digital Twin and Virtual Reality Based Methodology for Multi-Robot Manufacturing Cell Commissioning". In: *Applied Sciences* 10.10, p. 3633. ISSN: 2076-3417. DOI: 10.3390/app10103633.

Pericherla, Suryateja (2025). "IoT Architecture Layers". In: *StarterTutorial Blogs*. URL: https://www.startertutorials.com/blog/iot-architecture-layers.html.

Qi, Qinglin and Fei Tao (2018). "Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison". In: *IEEE Access* 6, pp. 3585–3593. ISSN: 2169-3536. DOI: 10.1109/access.2018.2793265.

Rathore, M. Mazhar et al. (2021). "The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities". In: *IEEE Access* 9, pp. 32030–32052. ISSN: 2169-3536. DOI: 10.1109/access.2021.3060863.

RoboCharts (2024). *RoboCharts*. URL: https://robodk.com/addin/com.robodk.app.robocharts.

RoboDK (2025a). URL: https://robodk.com/.

RoboDK (2025b). *RoboDK - Universal Robots*. URL: https://robodk.com/doc/en/Robots-Universal-Robots.html#UR-StartProg.
— (2025c). *RoboDK Documentation*. URL: https://robodk.com/doc/en/Basic-Guide.html#Guide.
— (2015-2025). *RoboDK API*. URL: https://robodk.com/doc/en/PythonAPI/robodk.html.
Robotiq (2019). *Robotiq 2F-85/2F-140 Instruction Manual*. https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_Instruction_Manual_e-Series_PDF_20190206.pdf.
— (n.d.). *Adaptive Grippers*. URL: https://robotiq.com/products/adaptive-grippers#Two-Finger-Gripper.
Sherwani, F., Muhammad Mujtaba Asad, and B.S.K.K. Ibrahim (Mar. 2020). "Collaborative Robots and Industrial Revolution 4.0 (IR 4.0)". In: *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. IEEE, pp. 1–5. DOI: 10.1109/icetst49965.2020.9080724.
Simumatik (2025a). URL: https://simumatik.com/simumatik-platform-features/.
— (2025b). *Simumatik Documentation*. URL: https://docs.simumatik.com/manual/getting_started/.
Singh, Ankit (2025). *The Role of Robotics in Electric Vehicle Battery Production*. URL: https://www.azorobotics.com/article.aspx?ArticleID=735&.
Singh, Maulshree et al. (May 2021). "Digital Twin: Origin to Future". In: *Applied System Innovation* 4.2, p. 36. ISSN: 2571-5577. DOI: 10.3390/asi4020036.
Sparke, Carmen (2024). *Maria Forsyth is our leading researcher in electrochemistry*. URL: https://www.theaustralian.com.au/special-reports/research-magazine/maria-forsyth-is-our-leading-researcher-in-electrochemistry/news-story/107b4643750ab0e253701cfdf7998e89?.
Stevens, Tim (2025). *How longer-lasting, faster-charging batteries are coming via software updates*. URL: https://www.theverge.com/electric-cars/627597/breathe-battery-software-charging-speed-range-volvo?.
Tao, Fei et al. (Apr. 2019). "Digital Twin in Industry: State-of-the-Art". In: *IEEE Transactions on Industrial Informatics* 15.4, pp. 2405–2415. ISSN: 1941-0050. DOI: 10.1109/tii.2018.2873186.
UniversalRobots (2023). *UR10e E-Series Datasheet*. https://www.universal-robots.com/media/1807466/ur10e_e-series_datasheets_web.pdf.
UR (2025). URL: https://www.universal-robots.com/products/ur10e/.
Wang, Baicun et al. (June 2021). "Smart Manufacturing and Intelligent Manufacturing: A Comparative Review". In: *Engineering* 7.6, pp. 738–757. ISSN: 2095-8099. DOI: 10.1016/j.eng.2020.07.017.
Wang, Wenwen et al. (June 2021). "Application of Digital Twin in Smart Battery Management Systems". In: *Chinese Journal of Mechanical Engineering* 34.1. ISSN: 2192-8258. DOI: 10.1186/s10033-021-00577-0.
Zhang, Zichao et al. (Apr. 2024). "A Review of Digital Twin Technologies for Enhanced Sustainability in the Construction Industry". In: *Buildings* 14.4, p. 1113. ISSN: 2075-5309. DOI: 10.3390/buildings14041113.

A. Appendices

A.1. Software versions

Name	Version	Type
RoboDK	v5.9.0	Simulation and Programming software
Polyscope	v5.20.0.0	UR Software
Python	v3.10.0	Programming Language
Visual Studio Code	v1.100.0	Integrated Development Environment (IDE)

Table A.1: Software versions

A.2. Required Libraries and Extensions for Project Functionality

Library	Type
PyQt6	Framework for creating graphical user interfaces (GUIs) in Python
RoboDK	Library for industrial robot simulation and programming
CasADi	Tool for numerical optimization and automatic differentiation
NumPy	Library for scientific computing and handling arrays and matrices
Pandas	Library for structured data manipulation and analysis
Matplotlib	Library for creating plots and visualizations in Python
Socket	Python standard module for network communication
OpenCV	Library for image processing and computer vision tasks

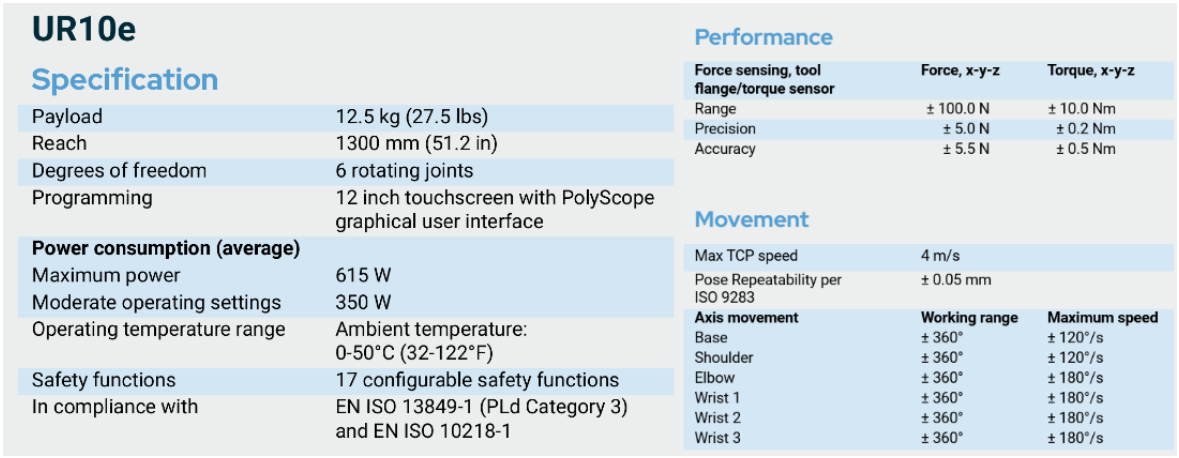
Table A.2: Libraries needed

VSCode Extension	Type
Python	Extension for VSCode providing full Python language support
Pylance	VSCode extension that provides fast and feature-rich Python language support and type checking

