POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's Degree Course in Mechatronic Engineering

Master Thesis

Design and evaluation of different digital control technologies of a mobile manipulator

Supervisor:
Prof. Dario Antonelli

Candidate:
Giuseppe La Spisa

Academic Year 2022-2023
July 2023

CONTENTS

ABSTRACT

The term Industry 4.0 indicates the fourth industrial revolution. It consists of a production and business management model based on the interconnection between physical and digital devices. In particular, the manufacturing sector is experiencing a new era of development powered by a series of emerging high technologies such as human-robot collaboration, the Industrial Internet of Things (IIoT), cyber-physical systems, augmented reality, machine learning, and artificial intelligence. These features are necessary to analyse, interpret, and predict the real behaviour of machines and devices used in the manufacturing sector, such as autonomous robots and automated guided vehicles (AGVs). One of the main benefits of the digital transition driven by Industry 4.0 is the real-time accessibility of comprehensive information from devices involved in the manufacturing processes, acquired by integrated sensors embedded in the machines and camera systems that monitor the shop floor, and transmitted through the internet network. As a result, these data can be stored for further processing and readily accessed and utilized to build a virtual environment as faithful as possible to the real one. In recent years, the concept of Digital Twin (DT) has gained significant importance in this field of research. It was created to operate in parallel and interact with real-world physical production equipment, enabling a bidirectional data transmission. Indeed, the DT can receive data from the factory floor and can also transmit data back to trigger specific actions in the manufacturing process. The DT can be used for a variety of purposes, including remote supervision, condition monitoring, and testing and validation. The advantages of this technology are:

- The possibility to analyse hypothetical scenarios and plan future changes, thus reducing implementation time and costs.
- The anticipation of potential problems that may arise in the future.
- The reduction of accidents since it can simulate all kinds of situations.
- The increase of productivity and optimization of costs.
- Avoiding the risks of damages and possible harmful actions for humans.

In addition to the Digital Twin concept, mention should be made of the Digital Model (DM) and the Digital Shadow (DS), other virtual simulation technologies used especially for the static analysis and real-time monitoring of the physical working environment. The main difference is the data transmission modality. Indeed, the Digital Model does not use any form of automated data exchange, whereas the Digital Shadow allows only one-way data flow. This means that a change in the states of a real physical object leads to a modification in the digital one, but not the opposite situation.

The objective of this thesis is twofold. First, to develop a DM of a Mobile Manipulator (MoMa) pick and place application, assessing its accuracy by comparing the simulation results with the reference ones obtained by means of Optitrack Motive, a real-time 3D tracking camera system. Second, the attempt to implement a DT, employing the ROS environment to acquire real-time data from the physical objects connected to the same network as the ROS machine. In this way, it is possible to monitor the movements of the robots in real-time, simulate possible scenarios, and communicate with the physical world in both directions. Finally, the Matlab ROS Toolbox provides the ability to acquire the data directly in this simulation workspace and, through Simulink, implement real-time control operations.

1 Introduction

1.1 Industry 4.0 and the concept of digital twins in manufacturing

The concept of Industry 4.0, also referred to as the Fourth Industrial Revolution, was initially introduced in 2011 during the Hannover Messe through a strategic programme initiated by the German government. [1]. In 2016, Klaus Schwab provided definitions and analysed the impact of technological advancements on the way of living and working in his book titled 'The Fourth Industrial Revolution', He described it as a revolution that "creates a world in which virtual and physical systems of manufacturing globally cooperate with each other in a flexible way" [2].

The integration of digital technologies into industrial processes has given rise to the concept of 'smart factories'. They exhibit high flexibility and adaptability to cope with rapid and dynamic changes in production processes within an increasingly complex environment [3]. Indeed, Industry 4.0 has significantly improved manufacturing efficiency by enabling interconnections between industrial equipment through the Internet of Things (IoT), cyber physical systems, cloud computing, machine learning, and artificial intelligence. These technologies are closely associated with the concept of predictive maintenance, which allows for the prediction of equipment health by analysing acquired data through appropriate algorithms. This approach minimizes the impact of unexpected events, breakdowns, or anomalies, leading to cost reduction, decreased downtime, and extended lifetime of industrial assets [4], [5].

The main key features and technologies, illustrated in *Figure 2* include:

- **Industrial Internet of Things (IIoT)**: IIoT involves a network of interconnected physical devices, machines, sensors, and other industrial assets. This machine-oriented technology enables the real-time data exchange, leading to optimal industrial processes [6].
- **Big Data Analytics**: The massive data sets generated from the entire manufacturing chain in Industry 4.0 need to be acquired, processed, stored, and extensively analysed to improve factory operations and product quality, reduce machine downtime, and enhance efficiency [7].
- **Cloud computing**: The NIST [8] defines cloud computing as: "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Cloud computing aims to deliver computing services that enhance the reliability, flexibility, and accessibility of manufacturing systems [9].
- **Artificial intelligence (AI) and Machine Learning (ML)**: AI and ML are powerful tools capable of learning from past or historical data and consequently developing intelligent predictive algorithms. In the manufacturing industry, these techniques find application in various fields such as predictive maintenance, optimization, troubleshooting, and control [10].
- **Cyber-Physical Systems (CPS)**: CPS are characterized as physical systems incorporating sensing, computation, and control capabilities. In the context of Industry 4.0, CPS and IIoT ensure seamless production processes and enable efficient data analysis [11].
- **Robotics and Automation**: Robots play a fundamental role in Industry 4.0, as they are becoming progressively more autonomous and intelligent. They are developing the capability to collaborate alongside humans and interact with one another [12].
- **Cybersecurity**: In such an interconnected manufacturing environment, the importance of safeguarding connected CPS and manufacturing lines from cybersecurity threats significantly increases. Consequently, ensuring secure and reliable communications, along with

implementing sophisticated identity and access management for both machines and users, becomes indispensable [13].

- **Additive Manufacturing (AM)**: 3D printing is an AM technique that proves highly valuable in producing a wide variety of customized products with complex geometries, all at relatively low costs [14].
- **Augmented Reality (AR) and Virtual Reality (VR)**: AR and VR are two interactive technologies. The first enriches the real-world with virtual elements, while the second creates fully digital simulation environments. Both technologies are significant resource in manufacturing, finding applications in various areas such as simulation, worker training, maintenance, assembly operations, and supervisory control [15].
- **Simulation**: In manufacturing, simulation plays a crucial role by utilizing real-time data to create accurate 2D or 3D virtual representations of processes and enabling behaviour monitoring. It allows for the evaluation of multiple scenarios across the product chain, facilitates real-time performance analysis, and empowers companies to identify and optimize processes while minimizing costs and risks [16].

To provide a more comprehensive explanation of the simulation concept in Industry 4.0, the introduction of three additional technologies is necessary: Digital Models (DM), Digital Shadows (DS), and Digital Twins (DT). These technologies exemplify some of the benefits of digital transformation, which is rapidly revolutionising the entire manufacturing industry. Manufacturers are increasingly adopting digital solutions to streamline operations, optimize and predict the behaviour of the production system, and real-time data elaborating. While the central concept revolves around the virtual representation of a physical process, it is crucial to note that DM, DS, and DT differ significantly in the level of data integration between their physical and digital counterparts [17]:

- **Digital Model**: A DM does not allow for automated and real-time data exchange between the physical environment and the digital one. As a result, any modifications in the real-world have no impact on the digital world, and vice versa.
  In summary, a DM is a virtual representation used for design and static analysis purposes.
- **Digital Shadow**: A DS is characterized by an automated flow of information from the real-world to the digital one. However, in the reverse direction, this process is only executed manually. Consequently, any changes made in the real environment are reflected in real-time in the digital image, but not the other way around.
  In summary, a DS provides real-time monitoring and analysis of physical assets.
- **Digital Twin**: A DT bridges the gap between the physical environment and the digital one, synchronizing them and enabling automatic dynamic data exchange in both directions.
  Any modification made in either of the two environments is visible in its corresponding counterpart.
  In summary, a DT offers a more comprehensive dynamic model with simulation and predictive capabilities for optimization and experimentation.

Figure 1 illustrates a summary of the data flow within these various digital technologies.

Digital twins also find applications in the field of process planning [18], where advancements in sensorization and connectivity have greatly enhanced the efficiency of transportation [19],[20]. Through embedded intelligence in Automated Guided Vehicles (AGVs) and in their successors, Autonomous Mobile Robots (AMRs), autonomous navigation enables improved operability within manufacturing plants [21]. The integration of AMRs into a collaborative environment, facilitated by

their connectivity, allows for real-time optimization of fleet routes and coordination with other machines, resulting in enhanced trajectory optimization and goods traceability. Additionally, AMRs exhibit significantly lower accident rates compared to conventional transport, contributing to improved occupational safety. Furthermore, the incorporation of new technologies in AMRs offers greater flexibility, enabling dynamic adaptation to new situations without human intervention [22].
The development of DT for virtual analysis of AMR behaviour aims to enable the identification of potential issues, achieve real-time control over the robot's actions, provide the ability to send commands, store data, and monitor motor and battery status [23].
Similarly to AGVs and AMRs, the implementation of a comprehensive digital twin for collaborative robots brings numerous advantages in terms of human-robot collaboration [24].

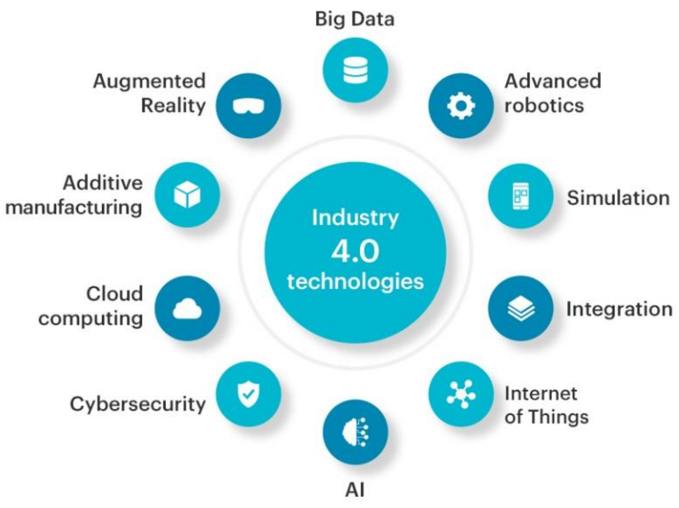


Figure 2: Technological pillars of Industry 4.0.

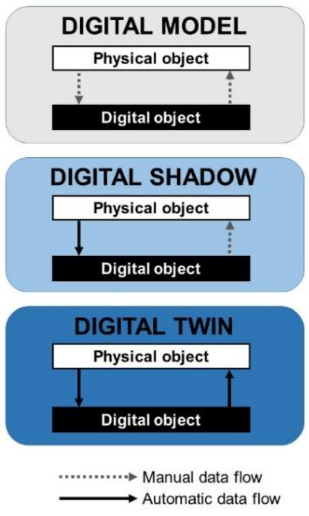


Figure 1: Data flow in Digital Model, Digital Shadow, and Digital Twin.

1.2 OBJECTIVES OF THE THESIS

The objective of this thesis is to implement a comprehensive digital environment for a Mobile Manipulator (MoMa). The main goal is to leverage the benefits of Industry 4.0, as discussed in the previous section, to create a highly accurate virtual model capable of monitoring and analysing data from the physical object. This virtual model will also enable simulation of various scenarios and establish two-way communication between the real and digital worlds. The thesis proposes the digitalization of the mobile manipulator from multiple perspectives, utilizing three distinct simulation technologies described earlier, starting from a DM and progressing to a DT. Each of them is discussed and evaluated independently, employing various software and data acquisition systems to assess its performance. The involved software and tools are:

- Excel
- SolidWorks
- Matlab, Simulink, and Simscape
- Node-RED
- Optitrack Motive
- Ubuntu Virtual Machine including ROS and Rviz

These will be described in more detail in the following chapters, explaining their features and implications.

In summary, the steps required to achieve the objective can be divided as follows:

- To accurately calibrate the Optitrack Motive system for real-time tracking of the MoMa's precise position.
- To assess the position error of the mobile robot by executing a given trajectory multiple times and comparing the results with those obtained from Optitrack.
- To analyse the mobile robot's accuracy in reaching known and fixed positions while encountering obstacles.
- To develop a 3D Digital Model of the MoMa that faithfully replicates the real-world processes.
- To create a Digital Shadow of the MoMa, treating the mobile robot and the collaborative robot as separate objects and leveraging real-time information from both.
- To realize a Digital Twin of the MoMa, enabling a bidirectional real-time data exchange with low latency, and seamless integration with different simulation systems to enhance control capabilities and expand applications.

The analysis is specifically related to tasks performed in the restricted environment of the Mind4Lab laboratory at Politecnico di Torino. However, it represents a dynamic scenario, comparable with a good approximation, to an industrial environment, in which people and machines coexist and interact with each other, constituting possible obstacles to the movement of the MoMa.
Within the laboratory, several research projects have been conducted on the mobile manipulator. However, the existing DM has certain limitations in terms of speed and accuracy when providing results [28]. Therefore, the implementations in this thesis aim to enhance the previous DM and go beyond by laying the foundation for a Digital Twin that can serve as a basis for future research objectives.

1.3 MOBILE MANIPULATOR (MOMA)

The MoMa used for the experiments described in this thesis consists of an Autonomous Mobile Robots (AMRs), model MiR100, by Mobile Industrial Robot, and a collaborative robot, model UR3, by Universal Robot, with an RG2 gripper, provided by OnRobot, attached to the end effector.

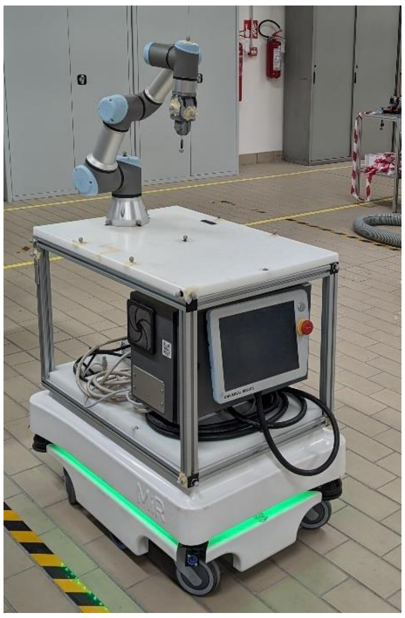


Figure 3: Mobile Manipulator (MoMa)

1.3.1 MiR100

It is extensively utilized in manufacturing to automate internal transportation and logistics processes. Its versatility is enhanced by the ability to customize it with various top modules, including bins, racks, lifts, conveyors, and even a collaborative robot arm, as demonstrated in this project. By leveraging embedded sensors, laser scanners, and 3D cameras, the MiR100 can accurately map its surroundings and continuously update this map in real-time using sensor data. This feature is crucial for identifying obstacles and inaccessible areas. As a result, the robot can autonomously navigate within the designated map, employing planning algorithms to identify the fastest and safest local path, while effectively avoiding obstacles that may hinder its progress (*Figure 6)*. The MiR100 is equipped with six wheels: two drive wheels positioned in the middle and four caster wheels at the corners.
It has a maximum payload capacity of 100kg and can reach a maximum linear speed of 1.5m/s.[1]

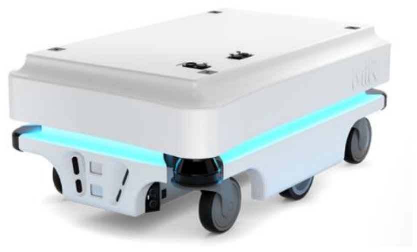


Figure 4: Mobile Industrial Robot MiR100.

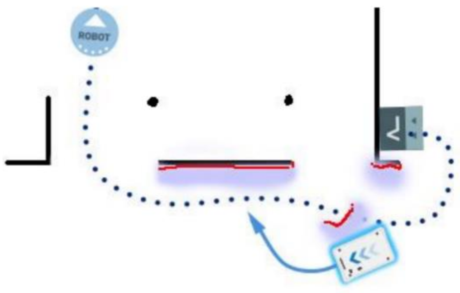


Figure 5: The global path is indicated with the dotted blue line and is visible on the map. The local path is indicated with the blue arrow, showing the robot driving around a dynamic obstacle.

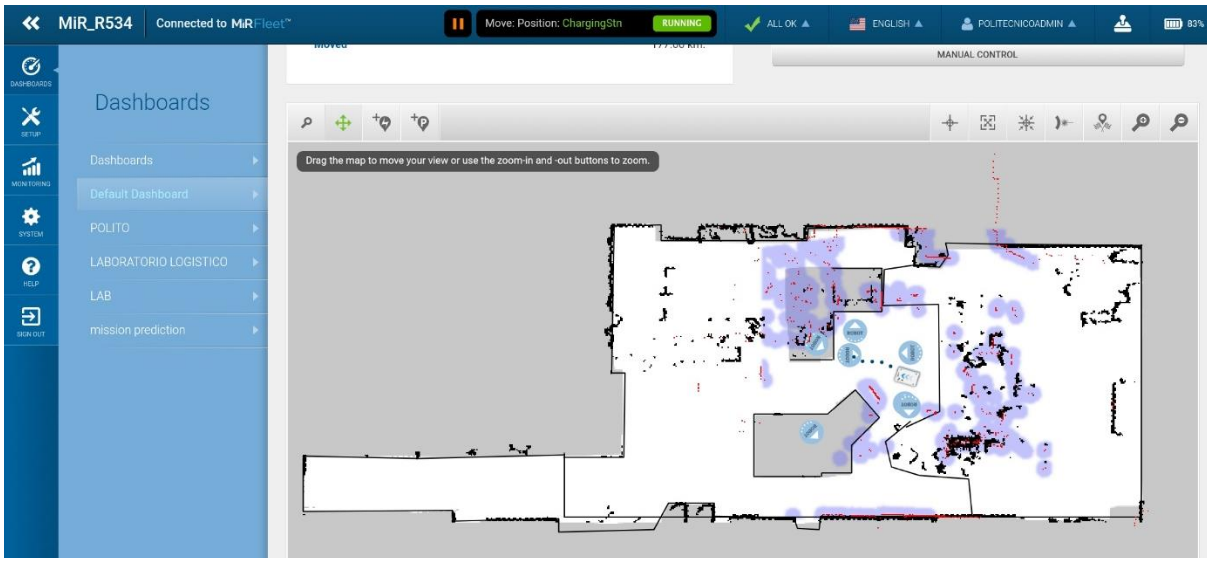


Figure 6: Laboratory live map on the MiR software.

1.3.2 UR3 and RG2

UR3 is the most compact collaborative robot in the UR catalogue, and it belongs to the CB3-series. This cobot is characterized by six degrees of freedom (six rotating joints, ±360° range), an infinite rotation on the wrist, a maximum extension from the base joint equal to 500mm, and a force/torque control action, provided by the sensor on the sixth axis. The UR3 is applied in a wide range of tasks, including assembly with a payload capacity of up to 3kg, as well as activities like glue dispensing,

[1] Mobile Industrial Robot official website: MiR100 (mobile-industrial-robots.com)

screw mounting and tightening, and pick and place operations[2]. The UR robot's programming is accomplished through the utilization of the teach pendant equipped with the Polyscope graphical user interface. The teach pendant allows for the selection of different actions to create a mission for the robot. Actions such as wait, move, and set can be employed. The move command offers three types of movements: moveL for linear motion, moveJ for non-linear and unpredictable motion, and moveP for circular motion at a constant velocity. Once the desired motion mode is chosen, a set of waypoints is defined to construct the trajectory. Waypoints can be established by specifying precise joints configuration or by manually moving the robot or utilizing the teach pendant motion commands, and then saving the current position as a new waypoint. In addition, UR collaborative robots are designed to seamlessly interface with all products in the OnRobot series. In the context of this thesis, which focuses on a pick and place task, the RG2 gripper is mounted at the end of the robot arm. By installing the OnRobot plugin on the UR teach pendant, the gripper status can be programmed within the same graphical Polyscope software.

Figure 8: Universal Robot UR3, CB3-series.

Figure 7: OnRobot RG2 gripper.

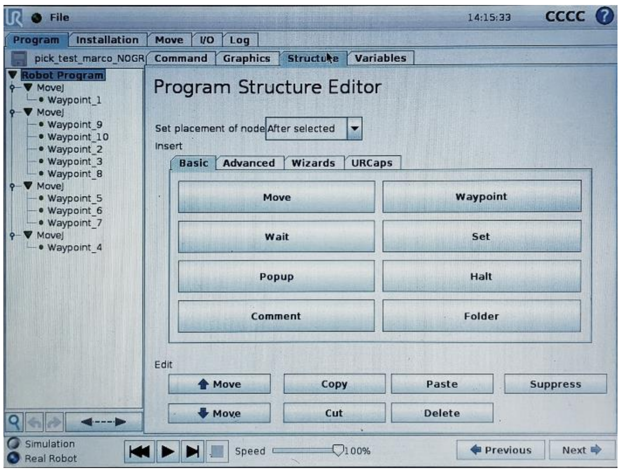


Figure 9: Polyscope UR available program options.

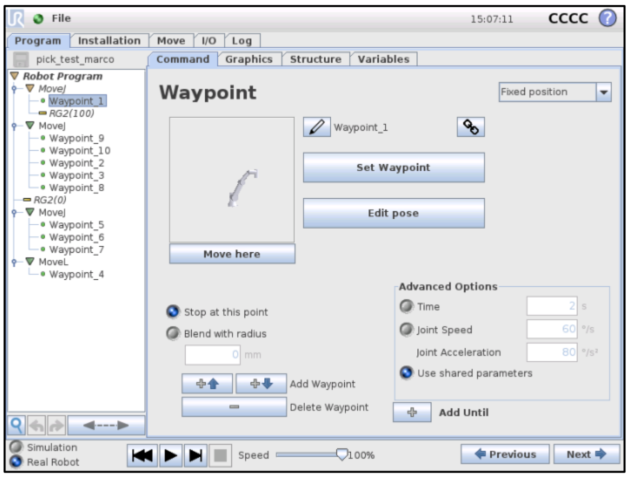


Figure 10: Example of pick and place program including RG2.

1.3.3 MiR100 and UR3 communication

Once the UR3 is mounted on MiR100 and powered by it, the UR feature needs to be enabled and the UR3's IP address has to be specified in the system settings section using the MiR web interface. This is feasible since the MiR robot is designed to facilitate the creation of a MoMa by integrating a UR cobot. As a result, within the mission section, there is an option to execute a UR program, along with the ability to move the MiR to a desired location on the map or modify its orientation.

[2] Universal Robot official website and UR3 datasheet: Mød CB3-familien (universal-robots.com)

2 ACQUISITION SYSTEMS

2.1 OPTITRACK MOTIVE

Motive is the best motion tracking software currently available. It utilizes real-time data captured by Optitrack cameras (see *Figure 13*), which accurately detect the position and orientation coordinates of specific markers placed on the object of interest. In Motive, a minimum of three markers is required to define a Rigid Body in 3D space, similar to how three coordinates define a plane. The presence of additional markers enhances overall tracking stability and reduces susceptibility to disturbances.
The Optitrack Motive specifications provide the following accuracy and latency parameters for tracking the position of a rigid body:

- Positional accuracies typically $\pm 0.2mm$ or less
- Rotational accuracies typically $\pm 0.1deg$ or less
- Latency $< 10ms$

The specified values are the ideal values provided by the official website[3]. However, the actual accuracy parameters depend on the calibration of the entire system, which, in turn, relies on factors such as camera conditions and environmental lighting. To calibrate the system, it is necessary to slowly move a dedicated wand with three calibration markers positioned on it until their position is accurately detected by all cameras. For the development of the tasks in this thesis, a calibration with a mean error (ME) of 0.531mm is utilized. This value currently represents the minimum achievable within the laboratory context where the simulations and analyses are conducted.
The level of accuracy of the measurements reported in the development of this thesis is calculated by considering the following observations and calculations:

- Uniform distribution
- Type B Evaluation of Standard Uncertainty (u)
- Expanded uncertainty (U) estimated by assuming a coverage factor k=2 at a confidence level of 95%, in according to ISO GUM[4]

Optitrack measurements model is given by: $\gamma = \mu \pm U$, where γ is the unknown value of the measurand, taking into account the disturbances, μ is the equally weighted mean, and $U = k \cdot u$ [25].

$$\gamma = \mu \pm U = \mu \pm ku = \mu \pm k\frac{ME}{\sqrt{3}} = \mu \pm 2\frac{0.531}{\sqrt{3}} = (\mu \pm 0.613)mm$$

The expanded uncertainty is $U = 0.613mm$. Due to the high amount of data obtained from the acquisition, a 95% confidence interval is selected to ensure a substantial number of measurements fall within the defined uncertainty limits.

Based on this result, Optitrack is considered the most accurate data acquisition system that faithfully represents the real-world. Therefore, it is utilized in this thesis as a reference model to compare the information obtained from other acquisition systems and evaluate the accuracy of the implemented digital models.

[3] Optitrack official website: OptiTrack - Motive - Specs
[4] ISO GUM: Guide to the expression of uncertainty in measurement - JCGM 100:2008 (GUM 1995 with minor corrections - Evaluation of measurement data (bipm.org)

2.1.1 Connecting Optitrack with Matlab

The data acquired from Optitrack can be integrated into new and existing applications, such as Matlab, allowing for the inclusion of custom plugins for real-time streaming in third-party applications[5]. NatNet's client/server architecture enables client applications to run on the same system as the tracking software (Motive), on separate system(s), or both. The SDK seamlessly integrates with standard APIs, tools, and protocols. By customizing the available Matlab scripts with relevant information (see Appendix 11.1.1), it becomes possible to manage all the desired data in real-time and store it for subsequent analysis.

Figure 11 displays the block diagram of the NatNet SDK network.

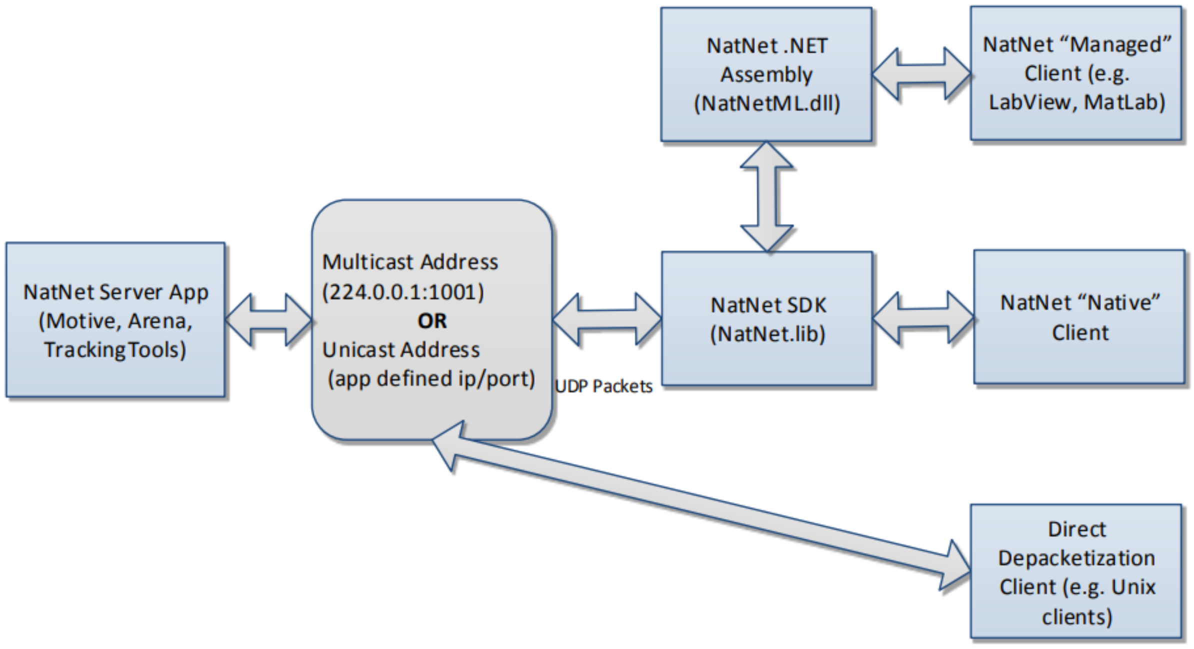

NatNet .NET Assembly (NatNetML.dll)
NatNet "Managed" Client (e.g. LabView, MatLab)
NatNet Server App (Motive, Arena, TrackingTools)
Multicast Address (224.0.0.1:1001) OR Unicast Address (app defined ip/port)
UDP Packets
NatNet SDK (NatNet.lib)
NatNet "Native" Client
Direct Depacketization Client (e.g. Unix clients)

Figure 11: NatNet SDK network block diagram.

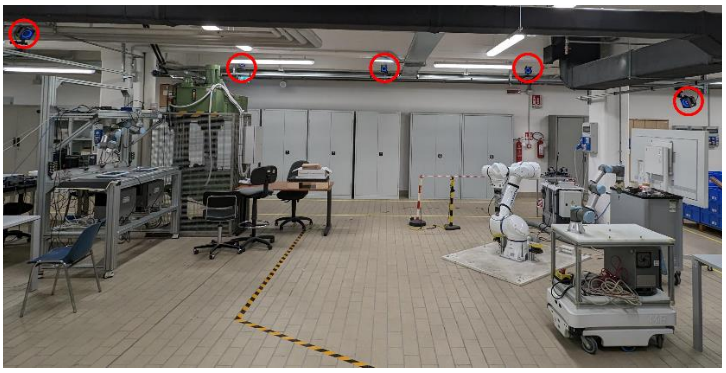


Figure 13: Mind4Lab simulation area with highlighted Optitrack cameras.

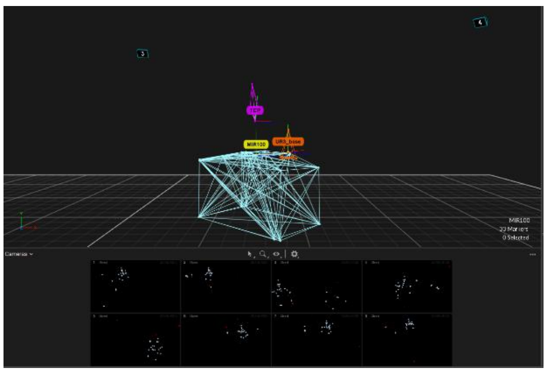


Figure 12: MoMa in Optitrack Motive system.

[5] NatNet API User's Guide: NatNet User's Guide (mocap.jp)

2.1.2 Important considerations

The real-time position data captured by Optitrack Motive represents the distances between the rigid body marker, which is set as pivot, and the Optitrack reference frame. These values are in millimeters. To compare this data with other acquisition systems and evaluate the accuracy of the MiR data, it is necessary to process the Optitrack data appropriately in Matlab. Specifically, the following procedure and calculation are performed.

Firstly, the MiR robot is moved to a known position (in yellow in *Figure 14*) with coordinates $(28.95; 21.00)m$. The MiR position is checked through ROS system developed in Simulink and described in subsection 2.3.1.
The Optitrack data captured at this location are: $z_{OPT} = -1958.5mm$ and $x_{OPT} = 1639.3mm$.
The z and x axis of Optitrack reference frame correspond respectively to x and y axis of the MiR reference frame, with the same orientations (see *Figure 14* for more details). The Optitrack area corresponds to that shown in *Figure 13*.
To obtain the Optitrack data comparable with the other acquisition systems, two offset values for both Optitrack coordinates are computed. These offsets represent the translation coordinates of the origin of the Optitrack reference frame with respect to the MiR one. The computations are shown below.

$$
\begin{aligned}
z_{offset} &= x_P + |z_{OPT}| = 28.95m + 1.9585m = 30.9085m \\
x_{offset} &= y_P - |x_{OPT}| = 21m - 1.6393m = 19.3607m
\end{aligned}
$$

Matlab script used to convert the Optitrack data is reported in Appendix 11.1.2.

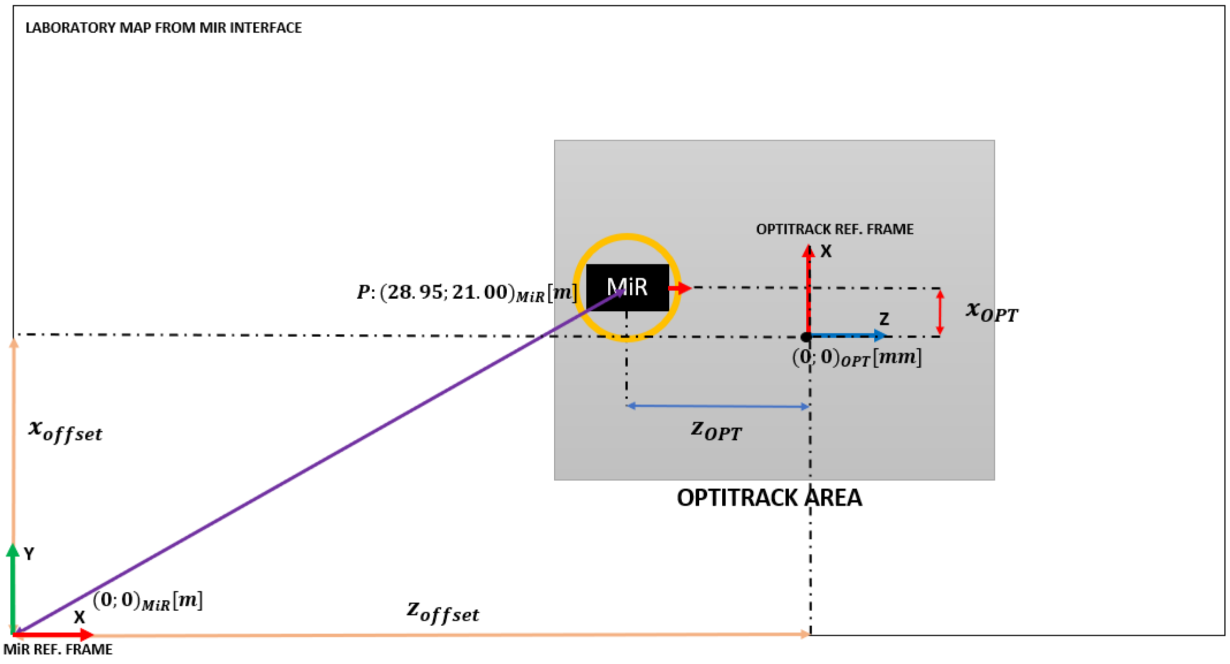


Figure 14: MiR and Optitrack reference frames.

2.2 Node-RED and Modbus/TCP protocol

Node-RED is a powerful programming tool designed to manage multiple IIoT devices in the context of Industry 4.0. With its intuitive browser-based interface, extensive library of nodes, and support for various protocols, Node-RED enables the seamless integration of physical devices, APIs, and online services[6]. It empowers users to effectively process and manage acquired data through JavaScript functions, while also facilitating data storage into files.
Modbus/TCP is a widely adopted industrial communication protocol that seamlessly combines the Modbus protocol with TCP/IP, providing robust connection-oriented communication. It ensures data integrity by effectively preventing errors at the destination. Additionally, the TCP protocol enables efficient concurrent communication among numerous devices, including sensors and actuators [26].

For the purposes of this thesis, two different connections are established in Node-RED, one with the MiR100 and another with the UR. The first one leverages the ROS bridge websocket server integrated in the mobile robot, employing a ROS subscriber node. The second one relies on the TCP protocol to establish communication with the universal robot, alongside the RTDE-interface node for data decoding. Through the UR connection, the information about joints position is obtained as well as data from the UR tool central point (UR_TCP). The reception of messages is accomplished using the dedicated Modbus nodes.
The implemented data acquisition schemes in Node-RED are illustrated in the figures below.

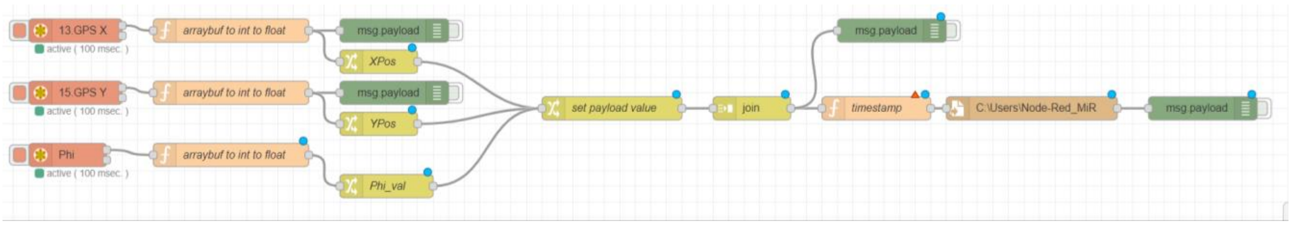


Figure 15: Node-RED scheme for MiR100 position data.

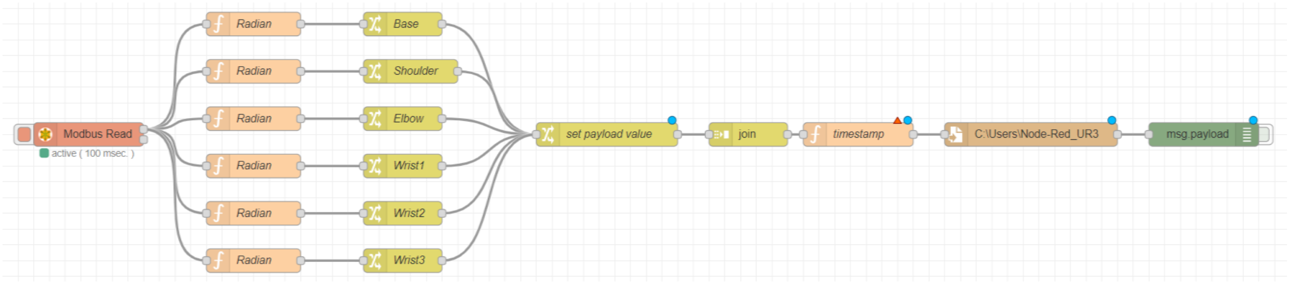


Figure 16: Node-RED scheme for UR3 joint states data.

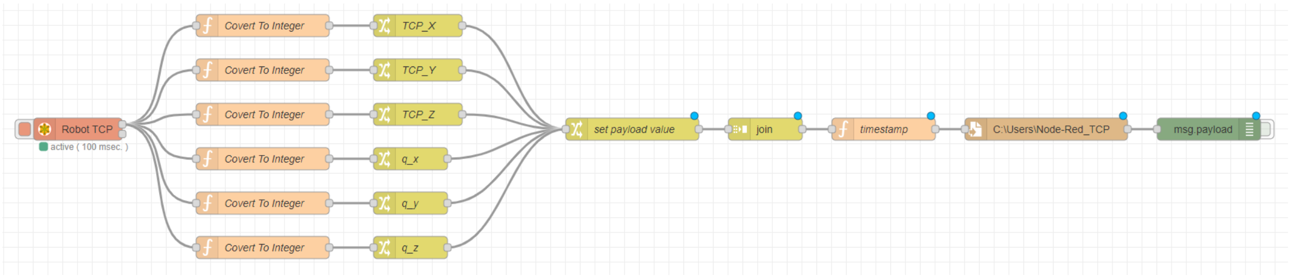


Figure 17: Node-RED scheme for UR3_TCP position data.