Table A.3: Extensions recommended

A.3. Technical Datasheet

Universal Robot UR10e



UR10e

Specification

Payload	12.5 kg (27.5 lbs)
Reach	1300 mm (51.2 in)
Degrees of freedom	6 rotating joints
Programming	12 inch touchscreen with PolyScope graphical user interface
Power consumption (average)	
Maximum power	615 W
Moderate operating settings	350 W
Operating temperature range	Ambient temperature: 0-50°C (32-122°F)
Safety functions	17 configurable safety functions
In compliance with	EN ISO 13849-1 (PLd Category 3) and EN ISO 10218-1

Performance

Force sensing, tool flange/torque sensor	**Force, x-y-z**	**Torque, x-y-z**
Range	± 100.0 N	± 10.0 Nm
Precision	± 5.0 N	± 0.2 Nm
Accuracy	± 5.5 N	± 0.5 Nm

Movement

Max TCP speed	4 m/s	
Pose Repeatability per ISO 9283	± 0.05 mm	
Axis movement	**Working range**	**Maximum speed**
Base	± 360°	± 120°/s
Shoulder	± 360°	± 120°/s
Elbow	± 360°	± 180°/s
Wrist 1	± 360°	± 180°/s
Wrist 2	± 360°	± 180°/s
Wrist 3	± 360°	± 180°/s

Figure A.1: UR10e datasheet

Full Datasheet: (UniversalRobots, 2023)

RobotiQ 2F-85

Datasheet: (Robotiq, 2019)

A.4. Time Plan

The figure A.2 shows the initial time plan that was planned for the completion of the thesis.

Activity	4	5	6	7	8	9	10	11	12
Create virtual model	X	X	X	X					
Connect virtual model			X	X	X	X	X	X	
Create monitoring system							X	X	X
Make predictive analytics									

Activity	13	14	15	16	17	18	19	20	21
Create virtual model									
Connect virtual model									
Create monitoring system	X	X	X	X	X	X	X		
Make predictive analytics				X	X	X	X	X	X

Figure A.2: Gantt chart

In Figure A.3, the comparison between the initial time plan, shown in light gray, and the final time plan followed, shown in dark gray, can be seen. Compared to the first proposed time plan in Figure A.2, several subsections have been added to the indicated sections to clarify and make the time dedicated to each activity more understandable. It is worth noting that due to the continuous changes made to the physical cell and the objects within it, certain tasks took longer than planned, not due to complications, but because of changes made during the process.

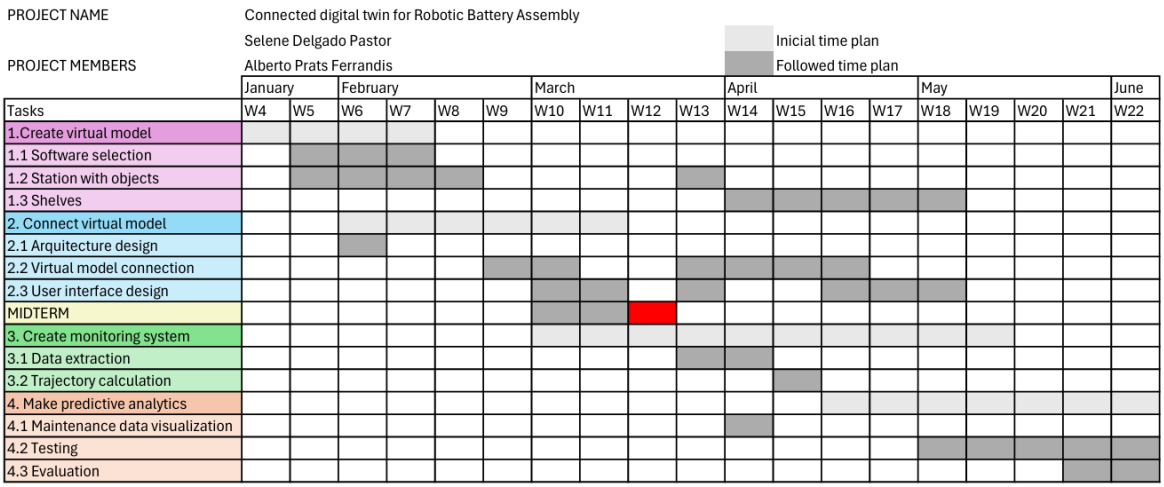


Figure A.3: Followed Gantt chart

A.5. Code Overview

In the following section, an overview of the code is carried out, firstly observing the UML diagram and then dwelling into each method of the classes implemented and providing a brief explanation of their logic. The order of the classes and methods is the order in which they appear in the program files. The main execution of the file resides in the function `RunUI()` which instantiates `Robolink`, prompts the user with the selection of the robot, and starts the execution of the UI. The code of the DT system can be found in `https://github.com/DigitalTwinThesis/DigitalTwin/`.

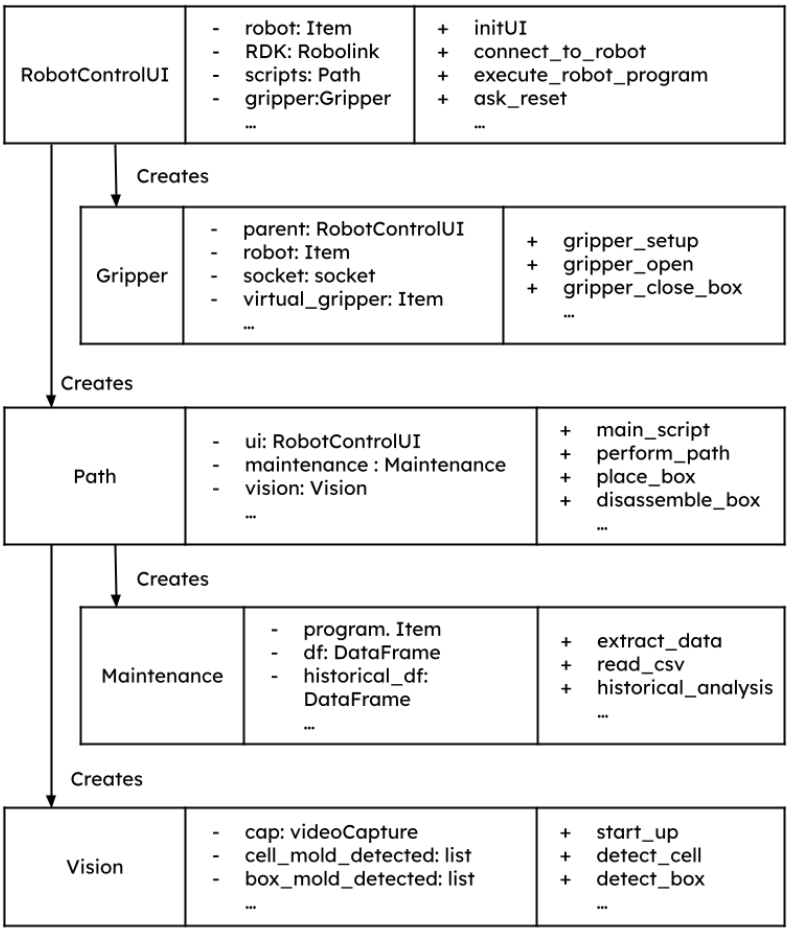


Figure A.4: UML diagram displaying class interaction and inheritance

A.5.1 Class: Vision

This class, programmed and developed by (Bolivar Perez and Sagarna Zabala, 2025), contains the methods necessary to: start to start the video capture of a Universal Serial Bus (USB)-connected camera using `vision.start_up()`, process the images and detect the cells, boxes and lids using `vision.detect_cell()`, `vision.detect_box()` and `vision.detect_lid()` respectively, and view the processed image using `vision.view()`. The detection methods return an array with the positions in which the objects have been detected.

A.5.2 Class: Maintenance

This class contains the methods to extract the joint data from the robot movements to a CSV file, and process said data to generate the desired graphs for visualization. The unit of distance is millimeters and the unit of rotation is degrees.

- `extract_data()` creates the CSV file with the data using `IntructionListJoints` .
- `read_csv()` loads and reads the CSV file, formats the data, deletes duplicated data and smoothens the values. It also establishes the robot's joint limits.
- `robot_joints_range()` reads the range in degrees of each joint and displays it on a graph alongside the mechanical limits of each joint.
- `robot_joints_usage()` reads the utilization of each joint and displays it in a graph.
- `robot_joints_over_time()` reads the position of each joint over time for the duration of the movement and displays it on a graph.
- `robot_joints_velocity()` reads the speeds of each joint over time for the duration of the robot movement and displays it on a graph.
- `robot_joints_acceleration()` reads the accelerations of each joint over time for the duration of the robot movement and displays it in a graph.
- `historical_data()` reads and calculates the total movement of each joint, the average speed and acceleration of each joint, and stores it as a new line in a secondary CSV file for later extraction for the historical data.
- `historical_analysis()` loads and reads the historical CSV file, processes the total movement and average speed and acceleration of each joint for all of the previous saved robot movements, and displays it all in a graph.

A.5.3 Class: Path

This class, originally programmed and developed by Bolivar Perez and Sagarna Zabala (2025) contains the methods; `main_script()` in charge of the logical flow and selection of the assembly and disassembly process, `script_execution()` in charge of the movement of the robot, and the methods: `place_box(), place_cell(), place_full_box (), place_lid(), disassemble_box(), disassemble_cell(), disassemble_full_box (),` and `disassemble_lid()` which are in charge of providing the order of the targets to carry out the assembly and disassembly itself. Having these pre-existing methods, `script_execution()` was modified, and the required methods were created to add the desired functionalities. Instances of both classes `Maintenance` and `Vision` were made in the initialization of this class.

- `script_execution()` was modified to include the necessary selection and loops to ensure responsiveness from the UI and enabling the pausing and resuming of the execution. It was also modified to call the method `perform_path()` instead of using `MoveJ`, to implement the trajectory refining.

- `perform_path()` selects the method of calculation, and calls the pertinent function to calculate the trajectory and perform the robot movement.
- `moveJ_path()` performs a normal `MoveJ` on the robot with the given target, but also appends said movement to the maintenance-related history script.
- `joint_path()` calculates the path to follow through interpolation, and splits the original path into smaller segments that are appended to a list which is then called in series. The robot movement is made with `MoveJ` and said movements are also appended to the history script.
- `casadi_path()` calculates the path to follow given a few parameters that can be added when calling the function `perform_path()`, if no parameters are added, a normal path will be followed. Once this method is called, it will set up the variables for the optimization, minimize acceleration for smooth movement if the parameters indicate a low power mode, initialize the constraints, extract the starting and ending positions of the path, limit the maximum translation per step for smoothness, add obstacle avoidance if the parameters indicate so (this will be performed by establishing the distance squared to be greater than the radius of the obstacle squared), and add the time constraints if the parameters indicate so. Once this is done, the NLP problem will be built, an initial guess will be taken and then the optimization problem will be solved. The trajectory is then extracted from the solution and the path is then followed in series by copying the target pose and updating the position. The movement of the robot is done through `MoveJ` and the movement performed are appended to the history script.

A.5.4 Class: Gripper

This class embarks the connection and control of the gripper, ensuring that the reconnection of the robot always takes place and encapsulating all of the gripper-related methods into a single, independent class. In the initialization of this class, the RoboDK item of the virtual gripper is selected to be able to connect to it and control it, and the method `gripper_setup()` is called upon initialization.

- `gripper_setup()` sets up the gripper by opening and reading the gripper programs and storing them as a command line, if the files are not found, it creates the program from RoboDK and stores it locally. Once this is done, it connects to the gripper via socket connection, prompting the user to retry the connection if it fails.
- `gripper_open()` checks if the gripper is being moved, if not, it sets the mode to simulation (to ensure that the program is ran locally on the simulation and not on the physical robot), moves the virtual gripper, executes the `detach` program to detach any object that the gripper might be holding in the virtual model, and then sets the run mode back to run on the physical robot. Once this is done, the command line extracted beforehand to move the physical gripper is sent to the physical gripper via socket connection. Sending the command line through socket connection forces the disconnection of the robot from RoboDK, so a reconnection to the physical robot is done once the command is sent, including a slight delay to ensure that the command has been sent correctly.