[6] Node-RED official website: Node-RED (nodered.org)

2.3 Robot Operating System (ROS)

ROS is an open-source environment that consists of a set of software libraries and tools used in robot applications. Communication within ROS is established through the principles of publishers and subscribers. Publishers are responsible for sending messages, while subscribers receive them. Messages are sent to topics, which are distinct from one another. Each topic can have multiple subscribers, and likewise, each node can publish to multiple topics. It is crucial for messages sent to a specific topic to comply with the corresponding data type [22]. The available topic list is obtained by running the command 'rostopic list' in ROS machine terminal.

In contrast to the Node-RED environment, ROS is installable on an Operating System, such as Linux or Ubuntu. It also provides the opportunity to utilize virtual machines, maintaining the hardware characteristics of physical devices. By leveraging ROS and Linux-based systems, users can seamlessly interface with multiple IoT devices, similar to Node-RED. Additionally, they can take advantage of a variety of tools and software to perform a multitude of tasks and actions related to connected devices. Linux OS provides access to 3D simulators like Gazebo and RVIZ, which can be connected to ROS through specific bridge nodes.
Gazebo is an advanced 3D simulator that enables the design and simulation of robot models and their workspaces, allowing for the execution of various scenarios with a high level of fidelity. It includes dynamic elements such as gravity, torque, friction, and a multitude of sensors [23].
RVIZ, on the other hand, is a powerful 3D ROS visualization tool that accurately reproduces the robot's perspective. It can process data from laser scanners and 3D depth cameras mounted on the AMR robot to detect obstacles and construct a 2D map within the robot's system. RVIZ also facilitates robot motion planning and pose estimation, allowing for real-time replication of these actions on the physical robot or within the Gazebo environment [27].
Both these software tools provide the capability to run robot models in Unified Robotics Description Format (URDF).

In addition, ROS enables the access to the GitHub repository, allowing users to enhance the Linux environment and add ROS packages and functions by leveraging the vast cloud library that contains projects and plugins developed by other users. The drivers used to establish a stable and suitable interface between MiR100[7], UR3[8], and ROS are obtained from the GitHub repository.

Based on these features, the ROS suite is considered one of the best systems for implementing comprehensive robot Digital Twins.

Real-time data acquired from robots can be stored in two ways. First, it can be directly saved to a bag file by executing the rosbag command on the ROS machine and specifying the desired topics. Alternatively, the rosbag command or setting can be used in Matlab and Simulink respectively (see subsection 2.3.1).
A fundamental aspect in achieving real-time data acquisition and control of the MiR is the time synchronization between it and the ROS machine. To ensure accurate TF transformations, simulations, and data control, it is essential to keep the maximum message transmission delay below 0.1s, as described in GitHub guide of the MiR driver. To meet this requirement, it is necessary to

[7] MiR100 driver for ROS: GitHub - DFKI-NI/mir_robot: ROS support for the MiR Robots. This is a community project to use the MiR Robots with ROS. It is not affiliated with Mobile Industrial Robots.
[8] UR driver for ROS: GitHub - UniversalRobots/Universal_Robots_ROS_Driver: Universal Robots ROS driver supporting CB3 and e-Series

update the time settings through the official web interface of the MiR by selecting 'Set device time on robot.' After completing this step, the connection between the mobile robot and the ROS system should be restarted. This can be done by executing the 'roscore' command on the Ubuntu command line, which initializes the machine hosting the ROS environment as the master. Additionally, execute the command 'roslaunch mir_driver mir.launch' to run the MiR driver and set the robot as the slave. The time delay parameters can be seen by running the command 'rosrun tf tf_monitor'.
It's important to note that both systems have to be connected to the same Wi-Fi network as the MiR. The results achieved for the development of this thesis are shown in Appendix 11.2.1.

Figure 18 shows an example diagram of the active nodes and topics utilized in the developed ROS environment for the MiR100 Digital Twin. This diagram is obtained by run the 'rqt_graph' in the Ubuntu command line. Rqt is a ROS software framework, which implements various GUI tools in the form of plugins.
In *Figure 18* the nodes from which the arrows originate represent the publishers, while the nodes at which they terminate correspond to the subscribers.

More details on the ROS environment developed for the purposes of this thesis are provided in the Digital Twin chapter (Chapter 7).

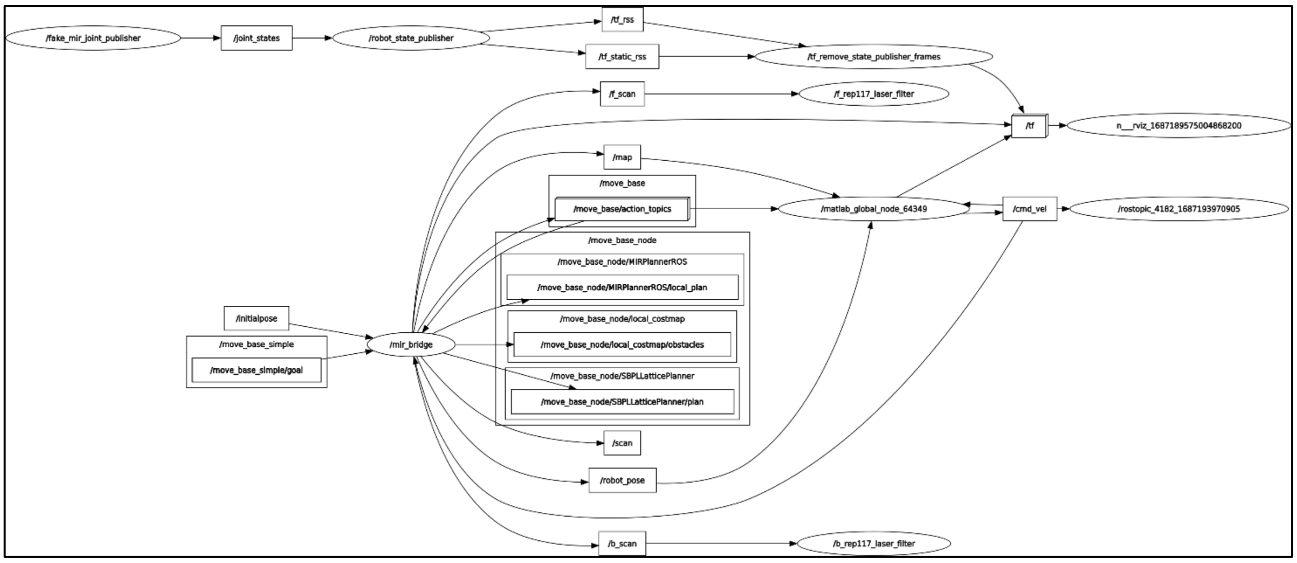

/fake_mir_joint_publisher
/joint_states
/robot_state_publisher
/tf_rss
/tf_static_rss
/tf_remove_state_publisher_frames
/f_scan
/f_rep117_laser_filter
/tf
n__rviz_1687189575004868200
/map
/move_base
/move_base/action_topics
/matlab_global_node_64349
/cmd_vel
/rostopic_4182_1687193970905
/move_base_node
/move_base_node/MIRPlannerROS
/move_base_node/MIRPlannerROS/local_plan
/initialpose
/move_base_node/local_costmap
/move_base_simple
/mir_bridge
/move_base_node/local_costmap/obstacles
/move_base_simple/goal
/move_base_node/SBPLLatticePlanner
/move_base_node/SBPLLatticePlanner/plan
/scan
/robot_pose
/b_scan
/b_rep117_laser_filter

Figure 18: Rqt diagram with active nodes and topics.

2.3.1 Connecting ROS with Matlab and Simulink

Matlab add-ons library provides the ROS Toolbox[9]. It acts as a bridge connecting Matlab and Simulink with the ROS. This toolbox offers a range of powerful Matlab functions and Simulink blocks that facilitate the visualization and analysis of ROS data by recording, importing, and utilizing rosbag files (see Matlab script in Appendix 11.2.4). ROS Toolbox also provides the capability to establish a connection with live ROS networks, enabling real-time access to ROS messages. This functionality is shown in *Figure 18*, where the Matlab node is subscribed to some MiR topics to receive information such as the current position and velocity of the robot. One remarkable feature of the toolbox is its ability to validate ROS nodes through desktop simulation and seamless integration with external robot simulators like Gazebo or hardware interfaces. Support for Simulink external mode allows for monitoring messages and modifying parameters while your model is running on hardware.

To enable ROS settings in Simulink, follow the steps below:

- Open Model Settings.
- In Hardware Implementation section, select ROS as the hardware board.
- Apply the changes.

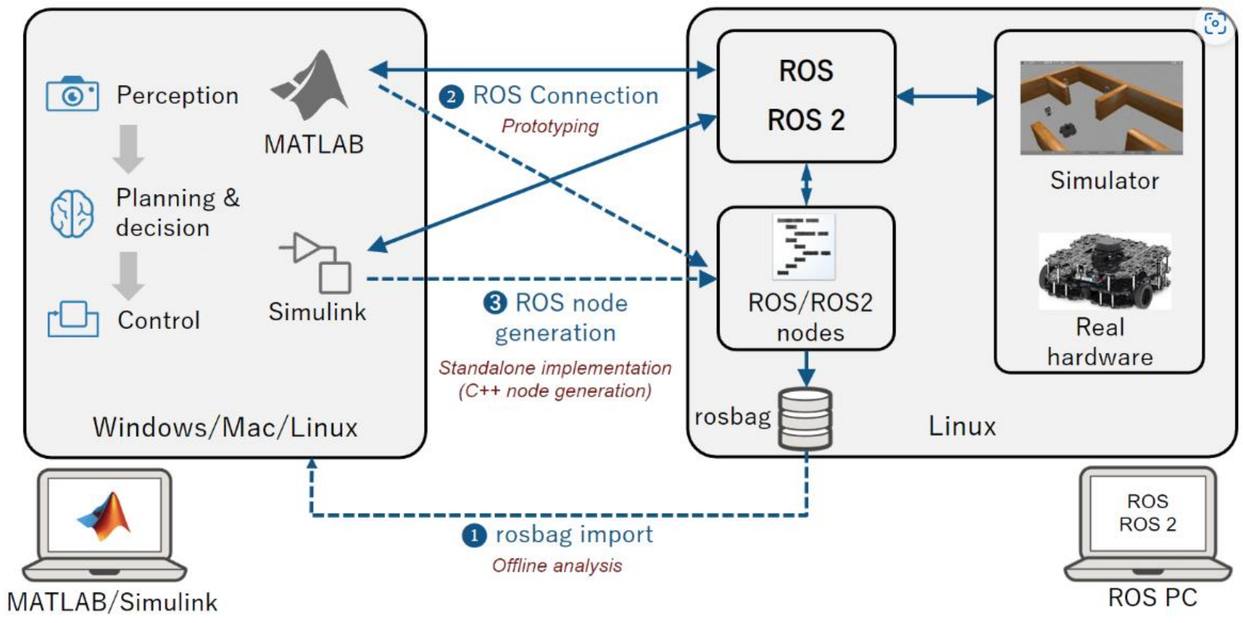


Figure 19: Summary diagram of communication between ROS devices and Matlab.

As mentioned in the previous section, it is also necessary to synchronize the timestamps between ROS devices in Matlab in order to compare the data recorded in bag files with data coming from other acquisition systems such as Node-RED and Optitrack, and to create temporal graphs for analysis.

Figure 20 displays a method for achieving time synchronization by utilizing Real-Time Sync block. Furthermore, the '/use_sim_time' parameter has to be set to true, through Get and Set parameter blocks.

[9] MATLAB ROS Toolbox documentation: ROS Toolbox Documentation - MathWorks Italia

Figure 21 and *Figure 22* depict the implemented Simulink model, which allows for the subscription of specific nodes to receive position and velocity messages from the MiR robot. Additionally, the model enables real-time data monitoring, by employing ‘display’ and ‘XY graph’ blocks.

The Matlab script to initialize the ROS network and Simulink functions are included respectively in Appendix 11.2.2 and 11.2.3.

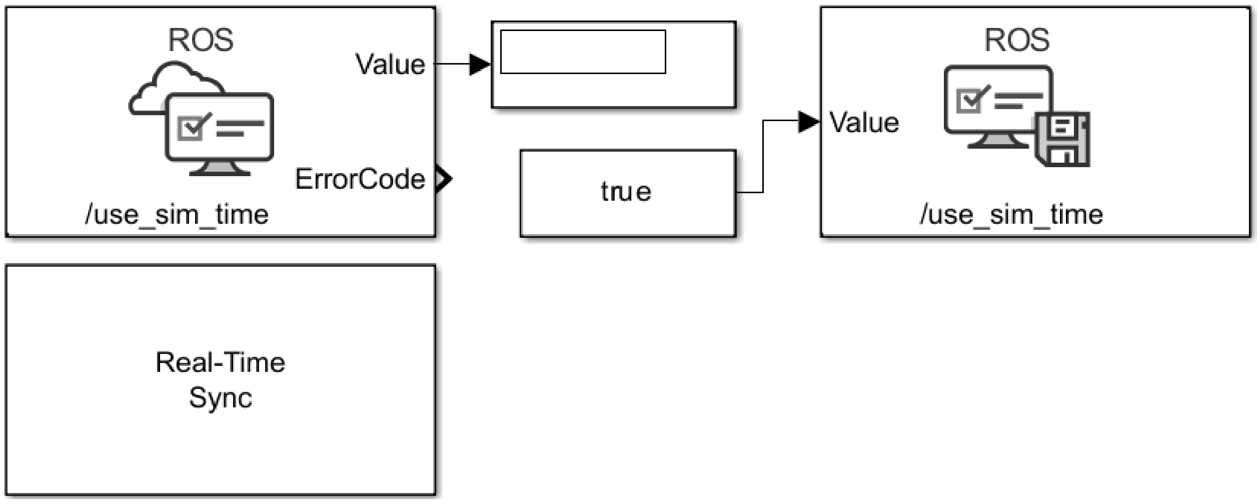


Figure 20: ROS and Matlab time synchronization in Simulink.

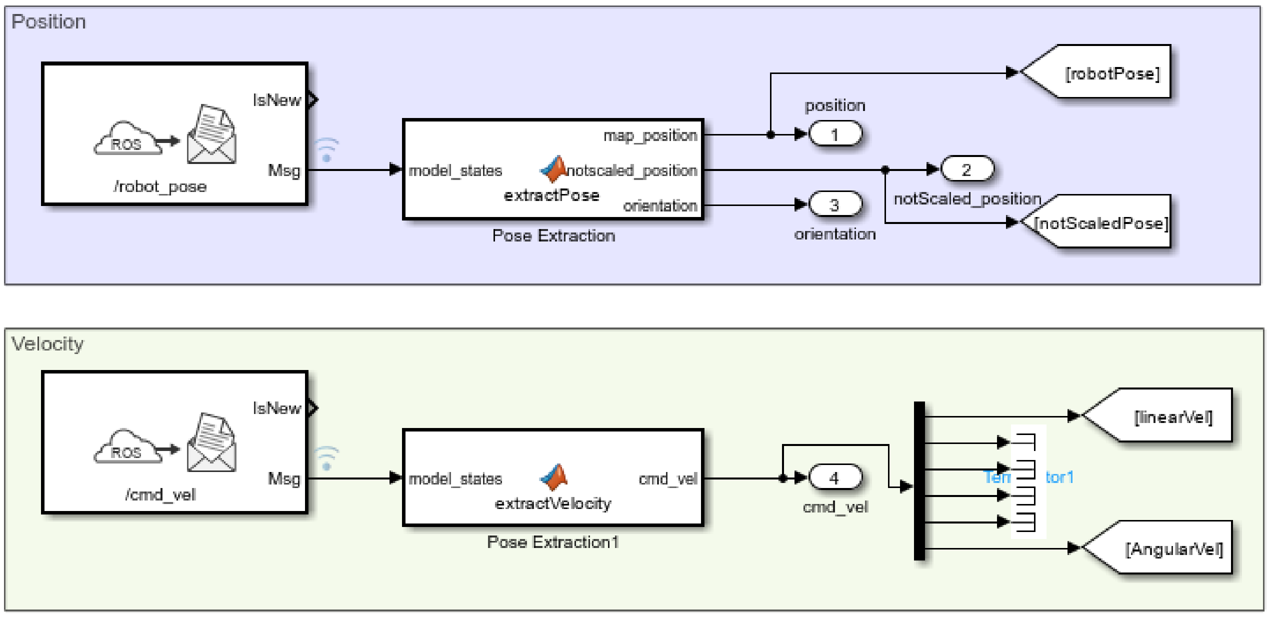


Figure 21: Simulink models to acquire MiR position and velocity.

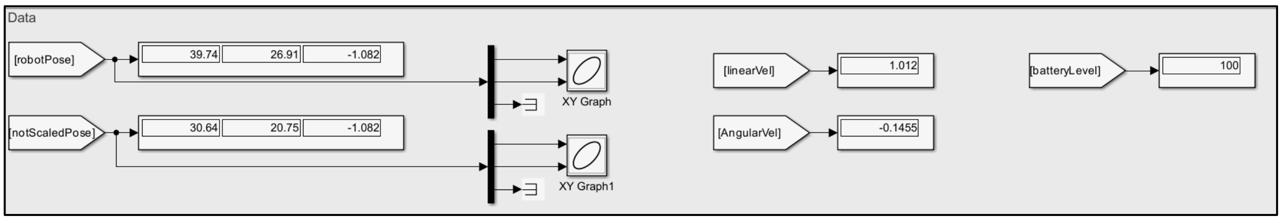


Figure 22: Simulink dashboard to monitor the MiR data in real-time.

2.3.2 Important considerations

To develop the purpose of this thesis, two schemes to acquire data through ROS Toolbox in Matlab have been tried:

1. Establish a direct connection between the Matlab PC and the MiR hardware, configuring the Matlab PC as the slave node and the MiR hardware as the master node (see *Figure 23*).
2. Connect the Matlab PC to an existing ROS network that includes the MiR robot as the slave and the machine running the ROS system as the master. Configure the Matlab PC as the slave node, while the ROS machine as the master node in the Matlab script (see *Figure 24*).

However, these two potential solutions exhibit unexpected differences in the message types and available topics. Based on these, it is necessary to modify the Matlab script (Appendix 11.2.4) and Simulink subscriber blocks (*Figure 21*) and functions (Appendix 11.2.3), as reported in the appendix, to accommodate to the correct message types.

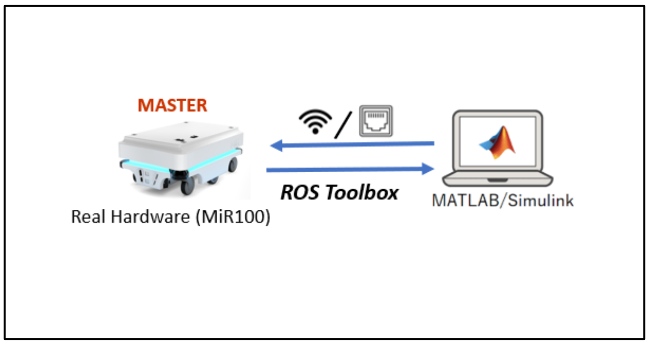


Figure 23: Matlab PC - MiR direct connection scheme.

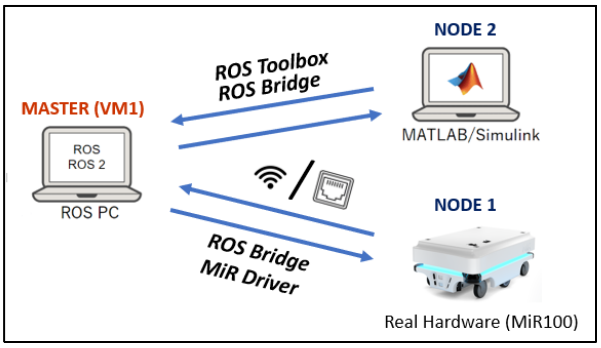


Figure 24: Matlab PC - ROS machine connection scheme.