- `gripper_open_cell()` works similarly to `gripper_open()`, but positioning the virtual gripper in a different position, and sending a different command line to the physical gripper.
- `gripper_close_box()` works similarly to `gripper_open()`, but positioning the virtual gripper in a different position, executing `Attach box{box_num}`, (being `box_num` the number of the box that has been passed as parameter to attach) and sending a different command line to the physical gripper. It also checks if the box to be gripped contains any cells, and if so, it executes the required script to also attach them virtually.
- `gripper_close_cell()` works similarly to `gripper_close_box()`, but positioning the virtual gripper in a different position, executing `Attach cell{cell_num}`, (being `cell_num` the number of the cell that has been passed as parameter to attach) and sending a different command line to the physical gripper.
- `gripper_close_lidl()` works similarly to `gripper_close_cell()`, but positioning the virtual gripper in a different position, executing `Attach lid{lid_num }`, (being `lid_num` the number of the lid that has been passed as parameter to attach) and sending a different command line to the physical gripper.

A.5.5 Class: RobotControlUI

This class is in charge of the whole UI functionality, creating and governing the buttons and methods to ensure the correct execution of the whole program. In the initialization of this class, instances of the `Gripper` and `Path` class were made, along with the calling of start-up methods such as `ask_reset()`, `initUI()`, `connect_to_robot()` and the resetting of both the virtual robot and model.

- `initUI()` creates and initializes the UI components and layout, including buttons and sliders: mode selection, manual mode position and rotation controls, speed control slider, move to home button, automatic mode controls (execute, pause and resume), gripper control and maintenance chart visualization and clearance. It also calls the `update_button_states()` method.
- `update_speed()` updates the robot speed taking the value from the slider.
- `start_maintenance()` calls all of the maintenance methods in series to display the joint charts, including the historical data.
- `clear_maintenance()` deletes the history script containing the robot movements and creates an empty one.
- `create_movement_button()` is a helper function to create the manual control buttons to move the TCP.
- `create_rotation_button()` is another helper function to create the manual control buttons to rotate the TCP.
- `activate_manual_mode()` is the method called when the manual mode button is pressed, it sets the mode to manual and calls the `update_button_states()` method.

- `activate_auto_mode()` is the counterpart to the previous method, setting the mode to automatic and calling the `update_button_states()` method as well.
- `update_button_states()` enables and disables the buttons depending on the mode, this includes enabling the manual control buttons only when the manual mode is active to ensure no accidental manual movement is performed during automatic mode and so forth. This method also changes the visual aspect of the manual and automatic mode buttons to visually demonstrate which ones are active.
- `connect_to_robot()` attempts to connect to the UR10e robot if it is not already connected, if the connection fails, it prompts the user to reconnect via the `ask_retry_connection()` method. The user will be prompted with reconnection until the connection is successful or until the user decides not to try again, in this case, the program will end.
- `move_to_home()` moves the robot to the predefined home position if the robot is not busy, if no home is detected a pertinent message is displayed. The movement of the robot is sent as a threaded call of `safe_movement()` to ensure UI responsiveness throughout the movement.
- `start_movement()` starts a continuous movement when a manual movement button is pressed, ensures that the robot is not busy, sets the direction of movement and starts a thirty millisecond timer to check if the button is still being pressed.
- `stop_movement()` stops the movement when the manual movement button is released.
- `start_rotation` starts a continuous rotation when a manual rotation button is pressed, ensures that the robot is not busy, sets the direction of rotation and starts a thirty millisecond timer to check if the button is still being pressed.
- `stop_rotation()` stops the rotation when the manual rotation button is released.
- `perform_movement()` executes the movement of the TCP every timer tick while a manual movement button is being held. It applies a translation to the current pose of the TCP given the direction of movement, and sends the movement as a threaded call of `safe_movement()` to ensure UI responsiveness.
- `perform_rotation()` executes the rotation of the TCP every timer tick while a manual rotation button is being held. It applies a rotation to the current pose of the TCP given the direction of rotation, and sends the movement as a threaded call of `safe_movement()` to ensure UI responsiveness.
- `safe_movement()` performs a safe `MoveJ()` movement with the target received through parameter, ensuring the robot does not exceed its joint limits and the UI does not crash.
- `execute_robot_program()` ensures the robot is not busy and then executes the robot program once. To do so, it obtains the assembly mode by prompting the user with the `ask_mode()` method and then, depending on the user's decision; either executes the `path.main_script()` method directly, prompts the

user to choose a box with `ask_selection()` and then calls `path.main_script()`, or prompts the user to select both a box and cells with `ask_selection()` and `ask_cell_selection()`, to then call `path.main_script()`. The execution of `path.main_script()` is called as a non-blocking threaded call to ensure UI responsiveness.

- `pause_robot_program()` pauses the execution of the robot program in `path.main_script()`, giving the option to resume execution. The pause of the robot movement is not immediate, the robot will finish its current movement (will reach the target it has been sent to go to) and will stop upon its arrival.
- `resume_robot_program()` resumes the execution of the robot program wherever the function
- `pause_robot_program()` has caused it to stop, ensuring that the execution of the assembly process remains as intended.
- `ask_mode()` creates a dialog window to select the mode of execution of the robot program, the options are 'Automatic Assembly', Disassembly' and 'Manual Assembly'. when the user selects the option, the method will return a value depending on their decision and will close the dialog window.
- `ask_cell_selection()` also creates a dialog window, but to select the cell for an assembly, it receives an array of the already selected cells and will only display the buttons for the cells that have not been selected yet, to ensure the same cell is not selected more than once. This method will return the value of the selected cell and will close the dialog window upon selection.
- `ask_selection()` functions in the same way as `ask_cell_selection()`, as it prompts the user to select a box through a dialog window that will close upon selection and will return the value of the selected box.
- `ask_reset()` creates a dialog window to display the image of the reset state of the cell and prompts the user to ensure that the physical cell is in the same state as the image, this ensures that the starting points of both the physical and virtual cells are the same and enforces the responsibility of the user before even starting the UI. the method will return the user's decision and will close the dialog window upon selection.
- `ask_retry_connection()` creates a dialog window to notify that the connection has failed for either the robot or the gripper (the object is received as a parameter). It will prompt the user with the decision of whether to try a reconnection or dismiss. Once again, the method returns the decision as a numerical value and closes the dialog window upon receiving the decision.

A.6. Latency and positional accuracy

In this section, the data extracted and processed to evaluate the latency and positional accuracy of the DT system can be found.

Sample	Delay in communication (s)
1	0,078810000
2	0,078600000
3	0,079480000
4	0,078760000
5	0,078590000
6	0,079550000
7	0,082350000
8	0,079940000
9	0,063980000
10	0,081920000
11	0,076670000
12	0,079160000
13	0,076260000
14	0,080460000
15	0,083670000
16	0,077070000
17	0,085730000
18	0,075830000
19	0,084750000
20	0,086190000
Average (s)	0,079388500
Average (ms)	79,388500000

Figure A.5: UI input to robot movement latency.