The observed topics are '/robot_pose', '/cmd_vel', and '/MC/battery_percentage'. The latter is an interesting parameter to monitor, as it could be utilized to implement a battery status controller in Matlab or Simulink and perform actions such as moving the robot to a charging location. Another important topic that shows some variations is '/move_base/goal'. It is necessary to publish a new goal position for the MiR.

These variances are summarized in the following table.

Table 1: Differences in the message types in two illustrated connection schemes.

TOPIC	**MESSAGE TYPE**	
	Case 1	**Case 2**
/robot_pose	geometry_msgs/Pose	geometry_msgs/Pose
/cmd_vel	geometry_msgs/TwistStamped	geometry_msgs/Twist
/MC/battery_percentage	std_msgs/Float64	Not available
/move_base/goal	mir_actions/MirMoveBaseActionGoal	move_base_msgs/MoveBaseActionGoal

3 Case Study

The case study in this thesis focuses on a pick and place task executed by the MoMa robot. Specifically, the MoMa is programmed to reach a designated location for object pickup (Loading station), proceed to the next release station (Unloading station), and then return to the starting point (Charging station). The stations are shown in
Figure 27.
In the Digital Model chapter (Chapter 5), two different versions of the case study are presented:

1. **Orientation angles not specified**: The initial orientation angle at which the MoMa moves towards the Loading station and the subsequent departure orientation angles from the stations after the UR has completed its mission are unspecified.
2. **Orientation angles specified**: All the mentioned orientation angles in point 1 are specified.

These two variations are conducted to assess the accuracy of the Digital Model in two different situations and analyse the impact of the specified level of details in the MoMa program on its accuracy.

Figure 25 and *Figure 26* display respectively the two MoMa programs utilized for this case study.

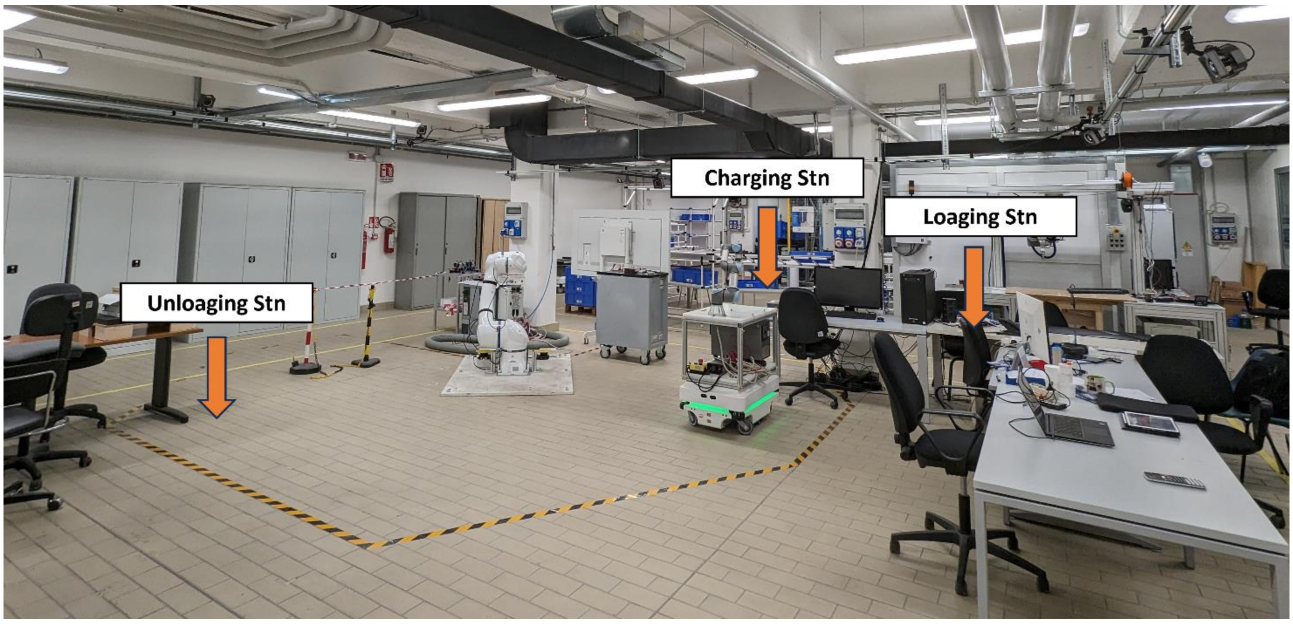
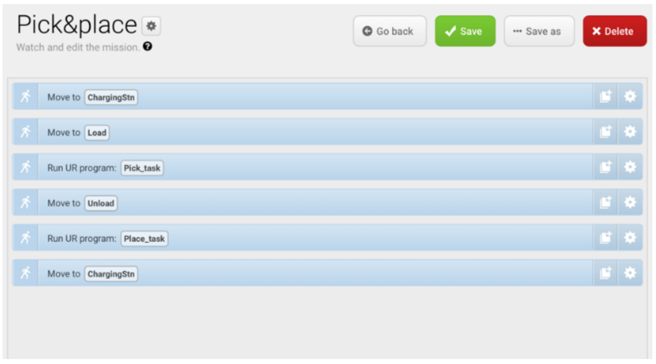
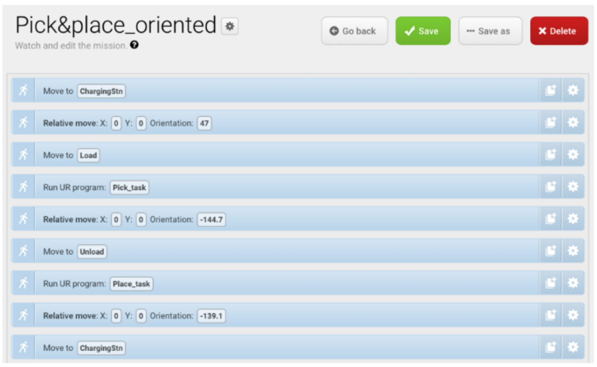


Figure 27: Pick and Place stations.



Move to ChargingStn
Move to Load
Run UR program: Pick_task
Move to Unload
Run UR program: Place_task
Move to ChargingStn

Figure 25: MoMa program with orientation angles not specified.



Move to ChargingStn
Relative move: X: 0 Y: 0 Orientation: 47
Move to Load
Run UR program: Pick_task
Relative move: X: 0 Y: 0 Orientation: -144.7
Move to Unload
Run UR program: Place_task
Relative move: X: 0 Y: 0 Orientation: -139.1
Move to ChargingStn

Figure 26: MoMa program with orientation angles specified.

4 PRELIMINARY TESTS

One of the primary objectives of this thesis is to assess the accuracy of the implemented Digital Twin and evaluate its effectiveness in a hypothetical industrial working environment. To achieve this, preliminary tests are conducted to compare the MiR position data sampled by ROS and Node-RED during the execution of the case study depicted in *Figure 26*, with the data captured by the Optitrack Motive acquisition system, which more accurately represents the real values (see Section 2.1).
Indeed, due to the dynamic nature of the MiR, which is constantly in motion and influenced by variables like obstacles along its path, accurately determining its real position relative to the operating environment becomes challenging. Therefore, Optitrack represents a very useful tool to obtain the most precise estimation possible of the MiR's position.

The ROS system is developed according to the diagram shown in the *Figure 24*.

4.1 GENERAL SETTINGS

4.1.1 Utilized software and tools

- Excel
- MiR100 software
- Node-RED
- Ubuntu 20.04.6 LTS with ROS Noetic version
- Oracle VM VirtualBox 7.0.8
- Matlab 2022b and Simulink equipped with ROS Toolbox
- Optitrack Motive

4.1.2 Sampling frequencies

Table 2: Sampling frequency of acquisition systems.

ACQUISITION SYSTEM	SAMPLING FREQUENCY [Hz]
Node-RED	10
ROS (Rosbag in Simulink)	2
Optitrack Motive	20

4.2 REPEATABILITY TEST

The proposed test involves twelve consecutive repetitions of the same trajectory, with the position of the MiR sampled at three stations: Loading, Unloading, and Charging.

4.2.1 Results and analysis

The analysis is conducted by first evaluating the accuracy of the systems used for data acquisition and then evaluating their reliability in terms of uncertainty.

I. **Accuracy and precision**: The positional data of the mobile robot, obtained in the twelve repetitions, are graphically presented for each station, along with the reference position obtained from the MIR's web interface (*Figure 28*). The black circle indicates the positioning accuracy defined in the mir datasheet[10], equal to 0.05m. As expected, the ROS system demonstrates the most consistent behaviour with this value, as it gathers data from the MIR sensors. Indeed, except for the Loading station, the data obtained with ROS consistently

[10] MiR datasheet: Technische Daten MiR 100 (EN).pdf

exhibits shorter distances from the reference point compared to other acquisition systems. Regarding the Loading station, it is observed that only a few values fall within the defined range from the datasheet, but the majority of the data still remains close to it. A possible explanation could be given by the fact that near this station there are several chairs (see *Figure 27*), which constitute an obstacle for the movement of the MiR that reaches the destination with less precision. On the other hand, Node-RED exhibits the poorest performance among the systems, with nearly all of the samples falling outside the range defined by the black circle. This will be further confirmed in the next chapter during the analysis of the Digital Model. However, by looking at the charts, it can be approximated that the maximum positional error obtained to be around 0.1m, as the majority of the data falls within this range for all stations, with only a few exceptional cases. To reinforce these assessments, the mean error values are considered, which represent the mean distances obtained for each system and station, as shown in *Table 4*. It can be observed that ROS provides the best results, with the maximum error occurring at the Loading station. Meanwhile, Node-RED confirms the previously described behaviour, as initially inferred from the graphs. In conclusion, ROS is a more robust and accurate system compared to Node-RED and both are precise systems, as the data collected for each station are quite close to each other.
Regarding Optitrack, it exhibits a more scattered distribution of samples, which in turn impacts the errors. Moreover, this system also presents a higher number of values outside the 0.1m range, possibly due to a couple of repetitions that were not conducted under optimal conditions. As expected, Optitrack is the least accurate and precise system when compared to the reference points. This is because, unlike ROS and Node-RED, Optitrack does not directly acquire data from the environment map created by the mir. Instead, it captures the position of the moving object relative to the real-world frame.

Table 3: Station reference positions from map created by MiR100.

STATIONS	X [m]	Y [m]	PHI [deg]
Loading Stn	29.2	22	90
Unloading Stn	31.427	18.852	-90
Charging Stn	28.95	21	0

Table 4: Mean errors (distances from ref. values).

MEAN ERRORS (DISTANCES) [m]			
STATIONS	Node-RED	ROS	Optitrack
Loading Stn	0.070	0.058	0.056
Unloading Stn	0.052	0.030	0.070
Charging Stn	0.087	0.040	0.071

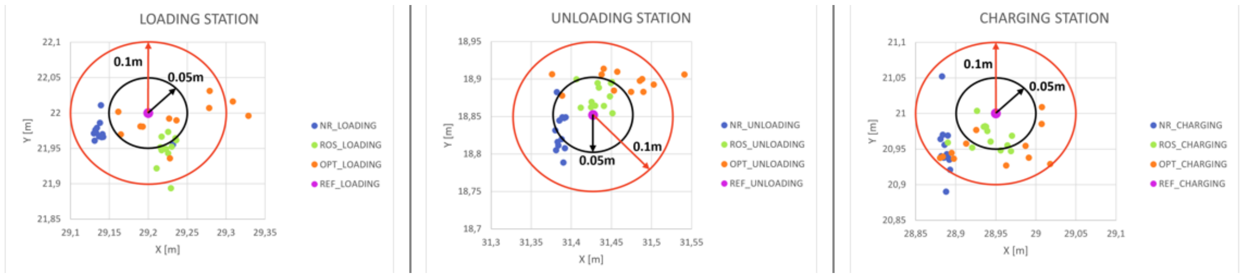


Figure 28: Repeatability test charts for accuracy and precision evaluations

II. **Uncertainty**: The uncertainty analysis for the three acquisition systems is conducted by comparing their extended uncertainties (U), which are calculated using the T-Student distribution due to the limited availability of few samples.

As already seen in section 2.1, the extended uncertainty is computed by mean of the following formula:

$$U = \pm ku = \pm k \frac{ME}{\sqrt{3}}$$

Regarding to the MiR system, a confident factor (k) equal to 2 is chosen based on the PUMA (Procedure for Uncertainty of Measurement Management) method with degrees of freedom (dof) equal to 100 and a confidence interval of 95%. This decision is driven by a high level of confidence in the positioning accuracy value provided by the OEM as stated in the MiR datasheet.
The dof for the three acquisition systems are determined by n-1, where *n* is the number of available samples. Based on this information and a confidence interval of 95%, the confident factor is computed by applying the inverse of the T-Student distribution in Excel. The obtained result is k = 2.201, which is consistent with the GUM table. The standard uncertainty (u) of each system is determined through covariance, which is obtained as a linear combination of two factors: the standard uncertainty of the instrument or device, and the standard deviation (σ) relative to the mean value of the acquired data. The formula used is provided below.

$$u^2 = \sqrt{\sigma^2 + {u_{sys}}^2}$$

For the Node-RED and ROS systems, the standard deviation of the MiR is considered, whereas, for Optitrack, the value computed in section 2.1, which is equal to 0.307mm, is taken into account.
Finally, for this analysis, the mean values of the x and y coordinates from the twelve repetitions are considered separately.
The results obtained are presented in the tables below.

Table 5: Uncertainties for the MiR100 system.

MiR100 SYSTEM	
Degrees of freedom	100
Confident factor (k)	2
Positioning accuracy (ME) [m]	0.05
Standard uncertainty (u) [m]	0.029
Extended Uncertainty (U) [m]	0.058

Table 6: Uncertainties at the Loading station.

LOADING STATION						
	X coordinate			**Y coordinate**		
	Node-RED	**ROS**	**Optitrack**	**Node-RED**	**ROS**	**Optitrack**
Degrees of freedom	11	11	11	11	11	11
Confident factor (k)	2.201	2.201	2.201	2.201	2.201	2.201
Standard deviation (σ) [m]	0.027	0.008	0.053	0.014	0.021	0.023
Standard uncertainty (u) [m]	0.039	0.030	0.053	0.032	0.036	0.023
Extended Uncertainty (U) [m]	0.087	0.066	0.116	0.070	0.078	0.051
Mean values of 12 samples [m]	29.144	29.223	29.233	21.973	21.948	21.992

Table 7: Uncertainties at the Unloading station.

UNLOADING STATION						
	X coordinate			Y coordinate		
	Node-RED	ROS	Optitrack	Node-RED	ROS	Optitrack
Degrees of freedom	11	11	11	11	11	11
Confident factor (k)	2.201	2.201	2.201	2.201	2.201	2.201
Standard deviation (σ) [m]	0.004	0.014	0.045	0.025	0.016	0.012
Standard uncertainty (u) [m]	0.029	0.032	0.045	0.038	0.033	0.012
Extended Uncertainty (U) [m]	0.064	0.071	0.098	0.084	0.072	0.026
Mean values of 12 samples [m]	31.387	31.434	31.461	18.829	18.877	18.897

Table 8: Uncertainties at the Charging station.

CHARGING STATION						
	X coordinate			Y coordinate		
	Node-RED	ROS	Optitrack	Node-RED	ROS	Optitrack
Degrees of freedom	11	11	11	11	11	11
Confident factor (k)	2.201	2.201	2.201	2.201	2.201	2.201
Standard deviation (σ) [m]	0.004	0.022	0.051	0.037	0.017	0.024
Standard uncertainty (u) [m]	0.033	0.051	0.051	0.066	0.045	0.024
Extended Uncertainty (U) [m]	0.072	0.111	0.113	0.145	0.100	0.053
Mean values of 12 samples [m]	28.887	28.941	28.947	20.951	20.968	20.953

To assess the compliance of the acquisition systems, particularly in relation to the MiR system, the charts below display the mean values and their corresponding extended uncertainties for each system and station. If a value, along with its uncertainty range, falls entirely within the highlighted green limits, which represent the extended uncertainty of the reference station coordinate from the MiR system, then that system is deemed compliant with the MiR specifications. Conversely, if a significant portion of the uncertainty range of a system overlaps with the MiR system, it is likely that the employed system is compliant, but it cannot be definitively stated at a 95% level of confidence. In all other cases, it is probable that the system is not compliant.

In this specific case study, it is observed that all the data, along with their respective extended uncertainties, overlap with the reference limits. Specifically, the majority of the systems are compliant with the specifications for each station, although it cannot be affirmed with 95% confidence level. Notably, at the Charging Station, the Node-RED system exhibits a less compliant behaviour compared to the other two systems.

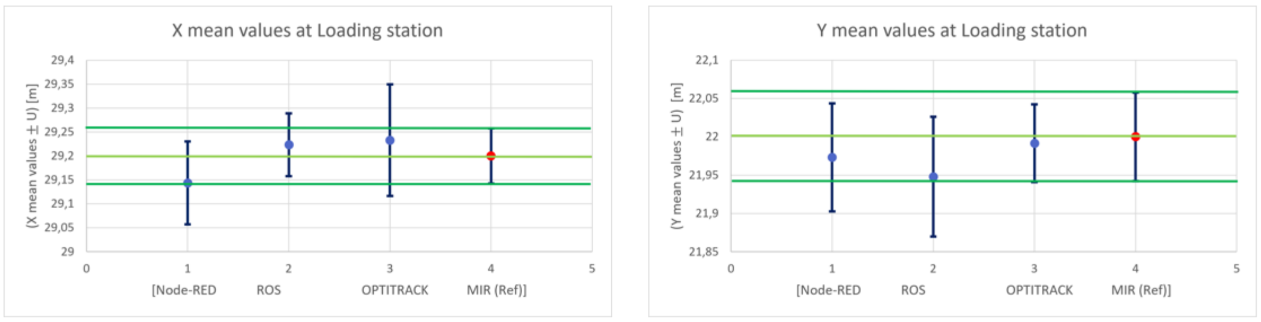


Figure 29: Mean values with extended uncertainties at Loading station.

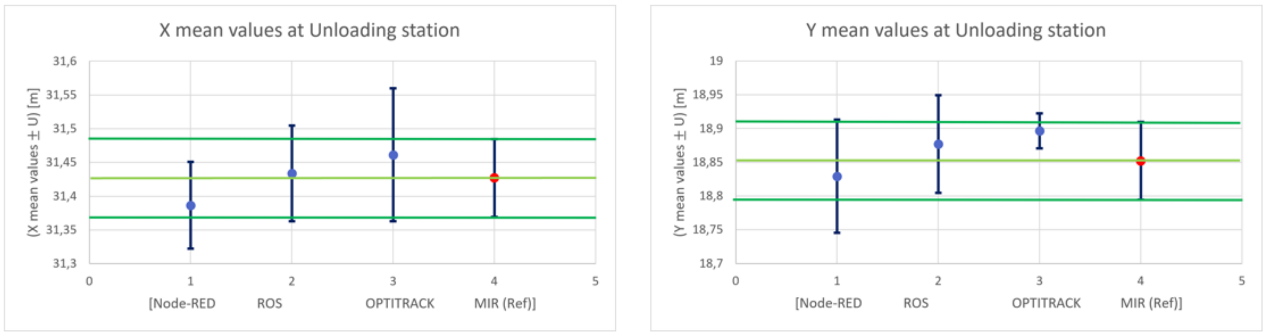


Figure 30: Mean values with extended uncertainties at Unloading station.