Sample	Delay in communication (s)
1	0,22800
2	0,23230
3	0,22710
4	0,23301
5	0,24285
6	0,24140
7	0,23092
8	0,23295
9	0,23192
10	0,23979
Average	0,23402

Figure A.6: UI input to virtual gripper response latency.

Sample	Delay in communication (s)
1	3,26720
2	3,16010
3	2,87305
4	2,89690
5	3,22966
6	2,65798
7	2,82725
8	2,78547
9	2,71312
10	2,77600
Average	2,91867

Figure A.7: UI input to robot reconnection after physical gripper movement.

	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Virtual	-90,98	-120,97	-69,66	-168,04	-1,25	88,44
Physical	-90,98	-120,97	-69,66	-168,03	-1,25	88,44
Virtual	-77,59	-122,28	-97,22	-140,25	280,67	88,97
Physical	-77,6	-122,28	-97,22	-140,25	280,67	88,97
Virtual	-89,99	-106,44	-86,49	-77,09	90,01	0
Physical	-90	-106,43	-86,48	-77,09	90	0
Virtual	-64,04	-85,88	-121	-153,12	-63,93	90,24
Physical	-64,04	-85,88	-121	-153,12	-63,92	90,25
Virtual	-97,51	-115,87	-100,19	-143,95	-7,51	0
Physical	-97,51	-115,86	-100,19	-143,95	-7,52	0
Virtual	-100,4	-97,04	-110,79	-152,17	-10,41	90,24
Physical	-100,41	-97,04	-110,78	-152,16	-10,41	90,25
Virtual	-97,58	-132,89	-77,44	-149,45	-7,85	89,55
Physical	-97,57	-132,89	-77,44	-149,45	-7,85	89,55
Virtual	-90,93	-123,18	-74,72	-160,69	-1,2	88,37
Physical	-90,93	-123,18	-74,73	-160,69	-1,2	88,37
Virtual	-103,8	-142,47	-32,73	-94,8	90	75,81
Physical	-103,81	-142,47	-32,73	-94,8	90	75,8
Virtual	-75,45	-112,39	-96,25	-61,36	90	103,8
Physical	-75,44	-112,4	-96,24	-61,37	90	103,81

Figure A.8: Virtual and physical robot joint comparison for ten positions along the assembly and disassembly process.

A.7. Trajectory Optimization Times

Located below, in Figure A.10 are the tables showing the times for the ten iterations of the different optimization methods tested, per step of the test path, as well as the averages obtained from them in Figure A.9.

	Average times per optimization			
Program step	Time optimization	Normal optimization	Consumption optimization	Obstacle avoidance
1	2,96646089800103	2,98534919990020	3,10177640000074	3,00547939799887
2	2,97352699803177	2,98523397900052	3,11341659999840	2,98796186059936
3	2,98963559379948	3,02168670168038	3,20072938000034	3,06506476000022
4	3,00412449939762	3,03559719999475	3,19146979000034	3,01110918751345
Total	11,93374798922990	12,02786708057590	12,60739216999980	12,06961520611190

Figure A.9: Average execution times for each step in the test path, per type of optimization.

	Time optimization iteration step duration (s)										
Step number	1	2	3	4	5	6	7	8	9	10	Average (s)
1	2,92933971469360	2,89785829098320	2,93676655171858	3,00400945776439	2,98470555180749	2,95662770923120	2,87554999821060	3,00659298666964	3,01370848936289	3,05945022956871	2,96646089800103
2	2,99624522184251	2,99842479634485	3,02301982254648	3,01284824096114	2,88333813413108	2,89962771934333	2,93655496345606	2,92308685140302	3,00312610207887	3,05899612821035	2,97352699803177
3	2,99939383212125	2,97021758533318	2,94215348746456	2,94774660911615	2,98604911302720	3,06014851821299	3,05801069883304	2,91605057459982	2,96100484303851	3,05558067624811	2,98963559379948
4	3,00602882597348	3,00132616439921	3,01344578238407	2,98081460881632	2,94932122583231	3,10340096308825	2,99769269961838	2,97467178044455	3,08879416281167	2,92574878060796	3,00412449939762
Total	11,93100759463080	11,86782683706040	11,91538564411370	11,94541891665800	11,80341402479810	12,01980490987580	11,86780836011810	11,82040219311700	12,06663359729190	12,09977781463510	11,93374798922990

	Normal optimization iteration step duration (s)										
Step number	1	2	3	4	5	6	7	8	9	10	Average (s)
1	2,91693211339898	2,95165762206685	3,01425883099093	3,02213920357516	2,97962054857415	2,99671223960312	3,01920903155341	2,99152005432591	2,92871616160606	3,03272619330745	2,98534919990020
2	2,94509548873286	3,03405533376066	3,01867865111426	2,89166392394877	3,04095130963914	3,06486344130137	3,04482690694014	2,92686223304519	2,95270171581702	2,93264078570580	2,98523397900052
3	3,04009904284455	3,07089578306839	2,92946350035519	3,01691649069377	3,10254344345869	3,05875854140925	3,04714227832420	2,97994700654956	3,00908915870379	2,96201177139641	3,02168670168038
4	3,12102918173590	3,00747590351393	3,00965291210220	3,04556464975501	3,07040322444508	3,13008205421141	2,99693482112593	2,98860066515764	3,04629562696451	2,93993296093588	3,03559719999475
Total	12,02315582671230	12,06408464240980	11,97205389456260	11,97628426797270	12,19351852611710	12,25041627652520	12,10811303794370	11,88692995907830	11,93680266309140	11,86731171134550	12,02786708057590

	Consumption optimization iteration step duration (s)										
Step number	1	2	3	4	5	6	7	8	9	10	Average (s)
1	3,11485384302464	3,09332789663273	3,14312822511969	3,07439052403511	3,03442664719780	3,01338429763215	3,08116134629095	3,19266029590871	3,10182520387690	3,16860572028873	3,10177640000074
2	3,12045806243712	3,16332914016695	3,11878759024314	3,11899335132080	3,06883762014103	3,13657465840391	3,01716643066106	3,19701751267167	3,15222107735780	3,04078055658052	3,11341659999840
3	3,29659465470216	3,17635420837031	3,23755941944245	3,10448870061502	3,22862601235836	3,27987626691345	3,10209256176695	3,22763991236365	3,24098840196214	3,11307366150891	3,20072938000034
4	3,11117817055593	3,12271449208956	3,18516071167122	3,23307793080565	3,13610970262929	3,28551660638619	3,21729140584165	3,22272332761050	3,20428164606908	3,19664390634432	3,19146979000034
Total	12,64308473071980	12,55572573725960	12,68463594647650	12,53095050677660	12,46799998232650	12,71535182933570	12,41771174456060	12,84004104855450	12,69931632926590	12,51910384472250	12,60739216999980

	Obstacle avoidance iteration step duration (s)										
Step number	1	2	3	4	5	6	7	8	9	10	Average (s)
1	2,92773361580245	2,91110467931075	2,96638342990662	3,10119230285820	2,95544538627956	2,97438945462549	3,03347437899765	3,10363120919987	3,06188361614029	3,01955590686779	3,00547939799887
2	3,07251446615157	2,93080542380016	2,94952020211958	3,06515811622795	3,01039730669425	2,90108359935806	2,97848301325023	3,01012151503297	2,92512713416361	3,03640782919522	2,98796186059936
3	3,07067137247644	3,13098456695383	3,09224951840494	3,02601842033456	3,10293639832414	3,00636893203703	3,03804534373495	3,01189847769020	3,13829903171400	3,03317553833211	3,06506476000022
4	3,04324030635958	3,01821955177260	2,95985655493610	2,99143880442740	3,06496232498548	2,98376694252590	3,04666534271389	2,98575744442198	3,08117776755963	2,93600683543195	3,01110918751345
Total	12,11415976079000	11,99111422183730	11,96800970536720	12,18380764384810	12,13374141628340	11,86560892854650	12,09666807869670	12,11140864634500	12,20648754957750	12,02514610982710	12,06961520611190

Figure A.10: All measured times of the execution of the different steps of the different optimizations tested.

A.8. Maintenance

This section provides the graphs generated by the maintenance determination for the execution of the test path. It also provides with the comparison alongside the graphs generated by the RoboCharts add-in.

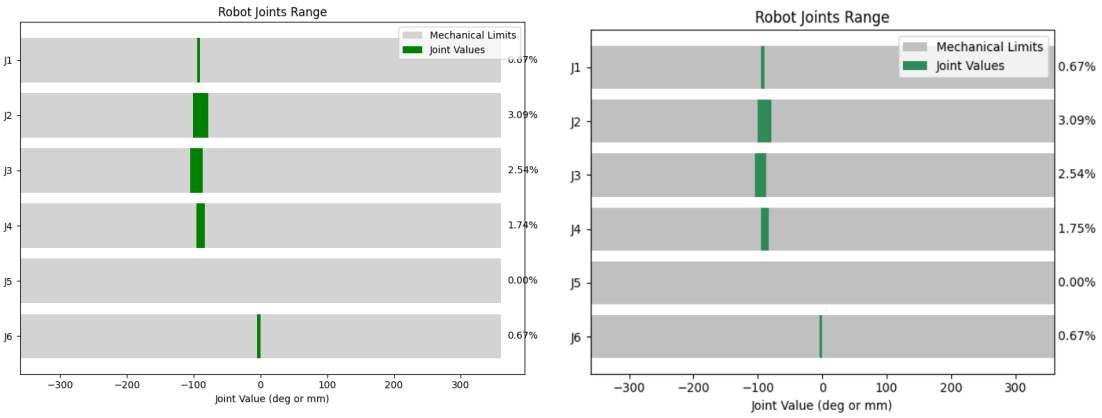


Figure A.11: Robot joint range comparison between the graphs generated by the maintenance determination (left) and the RoboCharts add-in (right).

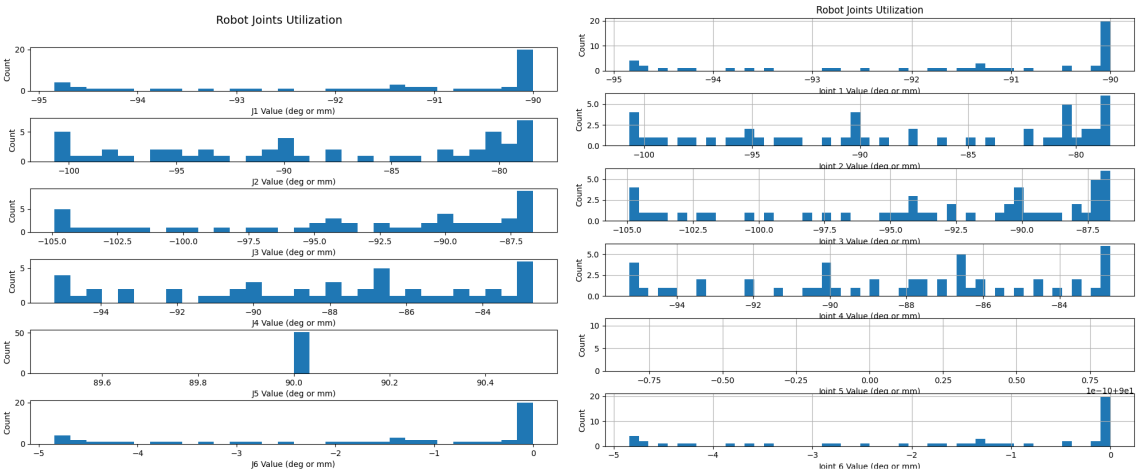


Figure A.12: Robot joint utilization comparison between the graphs generated by the maintenance determination (left) and the RoboCharts add-in (right).

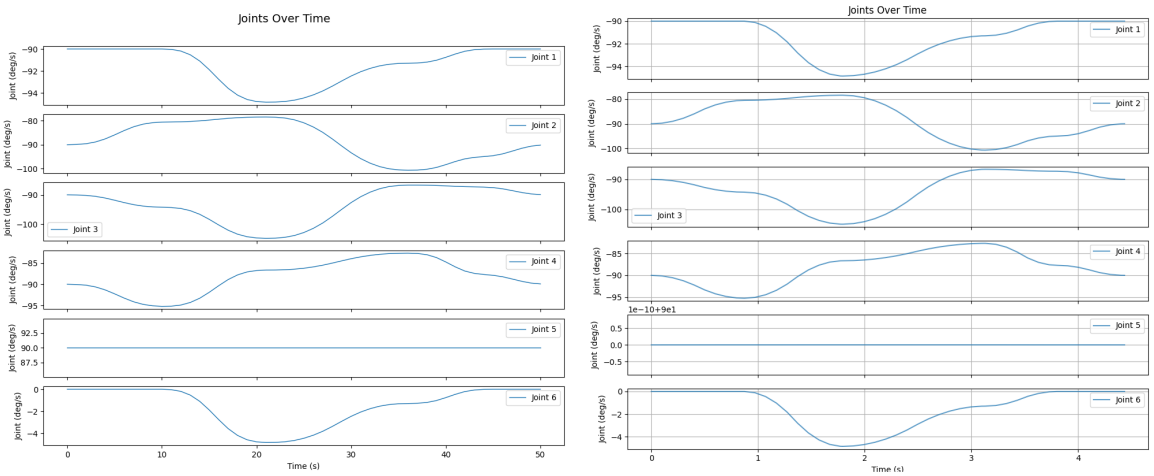


Figure A.13: Robot joints over time comparison between the graphs generated by the maintenance determination (left) and the RoboCharts add-in (right).