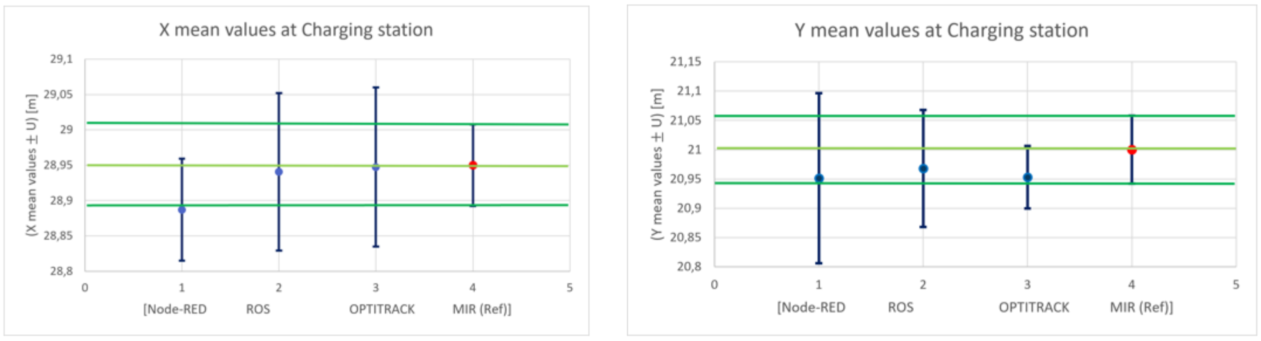


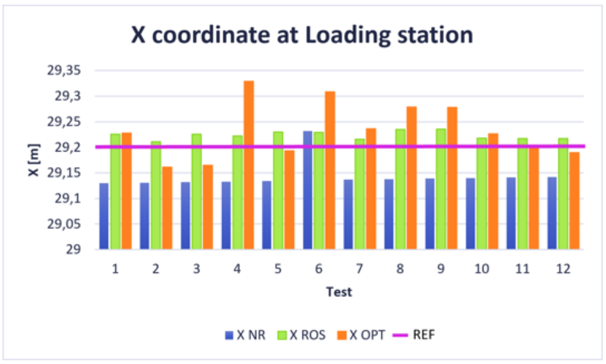
Figure 31: Mean values with extended uncertainties at Charging station.

III. **Comparison between ROS and Optitrack**: As mentioned at the beginning of this chapter, accurately estimating the real position of the MiR in relation to the real-world is challenging. To provide an assessment of the error in the data obtained with ROS compared to those assumed to be closer to the real values obtained with Optitrack, the measurement values from the different acquisition systems are shown through histograms, categorized by x and y coordinates for the three stations (see *Figure 32, 33, 34*). This allows for the evaluating which system, ROS or Node-RED, is closer to the Optitrack reference data.
Subsequently, to quantify the distance between ROS and Optitrack, the mean distance value between the two systems is calculated based on the twelve repetitions performed in the repeatability test. From the results presented in the

Table 9, a mean error of approximately 0.055m is obtained, except for the Loading station, where it is 0.065m. The possible reasons for this discrepancy correspond to the ones described during the analysis of accuracy.
By examining the results of the compliance test and the extended uncertainties, ROS generally performs better than Node-RED in terms of compliance with the limits defined by the uncertainty of Optitrack. This is particularly evident for the x coordinate, while for the y coordinate, both systems deviate considerably from the Optitrack range, although there is still overlap among all the data.

Table 9: Mean error between ROS and Optitrack.

ROS – OPTITRACK MEAN DISTANCES	
STATIONS	**MEAN ERRORS [m]**
Loading Stn	0.065
Unloading Stn	0.051
Charging Stn	0.054



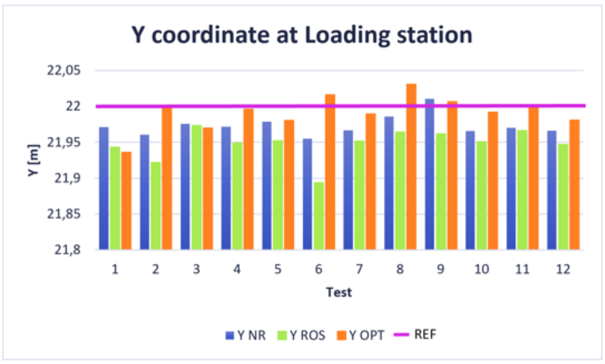
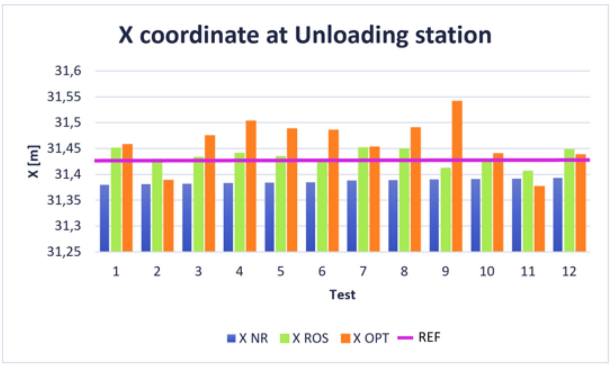


Figure 32: X and Y coordinates of data at Loading station.



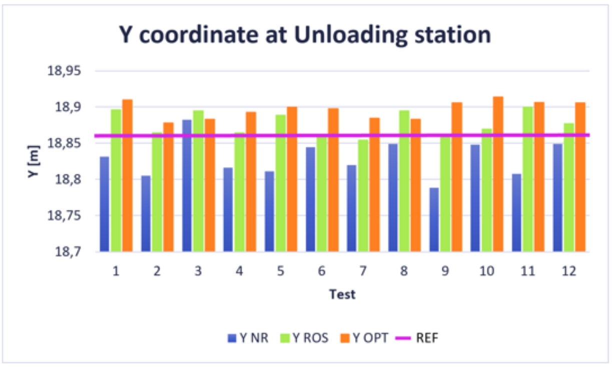
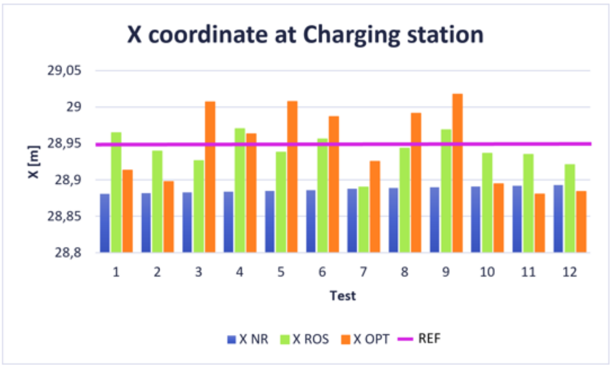


Figure 33: X and Y coordinates of data at Unloading station.



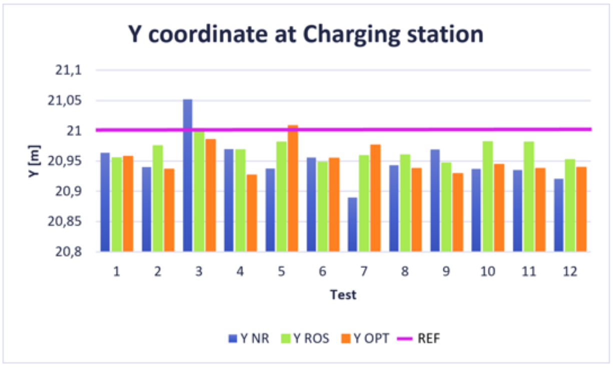


Figure 34: X and Y coordinates of data at Charging station.

4.3 Repetition of the test with one or two obstacles

The same analysis conducted for the repeatability test are repeated with the inclusion of one or two obstacles to assess their impact on the accuracy and precision of the MiR's navigation towards the designated stations. Specifically, ten repetitions are analysed, consisting of five trials with one obstacle and five trials with two obstacles. The positions of the obstacles and the trajectories executed by the MiR are depicted in the *Figure 35*.

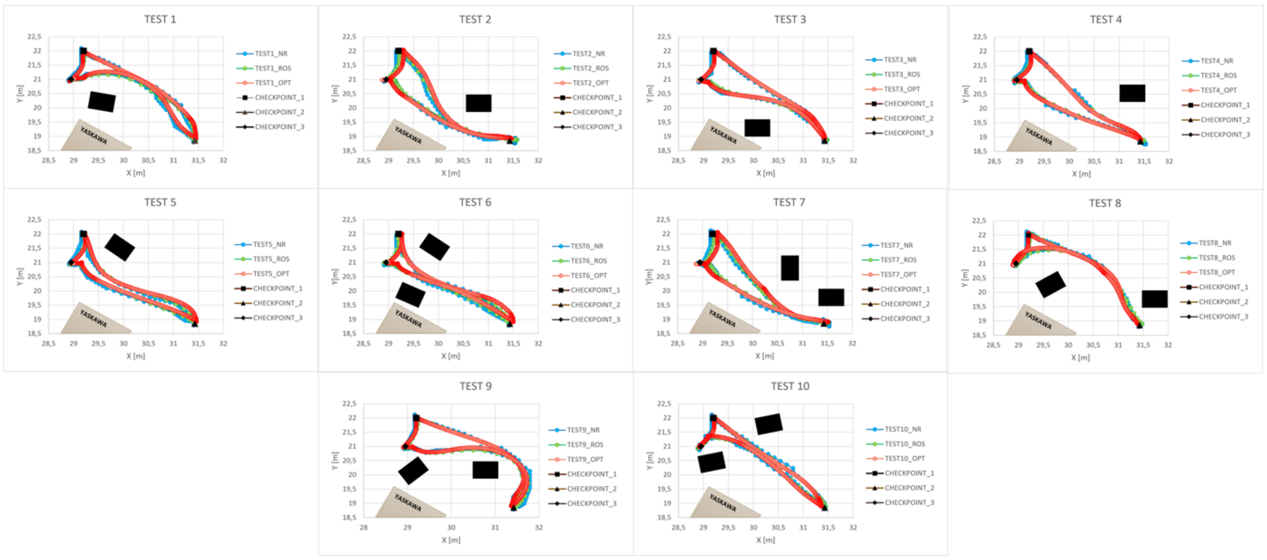


Figure 35: Ten scenarios with the presence of obstacles.

4.3.1 Results and analysis

I. **Accuracy and precision**: Considering the same reference positions for the three stations, as shown in the *Table 3*, even in the presence of obstacles, the errors obtained for the three acquisition systems are very similar to those of the previous test. In particular, ROS proves to be the most accurate and precise system, with the highest error occurring at the Loading station, similar to the repeatability test. Looking at the *Figure 36*, Node-RED and Optitrack exhibit behaviour consistent with the previous results, with Node-RED again having most of the samples outside the range corresponding to the positioning accuracy value provided by the MiR datasheet. However, the performance at the Loading and Unloading stations shows an unexpected improvement compared to the previous test. Even in this test, despite the presence of obstacles, all collected data have a maximum distance from the reference of approximately 0.1m.

Table 10: Mean errors (distances from ref. values) with obstacles.

MEAN ERRORS (DISTANCES) [m]			
STATIONS	**Node-RED**	**ROS**	**Optitrack**
Loading Stn	0.048	0.056	0.053
Unloading Stn	0.051	0.039	0.069
Charging Stn	0.078	0.044	0.052

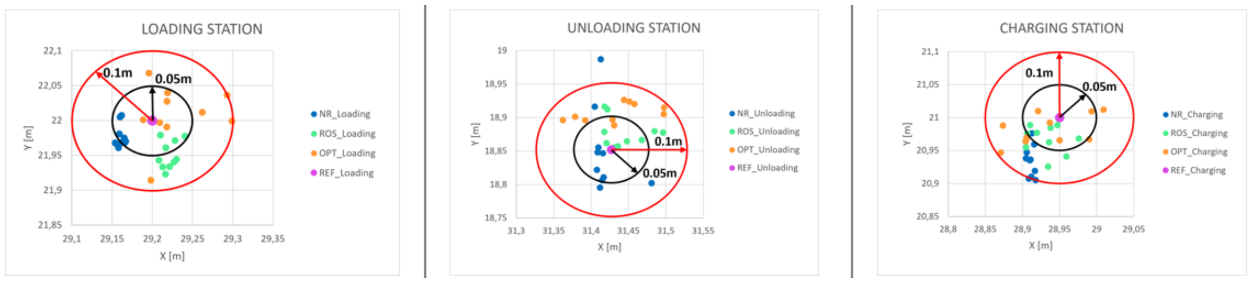


Figure 36: Obstacle test charts for accuracy and precision evaluations

II. **Uncertainty**: By employing the same calculations described in the repeatability test to determine the extended uncertainty, but with a degrees of freedom value of 9 for the three acquisition systems, the results presented in the tables below are obtained. Upon analysing the graphs for each station, it is observed that even in the presence of obstacles, all three systems are overlapped to the reference extended uncertainties of the MiR (indicated by the green lines). Therefore, all the systems are compliant, but it cannot be stated with the 95% confidence level. Similar to the repeatability test, Node-RED exhibits behaviour that is less compliant than the other two, particularly at the Charging station.

Table 11: Uncertainties at the Loading station with obstacles.

LOADING STATION						
	X coordinate			Y coordinate		
	Node-RED	ROS	Optitrack	Node-RED	ROS	Optitrack
Degrees of freedom	9	9	9	9	9	9
Confident factor (k)	2.262	2.262	2.262	2.262	2.262	2.262
Standard deviation (σ) [m]	0.004	0.010	0.038	0.015	0.019	0.039
Standard uncertainty (u) [m]	0.039	0.030	0.038	0.033	0.035	0.039
Extended Uncertainty (U) [m]	0.087	0.069	0.086	0.074	0.078	0.088
Mean values of 10 samples [m]	29.161	29.221	29.230	21.977	21.951	22.008

Table 12: Uncertainties at the Unloading station with obstacles.

UNLOADING STATION						
	X coordinate			Y coordinate		
	Node-RED	ROS	Optitrack	Node-RED	ROS	Optitrack
Degrees of freedom	9	9	9	9	9	9
Confident factor (k)	2.262	2.262	2.262	2.262	2.262	2.262
Standard deviation (σ) [m]	0.021	0.028	0.044	0.057	0.020	0.013
Standard uncertainty (u) [m]	0.036	0.040	0.044	0.064	0.035	0.013
Extended Uncertainty (U) [m]	0.081	0.090	0.099	0.145	0.080	0.029
Mean values of 10 samples [m]	31.419	31.444	31.434	18.849	18.877	18.907

Table 13: Uncertainties at the Charging station with obstacles.

CHARGING STATION						
	X coordinate			Y coordinate		
	Node-RED	ROS	Optitrack	Node-RED	ROS	Optitrack
Degrees of freedom	9	9	9	9	9	9
Confident factor (k)	2.262	2.262	2.262	2.262	2.262	2.262
Standard deviation (σ) [m]	0.004	0.022	0.047	0.022	0.020	0.022
Standard uncertainty (u) [m]	0.029	0.036	0.047	0.036	0.035	0.022
Extended Uncertainty (U) [m]	0.066	0.083	0.106	0.082	0.079	0.049
Mean values of 10 samples [m]	28.912	28.933	28.936	20.933	20.966	20.982

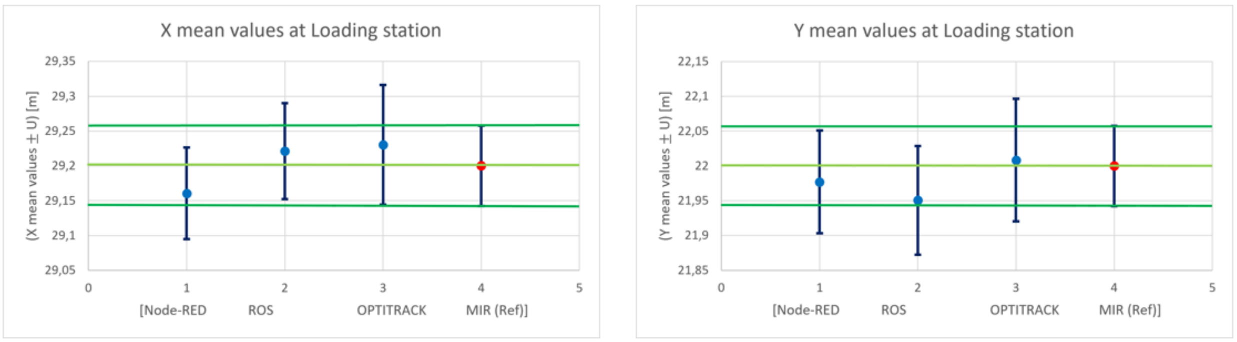


Figure 37: Mean values with extended uncertainties at Loading station in the presence of obstacles.

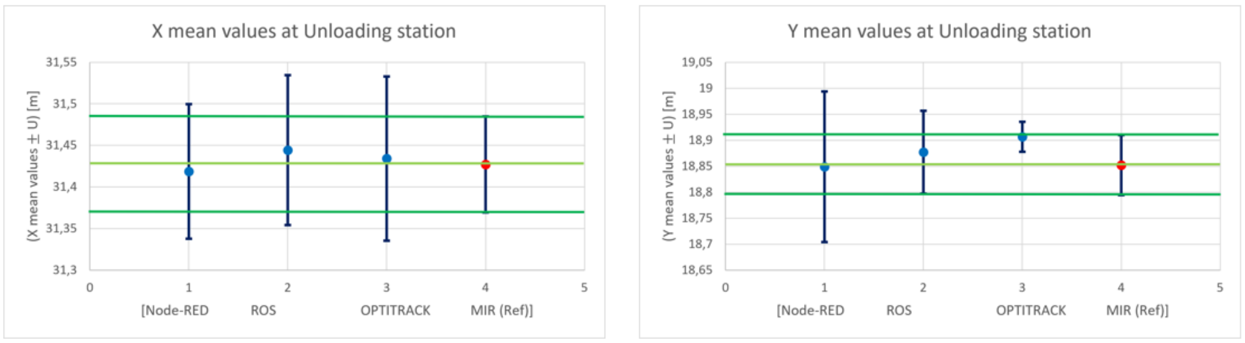


Figure 38: Mean values with extended uncertainties at Unloading station in the presence of obstacles.

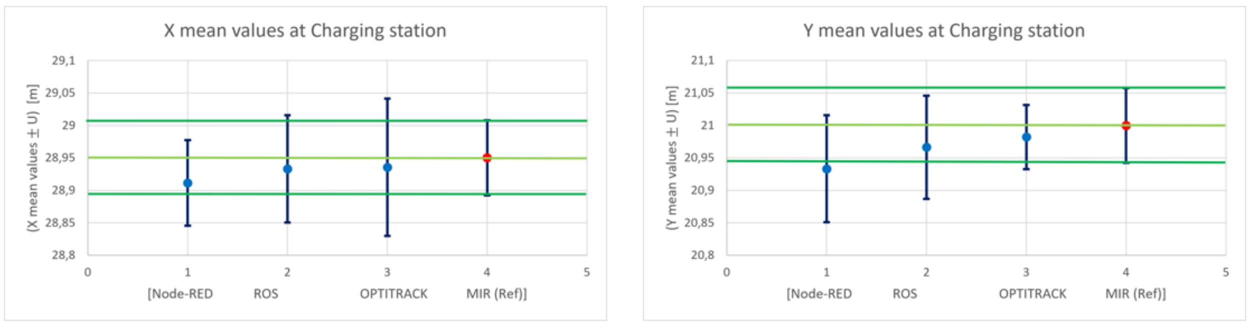


Figure 39: Mean values with extended uncertainties at Charging station in the presence of obstacles.

4.4 CONCLUSION

In conclusion, based on the execution and analysis of these two preliminary tests, it is evident that the ROS system, overall, exhibits greater accuracy and precision than Node-RED when evaluating acquired data in relation to the reference points aligned with the MiR's created map. Comparing to the real-world reference system, with Optitrack assumed as the closest representation to reality, ROS demonstrates a distance of approximately 0.055m.
The compliance test indicates that all systems are compliant to the specifications outlined in the MiR datasheet when considering the mean positions along with their extended uncertainties. However, achieving a 95% confidence level of compliance is only observed in a few cases and not in the presence of obstacles.
The presence of obstacles along the trajectory has minimal impact on the accuracy of the systems and does not alter the previously made considerations.

To improve precision and accuracy, the following approaches can be implemented:

• Increase the sampling frequency of the different acquisition systems to obtain more data points.
• Conduct additional repetitions for the repeatability test.
• Enhance the environment by removing any potential objects that could interfere with the trajectory and act as obstacles, especially near the destination points.

For the purposes of this thesis and considering the available information, in-depth statistical analysis is not required. The objective of these tests is to provide insights and initial comparisons among the different acquisition systems, facilitating the appropriate interpretation and evaluation of the results obtained during simulations with the Digital Model developed in the subsequent chapter.

5 Digital Model

5.1 Method

5.1.1 Utilized software and tools

The following tools and software are used to implement the DM:

- Matlab 2022b, Simulink, and Simscape Multibody
- Matlab Toolbox: Robotics System Toolbox, Mobile Robotics Simulation Toolbox, ROS Toolbox, Stateflow
- MiR100 and UR3 software
- Node-RED
- Ubuntu 20.04.6 LTS with ROS Noetic version
- Oracle VM VirtualBox 7.0.8
- Optitrack Motive
- Excel

The features and applications of the Node-RED, ROS, and Optitrack systems remain unchanged in accordance with the explanations presented in Chapter 2. The diagram depicting the connection between Matlab, ROS, and MiR is illustrated in the *Figure 24*.

The additional Toolboxes employed in Matlab for this project are discussed in the next sections.

5.1.2 General settings and considerations

The MiR100 is treated as a differential drive robot (see *Figure 41*) centered at the middle point along the wheel axle. The position is denoted as a vector with three elements [x, y, ϕ], with ϕ representing the orientation angle. Linear and angular velocities are obtained through the forward velocity kinematics equations, while the wheel speeds are determined using inverse kinematics.
These equations are:

$$\text{Linear velocity: } v_r = \dot{X}_r = \frac{v_R + v_L}{2} \qquad \text{Angular velocity: } \omega_r = \dot{\theta}_r = \frac{v_R - v_L}{2L}$$

$$\text{Right wheel: } w_R = \frac{2v_r + \omega_r(2L)}{2R} \qquad \text{Left wheel: } w_L = \frac{2v_r - \omega_r(2L)}{2R}$$

Figure 40 illustrates the angle diagram utilized as a reference to implement a function for scaling the orientation angle of the simulation model, ensuring consistent values with those of the actual model.

Regarding the solver configuration, an automatic solver with variable step-time is selected from the Simulink model settings.

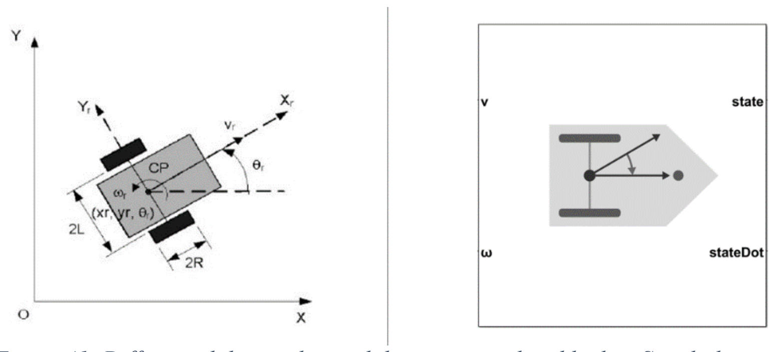


Figure 41: Differential drive robot and the corresponding block in Simulink.

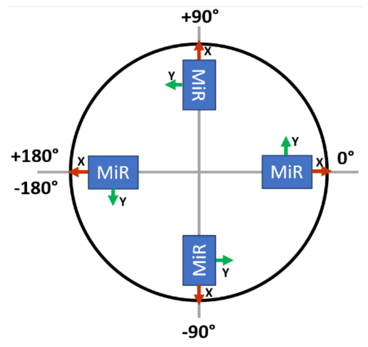


Figure 40: MiR orientation angle reference scheme.

5.1.3 Binary Occupancy Map

The ‘BinaryOccupancyMap’ function generates a 2D occupancy map object that is utilized to represent and visualize a robot's workspace, including obstacles. This function operates on an image of the environment map, captured by the mobile robot’s sensors and laser scanner. It produces an ‘OccupancyGrid’, a logical matrix where free zones are denoted by ‘0’ logic and obstacles by ‘1’ logic[11]. Consequently, this process generates an approximation map with a lower resolution compared to the actual environment, as depicted in *Figure 42*.

Since the resulting map is scaled compared to the actual measurements of the map created by the MiR, a scale factor is calculated using the reference point of the lower-left corner, as highlighted in *Figure 42*, of the Yaskawa robot platform present in the laboratory. This involves dividing the coordinates displayed in the two maps. The scale factor is calculated to be around 1.297. Considering that obtaining the exact coordinates of the same point in both maps is nearly impossible, so the resulting value is utilized as an approximation with the only purpose of providing a visualization of the simulated model that closely resembles the real one. It is important to note that this scaling factor does not affect the numerical results and the analysis of the simulation.

The Matlab script is shown in Appendix 11.3.1.

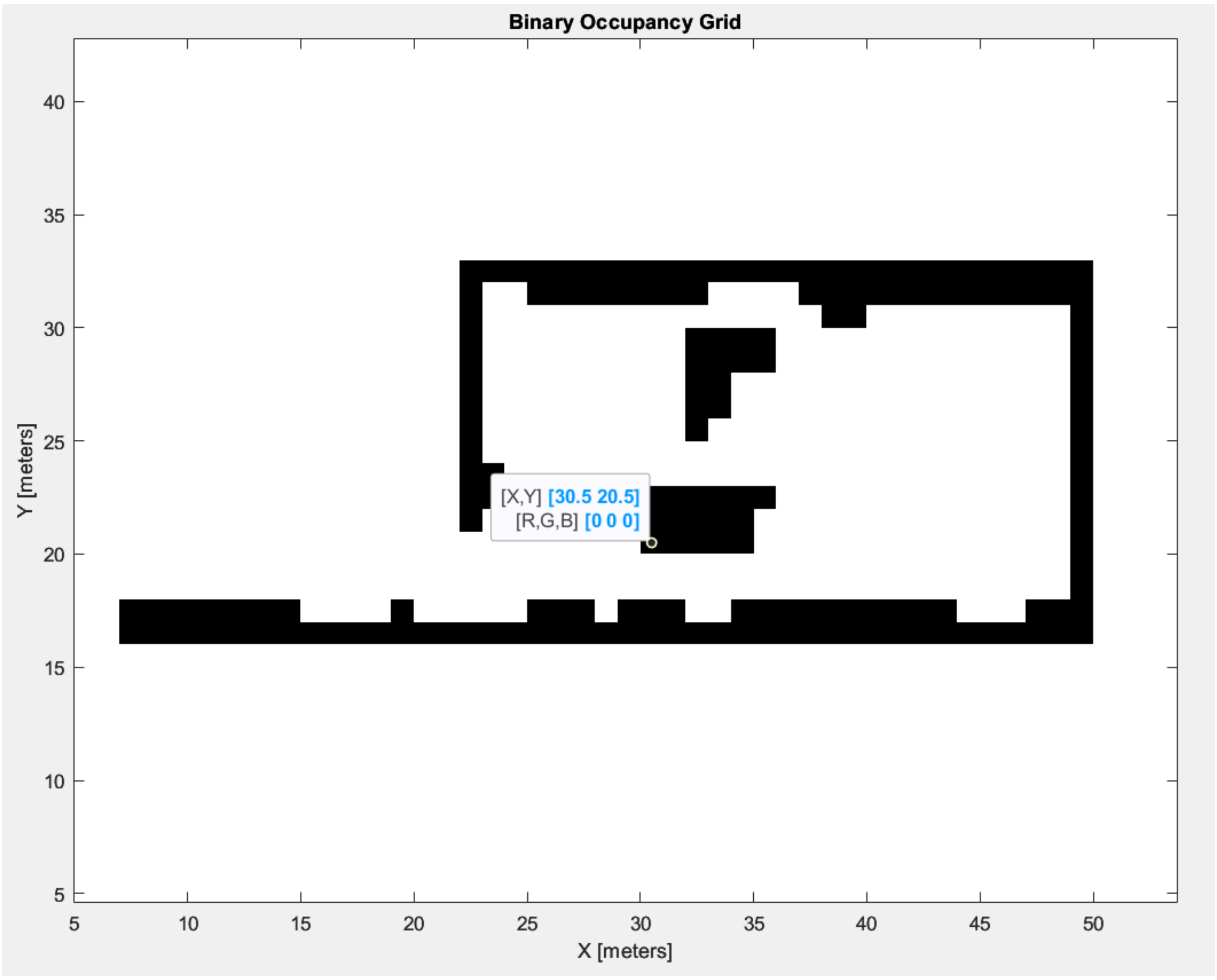


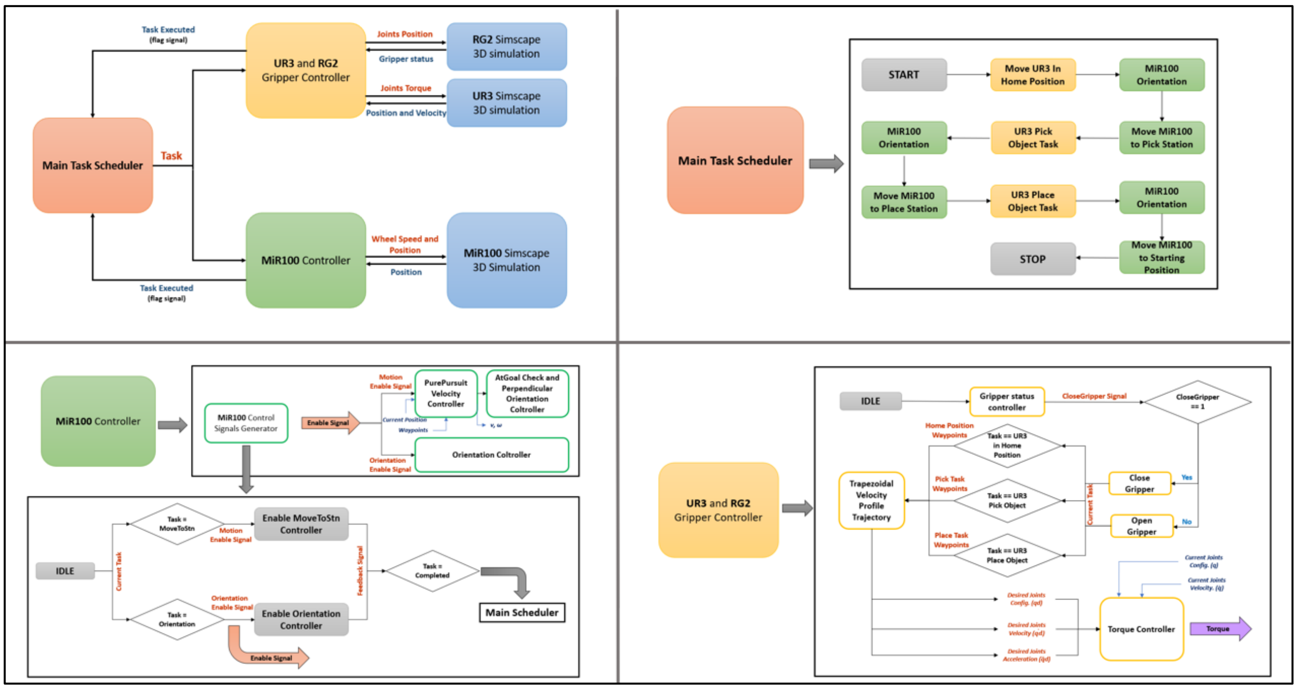
Figure 42: Binary Occupancy Grid in Matlab.

[11] BinaryOccupancyMap function: Create occupancy grid with binary values - MATLAB - MathWorks Italia

5.1.4 Main idea

The main concept behind the proposed DM is to implement a finite state machine that incorporates a main scheduler. Within this framework, various states are defined to facilitate the execution of the pick and place task performed by the MoMa, allowing for efficient management of actions for both the MiR100 and the UR3. Along with the main scheduler, two additional stateflows are developed within the two macro-blocks, representing the robots of the MoMa, to control and organize their actions. For the mobile robot, this control involves monitoring feedback signals to determine its arrival at reference positions or the achievement of the programmed orientation. On the other hand, the cobot's controller defines three primary states: ‘Home Configuration’, ‘Pick Task’, and ‘Place Task’. Within these states, control functions are specified to check the configurations of the UR joints and the gripper status. For the development of this Digital Model, the example ‘Warehouse Pick-and-Place Application Using Mobile Manipulator’ available on the official MathWorks website[12] was utilized as a reference.

Below are presented block diagrams to provide a clearer interpretation of the described concepts.



Main Task Scheduler
Task
Task Executed (flag signal)
UR3 and RG2 Gripper Controller
Joints Position
Gripper status
Joints Torque
Position and Velocity
RG2 Simscape 3D simulation
UR3 Simscape 3D simulation
MiR100 Controller
Wheel Speed and Position
Position
MiR100 Simscape 3D Simulation
Task Executed (flag signal)
Main Task Scheduler
START
Move UR3 In Home Position
MiR100 Orientation
Move MiR100 to Pick Station
UR3 Pick Object Task
MiR100 Orientation
Move MiR100 to Place Station
UR3 Place Object Task
MiR100 Orientation
Move MiR100 to Starting Position
STOP
MiR100 Controller
MiR100 Control Signals Generator
Enable Signal
Motion Enable Signal
PurePursuit Velocity Controller
AtGoal Check and Perpendicular Orientation Coltroller
Current Position
Waypoints
v, ω
Orientation Enable Signal
Orientation Coltroller
IDLE
Current Task
Task = MoveToStn
Motion Enable Signal
Enable MoveToStn Controller
Task = Orientation
Orientation Enable Signal
Enable Orientation Controller
Enable Signal
Feedback Signal
Task = Completed
Main Scheduler
UR3 and RG2 Gripper Controller
IDLE
Gripper status controller
CloseGripper Signal
CloseGripper == 1
Yes
No
Close Gripper
Open Gripper
Current Task
Home Position Waypoints
Task == UR3 in Home Position
Pick Task Waypoints
Task == UR3 Pick Object
Place Task Waypoints
Task == UR3 Place Object
Trapezoidal Velocity Profile Trajectory
Desired Joints Config. (qd)
Desired Joints Velocity (q̇d)
Desired Joints Acceleration (q̈d)
Current Joints Config. (q)
Current Joints Velocity. (q̇)
Torque Controller
Torque

Figure 43: Block diagrams of the DM.

[12] Reference example for DM development: Design and Simulate Warehouse Pick-and-Place Application Using Mobile Manipulator in Simulink and Gazebo - MATLAB & Simulink - MathWorks Italia

5.2 Development in Matlab and Simulink

This section provides a description of some notable features of the Digital Model implemented in Simulink (see *Figure 44*), based on the block diagrams shown in *Figure 43*.

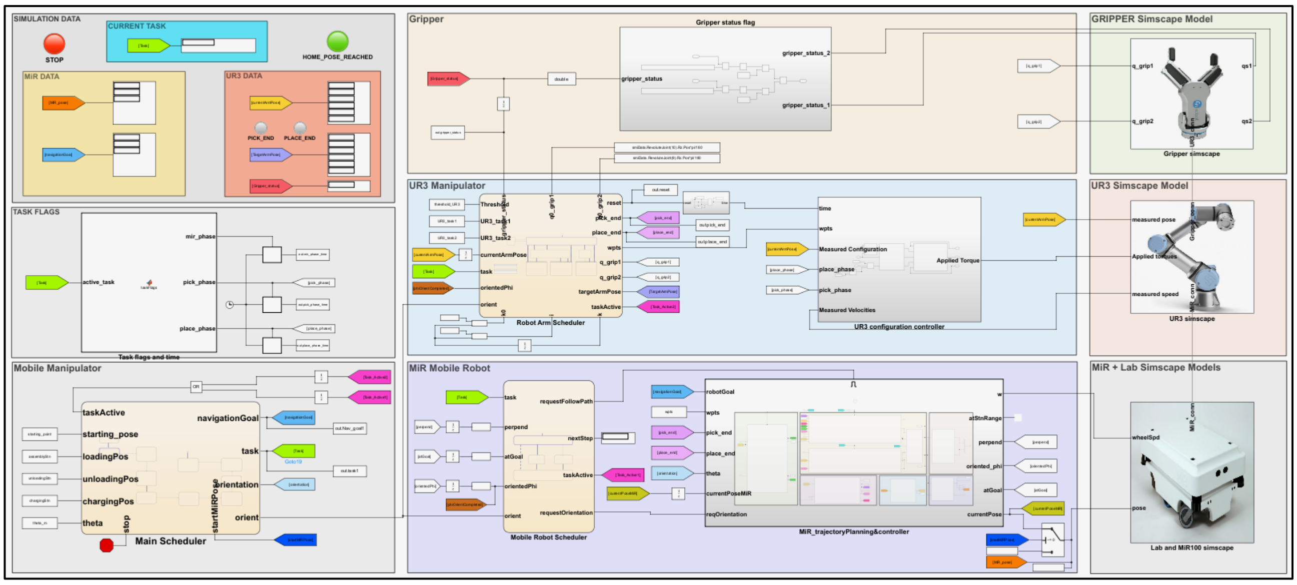


Figure 44: DM implementation in Simulink and Simscape.

5.2.1 Stateflow and Main task scheduler

Stateflow offers a graphical language including state transition diagrams and flowcharts. It facilitates the description of how Matlab algorithms and Simulink models respond to input signals, events, and time-based conditions. Furthermore, Stateflow empowers the design and development of supervisory control and operation planning[13].

The main task scheduler, depicted in *Figure 45*, plans operations for the MiR and UR, though a task signal. This is defined as a Matlab class (see Appendix 11.3.2) and assigned within each state of the scheduler. The control condition for state transitions is determined by an input feedback signal called 'taskActive'. This signal is sourced from either the MiR scheduler (labelled as 'Task_Active1') or the UR3 scheduler (labelled as 'Task_Active2'). The remaining input signals include the position coordinates of the three stations involved in the pick and place task, the starting point, which coincides with the Charging station in this case, and the vector of orientation angles ('theta') programmed on the MiR website, which specify the orientation when the MiR departs from the stations.

The task Matlab class consists of the following tasks: Arm_Pick, Arm_Place, Arm_Home, Robot_Navigate, and Idle. Correspondingly, the main scheduler comprises the following states:

- Start: Initializes the MoMa variables to their initial value in each Idle state of the schedulers.
- MoveArmHome: Moves the UR3 to the Home position, corresponding to the first waypoint in the Pick task program.
- OrientToNextGoal: Orients the MiR in the direction indicated by 'theta(i)', where 'i' represents the index corresponding to the station from which the robot needs to depart. In this state, the 'orient' flag is set to 1 to activate the orientation branch within the MiR scheduler.
- NavigateToPickPose: Enables the MiR motion to achieve the pick station (Loading station).