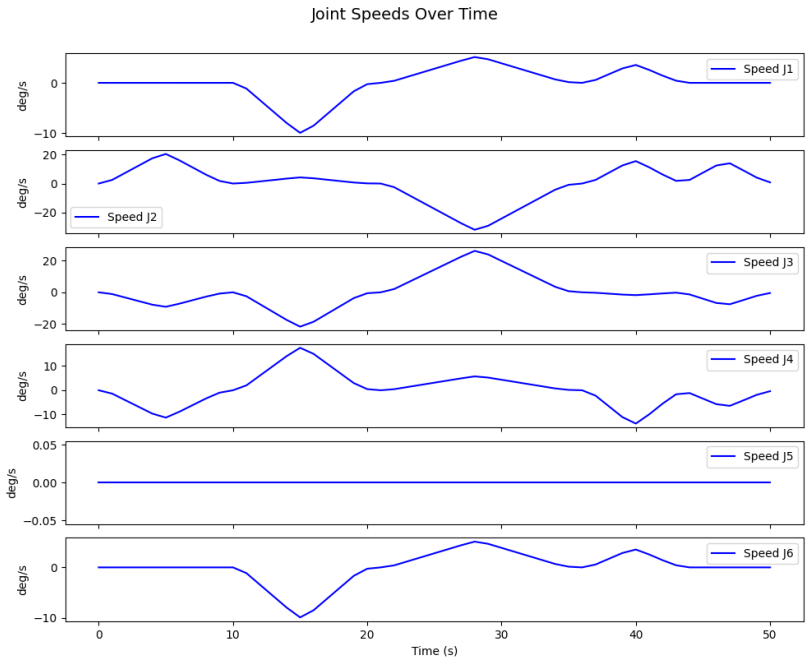


Figure A.14: Robot joint speeds over time graph generated by the maintenance determination.

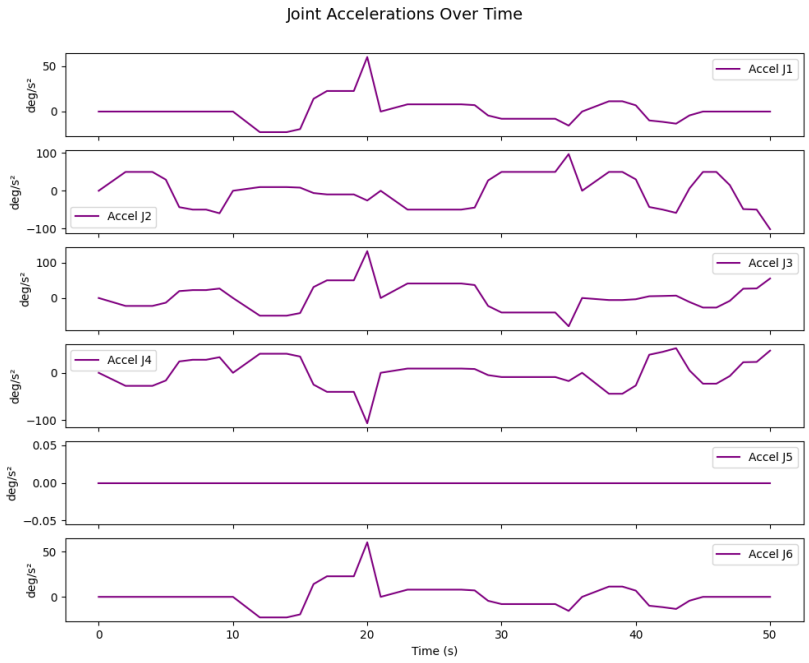


Figure A.15: Robot joint accelerations over time graph generated by the maintenance determination.

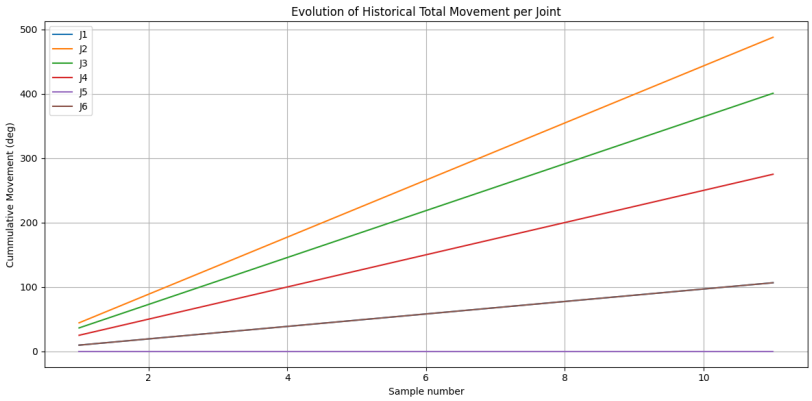


Figure A.16: Graph displaying historical robot joint cumulative movement over the samples taken, generated by the maintenance determination.

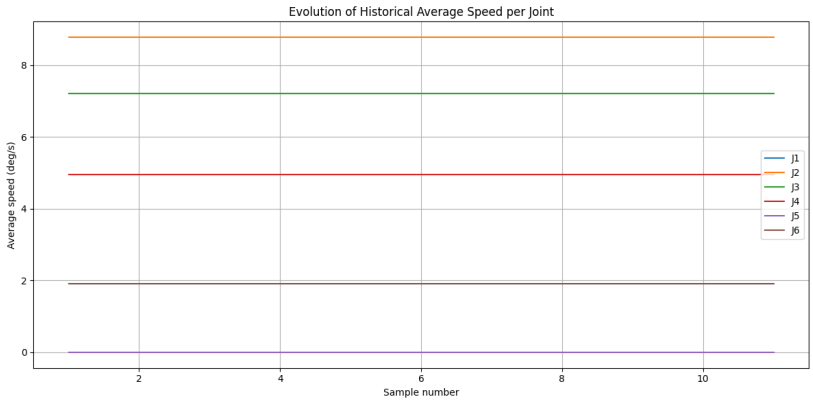


Figure A.17: Graph displaying historical robot joint speeds over samples taken, generated by the maintenance determination.