[13] Stateflow: Stateflow - MATLAB (mathworks.com)

- PickObject: Moves the UR3 and Gripper to perform the pick task. The UR3 configuration and Gripper status are managed and controlled by the respective Pick state within the UR scheduler.
- NavigateToPlacePose: Enables the MiR motion to achieve the place station (Unloading station).
- PlaceObject: Moves the UR3 and Gripper to perform the place task. The UR3 configuration and Gripper status are managed and controlled by the respective Place state within the UR scheduler.
- NavigateToEndPose: Enables the MiR motion to achieve the final station (Charging station).

Once this final state is successfully completed, the main scheduler sets the 'stop' signal to 1, thereby terminating the simulation.

The main output signals consist of the following: 'navigationGoal' provides the coordinates of the current goal station, 'orientation' is set to theta(i) as explained in the OrientToNextGoal state, and 'orient' defines a condition for the state transition within the MiR scheduler.

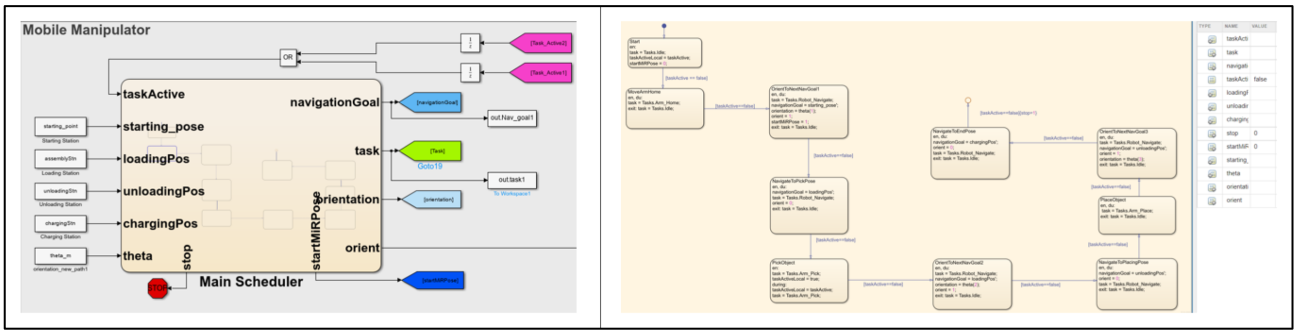
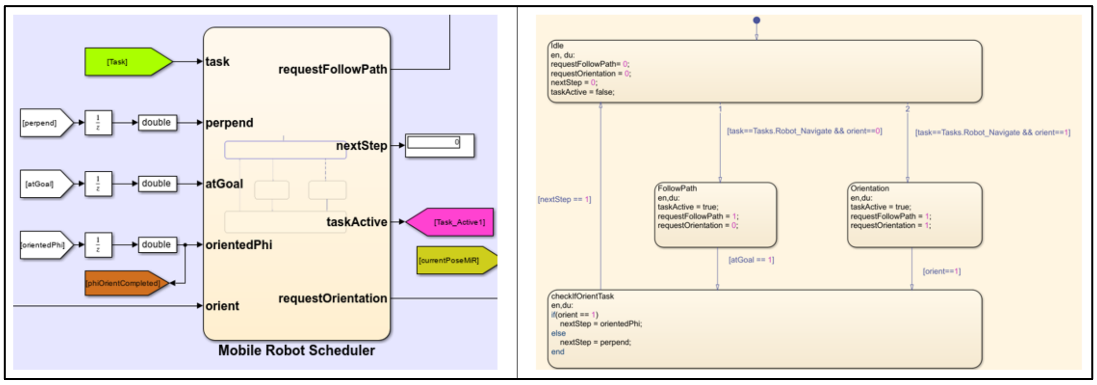


Figure 45: Main Task Scheduler in Simulink.

5.2.2 MiR100 controller

The MiR scheduler, illustrated in *Figure 46*, governs the actions of the MiR through the utilization of the 'orient' flag. Within this scheduler, two states, 'FollowPath' and 'Orientation', are defined and selected based on the false and true values of the 'orient' signal, respectively. These states determine the setting of the control signal 'requestOrientation' as either false or true. The 'requestOrientation' signal, indicated in light blue in *Figure 54*, activates or deactivates various subsystems in the MiR trajectory controller based on the required action, whether it is orientation or movement. This signal plays a crucial role in optimizing the model and preventing unnecessary processes. On the other hand, the 'requestFollowPath' signal enables the entire 'MiR_trajectoryPlanning&Controller' subsystem exclusively when a MiR action is needed.



```
Idle
en, du:
requestFollowPath= 0;
requestOrientation = 0;
nextStep = 0;
taskActive = false;
```

```
FollowPath
en,du:
taskActive = true;
requestFollowPath = 1;
requestOrientation = 0;
```

```
Orientation
en,du:
taskActive = true;
requestFollowPath = 1;
requestOrientation = 1;
```

```
checkIfOrientTask
en,du:
if(orient == 1)
   nextStep = orientedPhi;
else
   nextStep = perpend;
end
```

Figure 46: MiR scheduler in Simulink.

The structure of the ‘MiR_trajectoryPlanning&Controller’ subsystem is organized as follows: at the lower part, there are the blocks and functions responsible for controlling the position of the MiR robot, while the upper part is dedicated to the velocity control. Specifically, within this controller, the following subsystems are implemented:

- Pure Pursuit controller: It is a tracking algorithm designed for path following purposes. It takes waypoints as input and based on the current position of the robot provided as the second input, it produces linear and angular velocities as output. The waypoints correspond to the position data obtained from the ROS system during the previous execution of the task in the real-world. The controller’s tuning parameters include the desired linear velocity, maximum angular velocity, and look ahead distance. The look ahead distance determines how far ahead along the path the robot should consider computing the angular velocity. Choosing an appropriate look ahead distance is crucial, as a small value can lead to quick movements towards the path with potential oscillations, while a large value may result in significant curvature near corners. Examples of these situations are depicted in the figure below.
  The pure pursuit controller action is triggered when the value of requestOrientation is false.

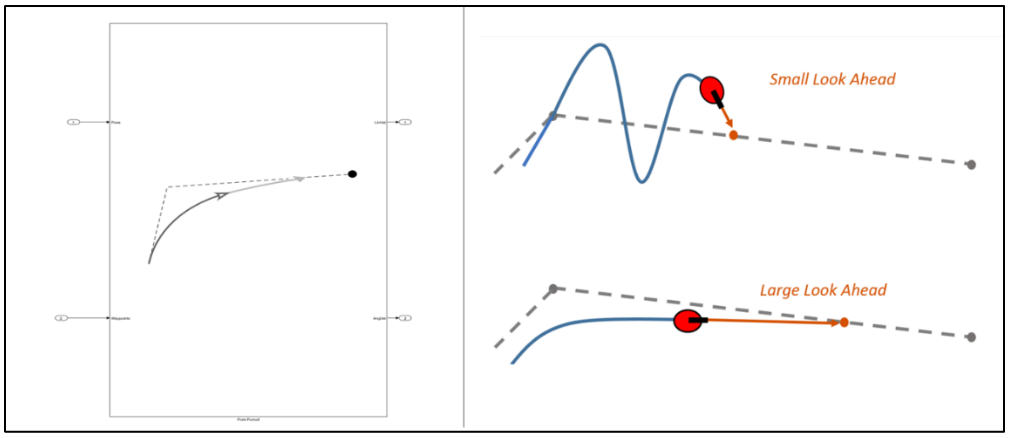


Figure 47: ‘Pure Pursuit Controller’ block in Simulink and Look Ahead parameter limit situations.

- Check distance to goal: This block verifies if the robot has arrived at the goal station. Due to the nature of the waypoints provided by the ROS system, the mobile robot cannot achieve perfect accuracy in reaching the target location, as analysed in the repeatability test in the previous chapter. These waypoints are utilized as the reference path for the robot's motion. Consequently, the reference position corresponding to the navigation goal differs from the real one, which is provided in input to this block as the variable ‘robotGoal’. Therefore, it is necessary to define a reasonable range within which the robot is considered to have reached the target. This range is typically set to approximately 0.1m, which corresponds to the maximum estimated error obtained from the repeatability test. This value is then compared to the distance between the real goal station coordinates and the current position of the DM.
  Specifically, two control signals are defined: ‘atStnRange’ and ‘atGoal’.
    - AtStnRange represents the condition to initiate robot orientation until it aligns with the orientation associated with the current goal station, as indicated in *Table 3*. To define the ‘atStnRange’ signal, two distance thresholds are employed for the x and y coordinates (see *Figure 48*). This approach helps enhance the reachability of the station, as the pure pursuit controller tends to introduce curvature near corners.
      This signal is provided by the Matlab function ‘checkAtStnRange’ (see Appendix 11.3.3) following the aforementioned procedure.

- AtGoal constitutes the condition to stop the robot's movement and launch the UR task. This signal is set to true when the distance from the goal falls below the chosen threshold, for example 0.1m, as described earlier.

The entire subsystem is enabled when the value of FollowPath is true.

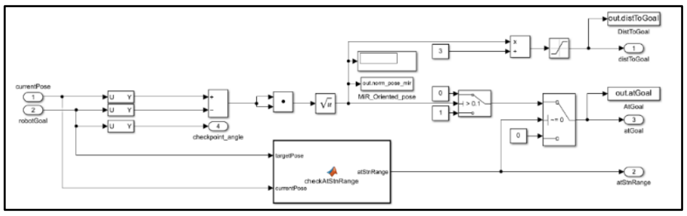


Figure 49: 'Check distance to goal' subsystem in Simulink.

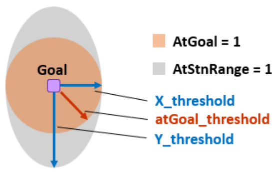


Figure 48: Illustration of the thresholds involved in the subsystem.

- Check the departure orientation angle: The purpose of this subsystem is to compare the absolute values and signs of the programmed departure angle (theta) for the MiR mission with the current orientation of the robot. During this phase, the robot's linear velocity is set to zero in order to achieve the desired orientation by controlling only the angular velocity, which is set to a specific value. The direction of angular velocity is determined by two Matlab function blocks: 'thetaQuarterSelection' and 'angVelSignController' (see Appendix 11.3.4). Based on the current orientation angle of the robot, if it needs to turn clockwise, the angular velocity sign is negative; otherwise, it is positive. The clockwise or counterclockwise rotation is determined by dividing the diagram shown in *Figure 40* into eight sections, where initially the reference orientation angle and the current angle are located, and checking in which direction the distance is smaller. The thetaQuarterSelection function outputs the current situation based on the quarter numbers corresponding to the positions of the two angles on the diagram. Then, the angVelSignController function implements a switch case to determine the sign of the angular velocity. The orientation task is considered complete and the output signal 'oriented_phi' is set to 1 when the absolute distance between the absolute values of the two angles is below a desired threshold and the angles signs are equal. The orientation mode selector switch allows for choosing whether to only consider the signs of the angles or also the difference between their absolute values.
  This subsystem is activated when the requestOrientation signal is true.

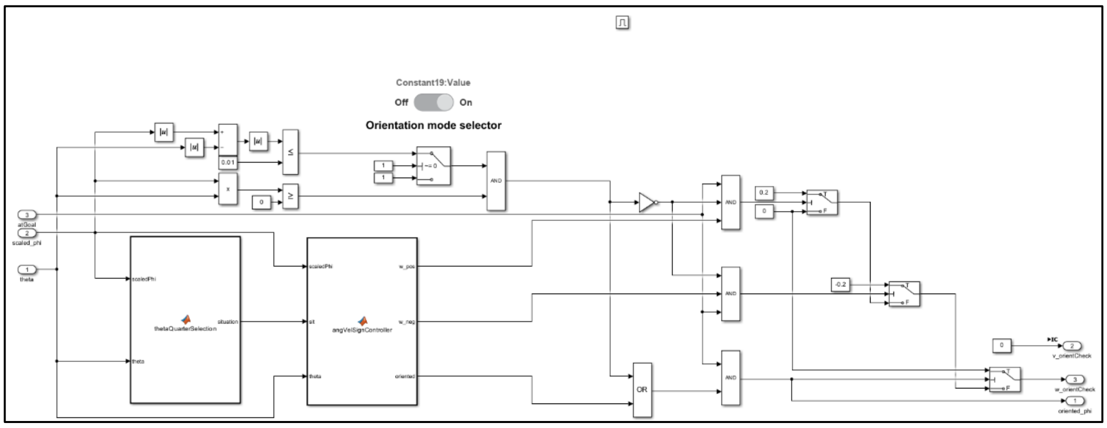


Figure 50: 'Check the departure orientation angle' subsystem.

- Perpendicular orientation at stations and velocity regulation: This subsystem further divided into two subsystems: ‘Perpendicular at stations’ and ‘Regulation velocity at goal’, respectively shown in *Figure 52* and *Figure 53*.
    - Perpendicular at stations: As described in ‘check the distance to goal’ subsystem section, specifically regarding the AtStnRange signal, when this signal is true, the 'Perpendicular at stations' block controls the difference between the current robot orientation (scaled_phi) and the reference orientation value of the goal station (checkpoint_angle). To achieve this, the linear velocity is set to zero, and the angular velocity is set to a moderate value. The sign of the angular velocity is determined as follows: if scaled_phi is greater than checkpoint_angle, the angular velocity is negative; if scaled_phi is lower than checkpoint_angle, it is positive. If scaled_phi is equal to checkpoint_angle, the angular velocity is set to zero, and the output signal ‘perpend’ is set to 1.
    The Matlab function setSignLinVel (see Appendix 11.3.5) is utilized to implement the reverse motion once ‘perpend’ becomes equal to 1.
    - Regulation velocity at goal: This subsystem consists of a linear velocity controller. It operates based on signals such as atStnRange, atGoal, perpend, and utilizes logic blocks to regulate the linear velocity obtained from the pure pursuit controller. If the robot needs to be oriented or is at the station position, the linear velocity is set to zero. Alternatively, if the robot is approaching the next goal position, the linear velocity is modulated by scaling it using a signal called ‘distToGoal’. This signal provides the distance to the next goal, with values saturated between 0.2 and 1. Consequently, as the robot gets closer to the new target position, the linear velocity gradually decreases.

The entire subsystem is enabled when the value of requestOrientation is false.

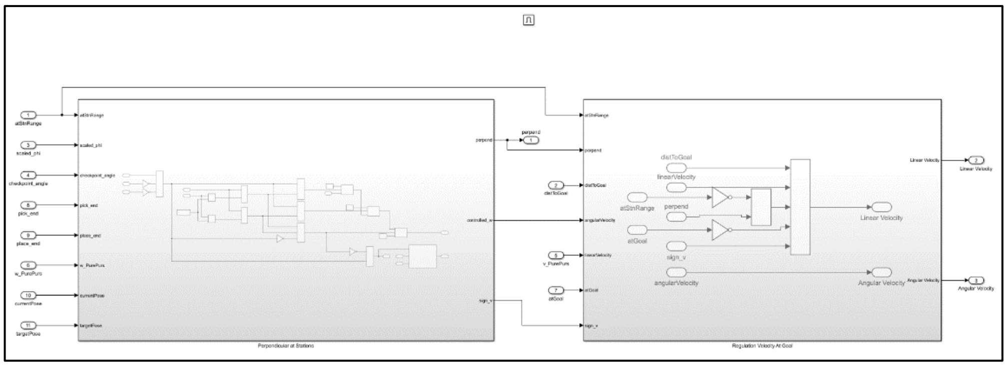
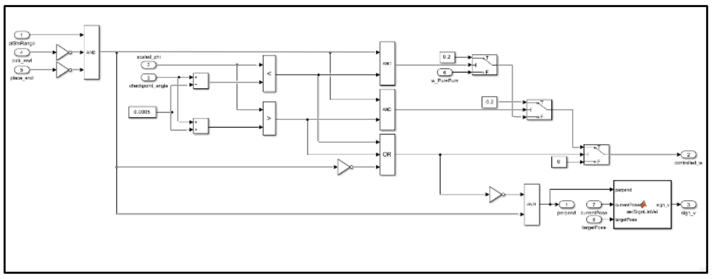


Figure 51: ‘Perpendicular orientation at stations and velocity regulation’ subsystem in Simulink.



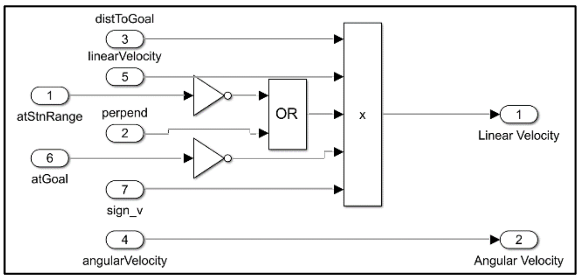


Figure 52: ‘Perpendicular at stations’ subsystem.

Figure 53: ‘Regulation velocity at goal’ subsystem.

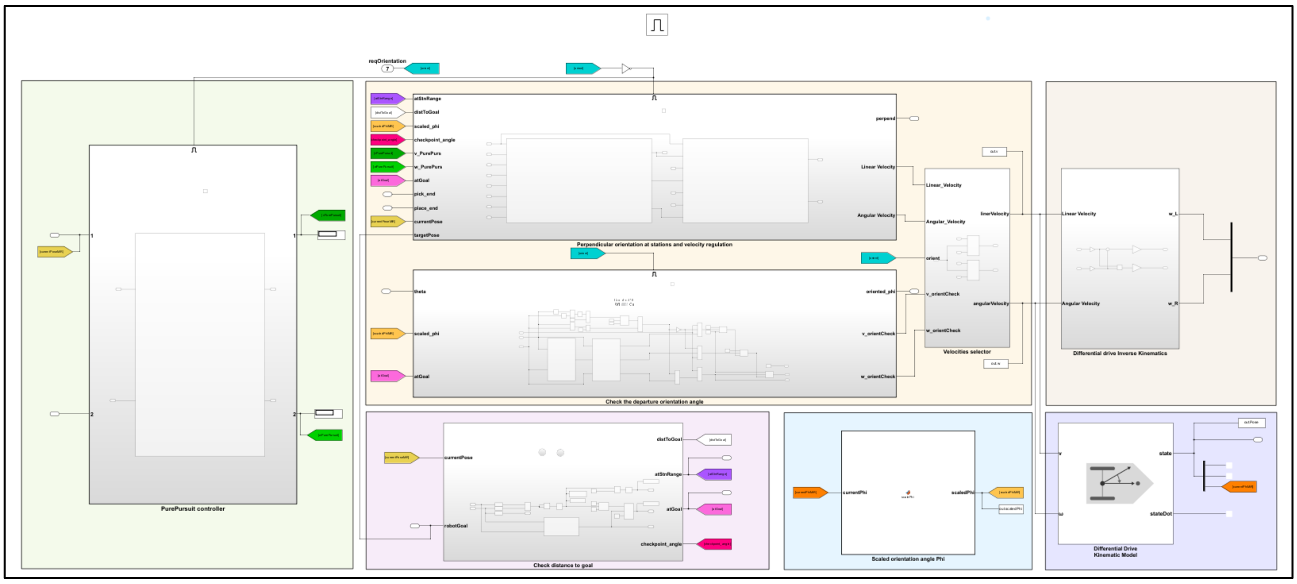


Figure 54: 'MiR_trajectoryPlanning&Controller' subsystem in Simulink.

5.2.3 UR3 and Gripper controller

The robot arm scheduler manages UR and gripper actions based on the task signal at the input. Two sets of waypoints, UR3_task1 and UR3_task2, are provided as reference for pick and place tasks. These sets, defined in the Matlab script related to UR3 settings, correspond to the values inserted in the UR3 programs developed on the teach pendant.