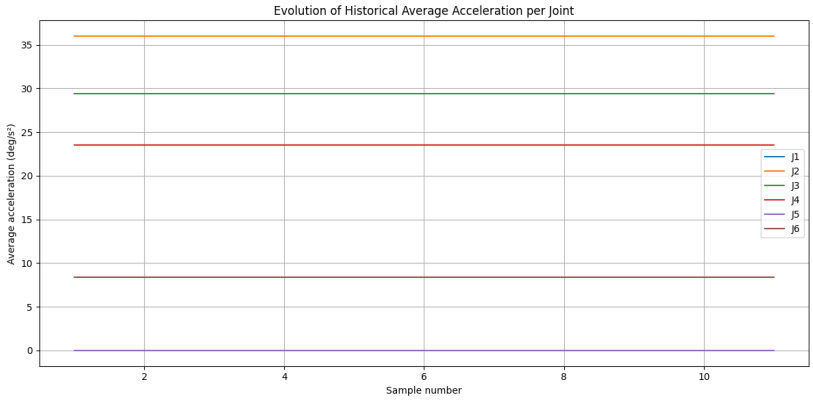
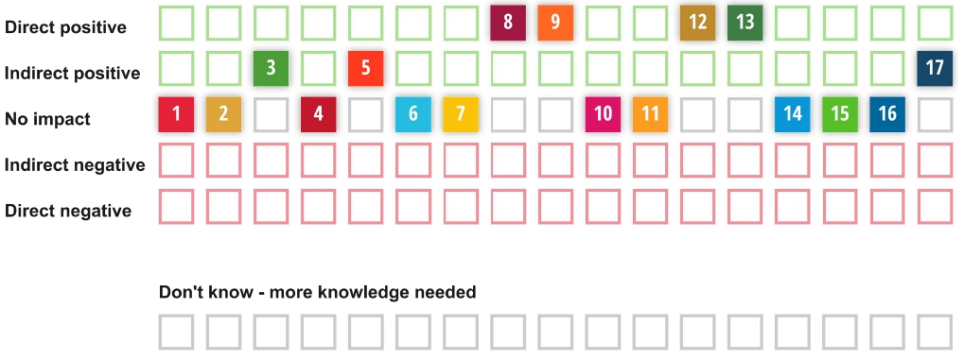


Figure A.18: Graph displaying historical robot joint accelerations over samples taken, generated by the maintenance determination.

A.9. Sustainable Development

SDG IMPACT ASSESSMENT TOOL

Connected digital twin for robotic battery assembly



Description

The project aims to create and implement a real-time connected digital twin for robot battery assembly. By leveraging the potential of digital twin technology, the project aims to optimize and automate battery cell assembly and disassembly with the help of a UR10e robot. Among the major aims of this project is enhancing efficiency, reducing energy consumption, and enhancing safety in the industrial environment. The digital twin will provide predictive maintenance, real-time monitoring, and process optimization with potential advantages for the manufacturing process and the environment.

Strategic choices

These are the prioritised areas that we will take action on.

- [x] Positive impacts we can strengthen even further
- [ ] Negative impacts we can eliminate or minimise
- [ ] Knowledge gaps we need to fill

NO POVERTY

End poverty in all its forms everywhere

Impact

NO IMPACT

Motivation

The digital twin for robotic battery assembly helps increase reuse, but does not combat poverty.

ZERO HUNGER

End hunger, achieve food security and improved nutrition and promote sustainable agriculture

Impact

NO IMPACT

Motivation

the digital twin for robotic battery assembly does not influence anything in agriculture and food production.

GOOD HEALTH AND WELL-BEING

Ensure healthy lives and promote well-being for all at all ages

Impact

INDIRECT POSITIVE

Motivation

Digital twins allow simulating new and potentially dangerous situations in which the user could be injured in real life, avoiding harm.

QUALITY EDUCATION

Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all

Impact

NO IMPACT

Motivation

It has an easy-to-use interface that is inclusive for everyone; however, it does not teach the user how to create a digital twin.

GENDER EQUALITY

Achieve gender equality and empower all women and girls

Impact

INDIRECT POSITIVE

Motivation

Process is apliable to both genders

CLEAN WATER AND SANITATION

Ensure availability and sustainable management of water and sanitation for all

Impact

NO IMPACT

Motivation

Digital twins are found in the virtual world, they cannot guarantee water

AFFORDABLE AND CLEAN ENERGY

Ensure access to affordable, reliable, sustainable and modern energy for all

Impact

NO IMPACT

Motivation

It reduces energy consumption, but it does not make it more accessible

DECENT WORK AND ECONOMIC GROWTH

Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

Impact

DIRECT POSITIVE

Motivation

Digital twins create a virtual copy of an object or process, thus allowing the modeling and simulation of production processes to identify opportunities to reduce waste and emissions.

INDUSTRY, INNOVATION AND INFRASTRUCTURE

Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

Impact

DIRECT POSITIVE

Motivation

Allows the assembly and disassembly of objects in the most optimal way after having made attempts in the simulation, thus helping to reduce waste.

REDUCED INEQUALITIES

Reduce inequality within and among countries

Impact

NO IMPACT

Motivation

It does not reduce inequality between countries since to create a digital twin you must have your own resources

SUSTAINABLE CITIES AND COMMUNITIES

Make cities and human settlements inclusive, safe, resilient and sustainable

Impact

NO IMPACT

Motivation

The digital twin for battery assembly does not make cities safer

RESPONSIBLE PRODUCTION -- AND CONSUMPTION

Ensure sustainable consumption and production patterns

Impact

DIRECT POSITIVE

Motivation

It reduces the use of resources by simulating the process beforehand before carrying it out, and it also allows for searching for different strategies to reduce and manage waste. Likewise, it allows disassembling objects, promoting reuse.

CLIMATE ACTION

Take urgent action to combat climate change and its impacts

Impact

DIRECT POSITIVE

Motivation

The reuse of objects, as well as the ability of digital twins to identify inefficiencies and methods to reduce waste, help with climate change, as well as the reduction of energy by using the physical robot less by conducting tests directly on the virtual robot.

LIFE BELOW WATER

Conserve and sustainably use the oceans, seas and marine resources for sustainable development

Impact

NO IMPACT

Motivation

Digital twins are found in the virtual world, they cannot preserve the oceans

LIFE ON LAND

Protect, restore and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss

Impact

NO IMPACT

Motivation

A digital twin could simulate a process to find a way to protect ecosystems; however, in this project, being focused on a robotic cell, it does not intervene in the protection of ecosystems.

PEACE, JUSTICE AND STRONG -- INSTITUTIONS

Promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels

Impact

NO IMPACT

Motivation

Our interface is indeed accessible to all audiences, but it does not promote peaceful societies.

PARTNERSHIPS FOR THE GOALS

Strengthen the means of implementation and revitalize the global partnership for sustainable development

Impact

INDIRECT POSITIVE

Motivation

The increase of digital twins in the industry helps reduce energy and water consumption, and encourages reuse.