The purpose of this scheduler is to ensure that the current joint configuration of the manipulator and the gripper status, provided as feedback signals, align with the current reference values within an acceptable threshold. To achieve this, two functions are defined: 'UR3goalReached' and 'GripperGoalReached' (see Appendix 11.3.6). The actual waypoints and gripper status are extracted from the set that corresponds to the current task being performed by the UR, whether it's pick or place. The extraction is organized using an index variable 'i', which increments its value each time both the UR and gripper reach the current reference configuration and status.

Within the scheduler, three main states are defined: Arm_Home_Pose, Arm_Pick, and Arm_Place. Before entering these states, a check is performed on the reference gripper value, assigning a flag signal 'closeGripper'. If the closeGripper flag is equal to zero, it indicates that the gripper needs to be opened, otherwise it needs to be closed (see *Figure 55*).

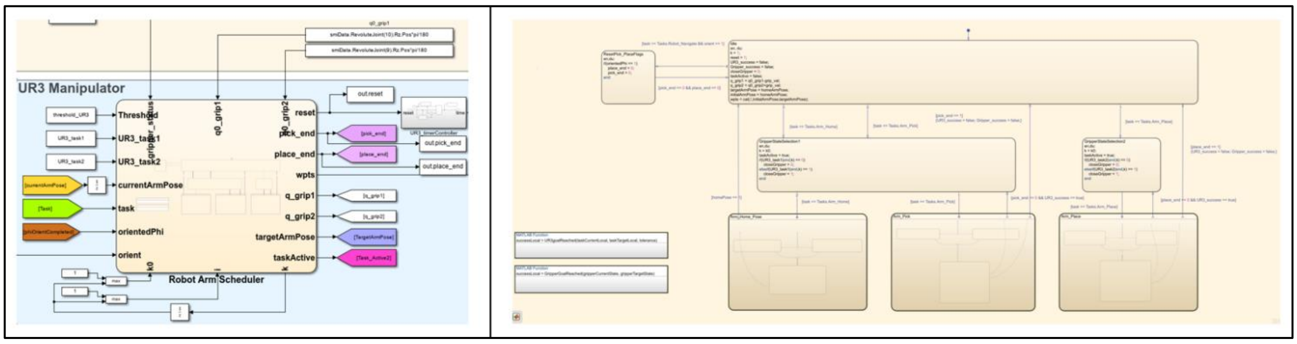


Figure 55: UR and Gripper scheduler.

The UR3 configuration controller consists of a trapezoidal velocity profile trajectory and a torque controller (see *Figure 56*). The trapezoidal velocity profile trajectory takes the following variables as inputs: Time, Waypoints, and EndTime.

- Waypoints: It is a 6x2 matrix, where the number of rows corresponds to the number of joints of the UR3, and the columns represent the current joints configuration and the reference configuration provided by the scheduler, respectively.
- Time: It indicates the time point along the trajectory. This value is obtained from the 'UR3_timerController', which resets the simulation timer each time a manipulator motion is successfully completed.
- EndTime: It is determined by a switch that selects between two different duration times. These durations correspond to the time at which the UR needs to reach the reference configuration, given by the second column of the waypoints matrix. The use of two different end times is due to the different completion times for the pick task and the place task. In this project, the end times are approximated by dividing the total duration time by the number of waypoints set for each task.

The torque controller is implemented based on the inverse dynamic equation, in order to obtain the torque to actuate the non-fixed revolute joints od the Simscape model. The inverse dynamic equation for this model is $\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q)$, where $q, \dot{q}$ and $\ddot{q}$ are respectively the current joints position, velocity and acceleration provided by the manipulator as feedback inputs to the controller. $M(q)$ is the joint space mass matrix, $C(q,\dot{q})\dot{q}$ is the velocity product torque, and $G(q)$ is the gravity torque. These variables are obtained using specific blocks available in the Robotics System Toolbox library. The torque controller equation is: $\tau = M(q)(\ddot{q_d} - K_d\dot{q_e} - K_p q_e) + C(q,\dot{q})\dot{q} + G(q)$, where $q_e = q - q_d$ is the error position and $\dot{q_e} = \dot{q} - \dot{q_d}$ is the velocity error. The variables q_d, $\dot{q_d}$, and $\ddot{q_d}$ correspond to the position, velocity, and acceleration generated by the trapezoidal velocity profile trajectory. By appropriately selecting the derivative and proportional gain coefficients (K) of the second-order differential equation, it is possible to achieve a close-to-zero error convergence.

This controller is taken from the example 'Perform Safe Trajectory Tracking Control Using Robotics Manipulator Blocks' available in the MathWorks website[14].

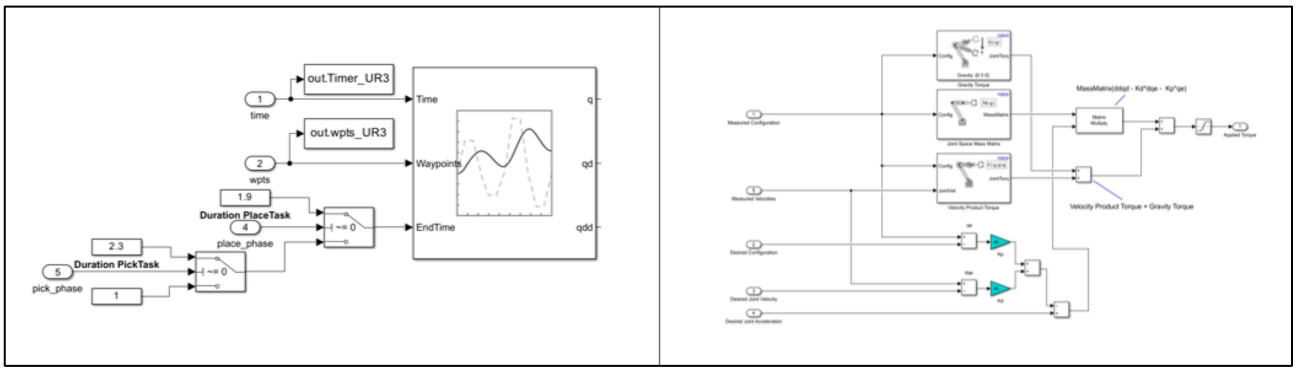


Figure 56: Trapezoidal velocity profile trajectory and Torque controller.

[14] Torque controller: Perform Safe Trajectory Tracking Control Using Robotics Manipulator Blocks - MATLAB & Simulink - MathWorks Italia

The gripper status logic is generated by comparing the actual values of each finger (gripper_status_1 and gripper_status_2), obtained from the Simscape model of the RG2 gripper, with the reference value corresponding to the closed status. If the difference is less than $2\pi/180$, the ‘gripper_status’ signal is set to 1; otherwise, it is set to 0.

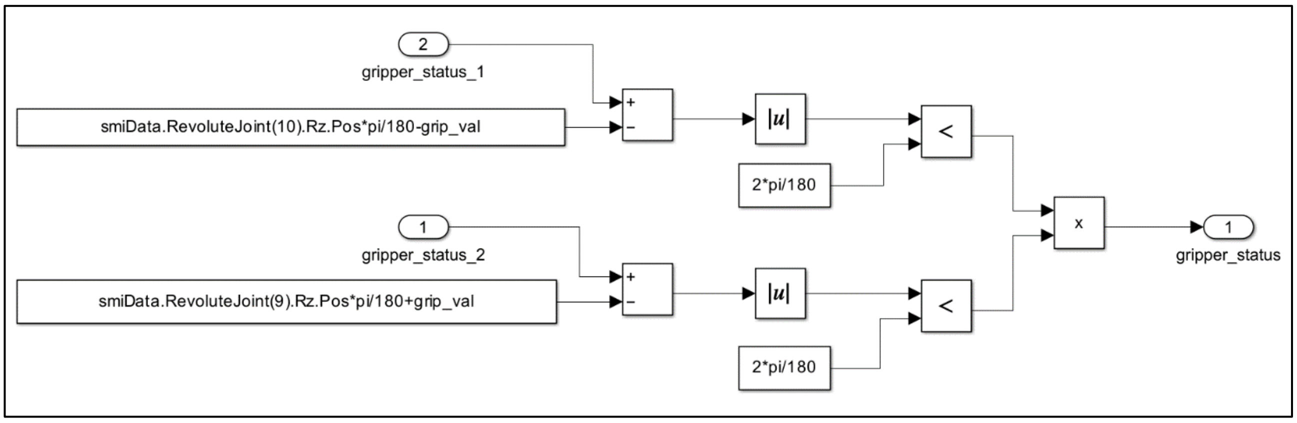

2
gripper_status_1
smiData.RevoluteJoint(10).Rz.Pos*pi/180-grip_val
+
-
|u|
2*pi/180
<
1
gripper_status_2
smiData.RevoluteJoint(9).Rz.Pos*pi/180+grip_val
+
-
|u|
2*pi/180
<
x
1
gripper_status

Figure 57: Gripper status flag subsystem.

5.3 SIMSCAPE MODELS

Simscape Multibody enables the creation of 3D mechanical models for physical systems within the Simulink environment. With Simscape, it is possible to use interconnected physical connections that seamlessly integrate with block diagrams and other modelling approaches. Moreover, Simscape allows for the importation of CAD assemblies, including masses, inertias, constraints, and contact forces between surfaces. This facilitates the creation of a comprehensive 3D animation of the model, which incorporates dynamic elements[15]. For the development of this DM, CAD files created with SolidWorks software, which were already available in the laboratory's database, are utilized [28]. These files are employed for constructing the 3D model of the simulation environment, specifically for modelling the tables used as designated stations for the pick and place task performed by the MOMA, as well as for creating the UR3 manipulator, and the OnRobot RG2 gripper.

The standard set of Simscape blocks required to initialize a Simscape Multibody project is illustrated below.

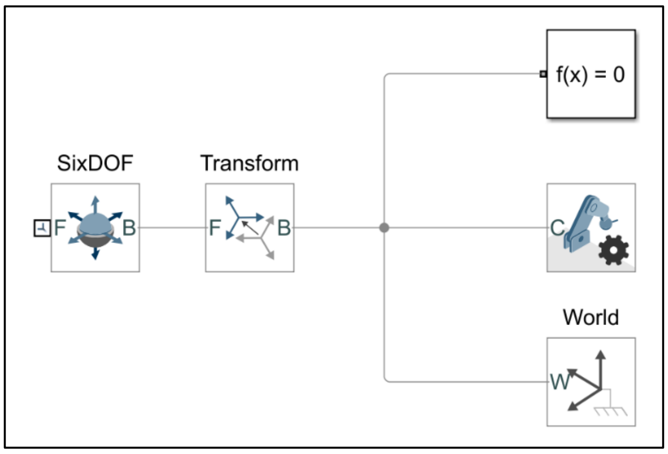


Figure 58: Standard set of Simscape blocks to initialize a Simscape Multibody project.

[15] Simscape Multibody: Simscape Multibody - MATLAB (mathworks.com)

- World block: It represents the reference frame with respect to the entire Simscape system moves.
- Mechanical Configuration: It sets mechanical and simulation parameters to the models connected to 'C' port. Specifically, it provides uniform gravity force and the linearization delta, which is utilized to compute numerical partial derivatives for linearization.
- Rigid Transform: It defines a fixed 3D rigid transformation between Base (B) and Follower (F) frames. It allows for setting the rotation and translation parameters. The specific block shown in Figure 58 is designed to position the mobile robot at the specified starting position.
- Six dof joint: This joint consists of three translational and three rotational degrees of freedom. It enables unconstrained 3D translation and rotation. Specifically, the follower frame initially undergoes translation with respect to the base frame, followed by independent rotational motion around the origin of the follower frame. The direction of the joint is determined by the movement of the follower frame in relation to the base frame.

5.3.1 Laboratory environment and MiR100

The laboratory 3D model consists of three stations that participate in the pick and place task, as well as the floor, columns, and two walls that define the boundaries of the laboratory space. Within this Simscape subsystem, the 'spatial contact force' block is utilized to include the dynamic forces that govern the contact between rigid body elements. For instance, it enables the contact between the MoMa wheels and the ground. The Laboratory model is placed in according to the binary occupancy map system.

The MiR100 3D model is implemented using the differential drive Simscape model from the Mobile Robotics Simulation Toolbox. To customize and enhance the model, the library associated with this block is disabled, allowing for the addition of other blocks. The developed model represents an approximation of the mobile robot by appropriately setting the values of geometry parameters.

These two models are shown in the figures below.

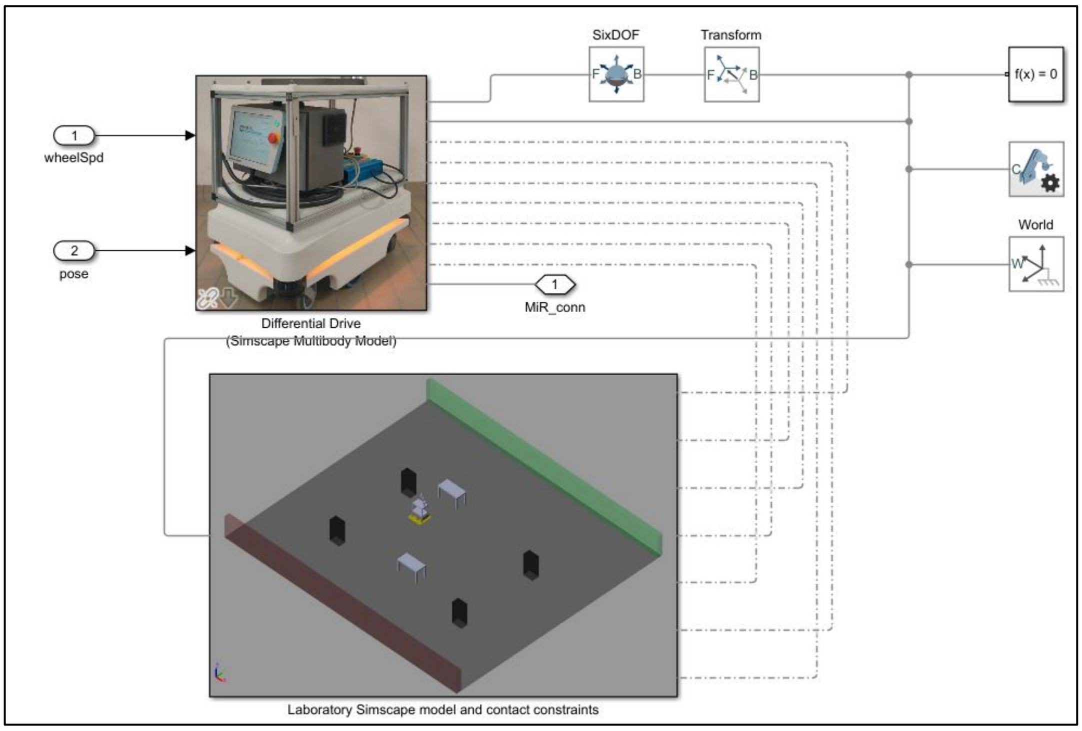

SixDOF
Transform
f(x) = 0
1
wheelSpd
2
pose
World
1
MiR_conn
Differential Drive
(Simscape Multibody Model)
Laboratory Simscape model and contact constraints

Figure 59: Simscape connections between the Laboratory and MiR100 models.

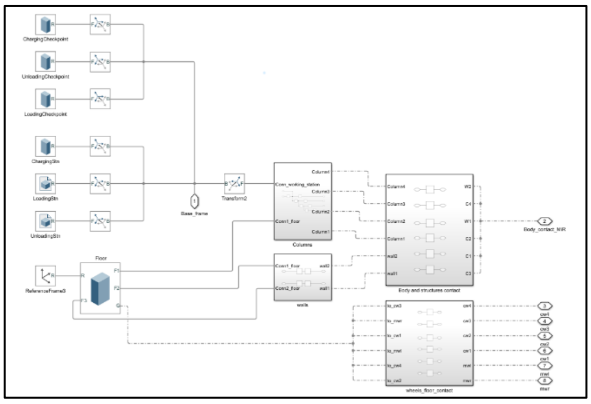


Figure 61: Laboratory Simscape Multibody model.

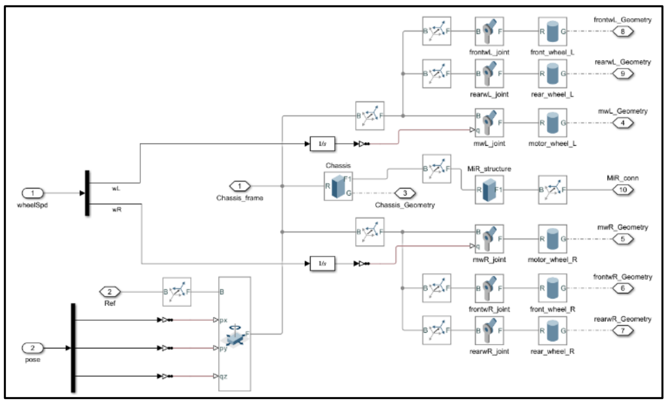


Figure 60: MiR100 Simscape Multibody model.

5.3.2 UR3 and RG2 Gripper

The UR3 Simscape Multibody 3D model is built upon the previously developed model in the laboratory and the model generated by the Matlab function Smimport[16]. This function takes the robot imported from the URDF models provided by the Robotics System Toolbox as input and generates a Simscape Multibody model. The model is subsequently customized to align with the other models that comprise the MoMa system and to accommodate the developed task.

The RG2 gripper Simscape Multibody 3D model is identical to the previously developed model in the laboratory [28].

The two models are illustrated in the figures below.

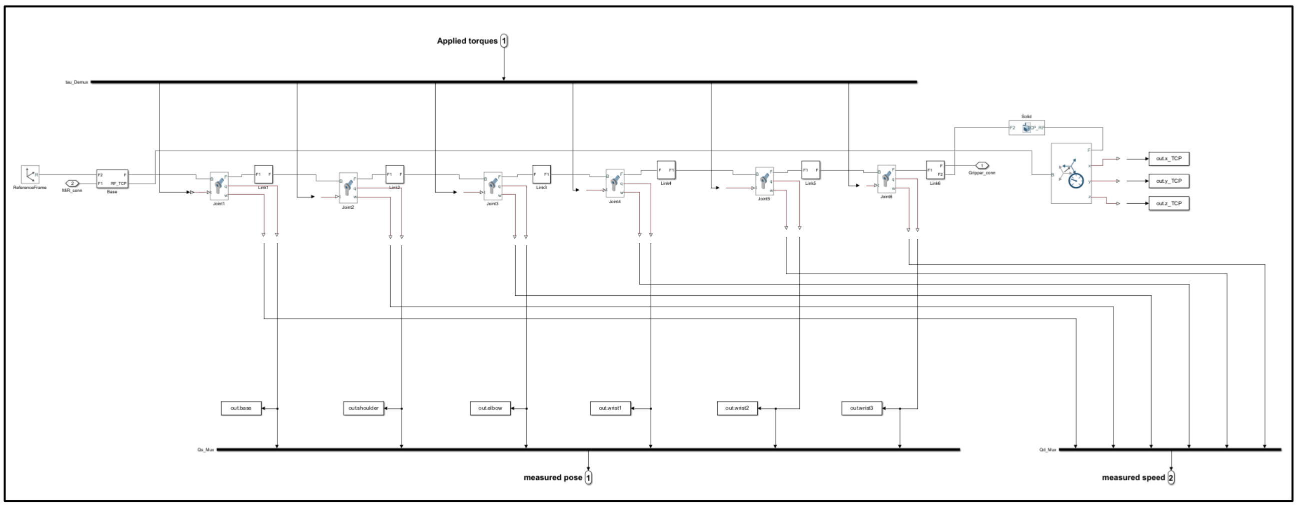


Figure 62: UR3 Simscape Multibody model.

[16] Smimport Matlab function: Import a CAD, URDF, or Robotics System Toolbox model - MATLAB smimport - MathWorks Italia

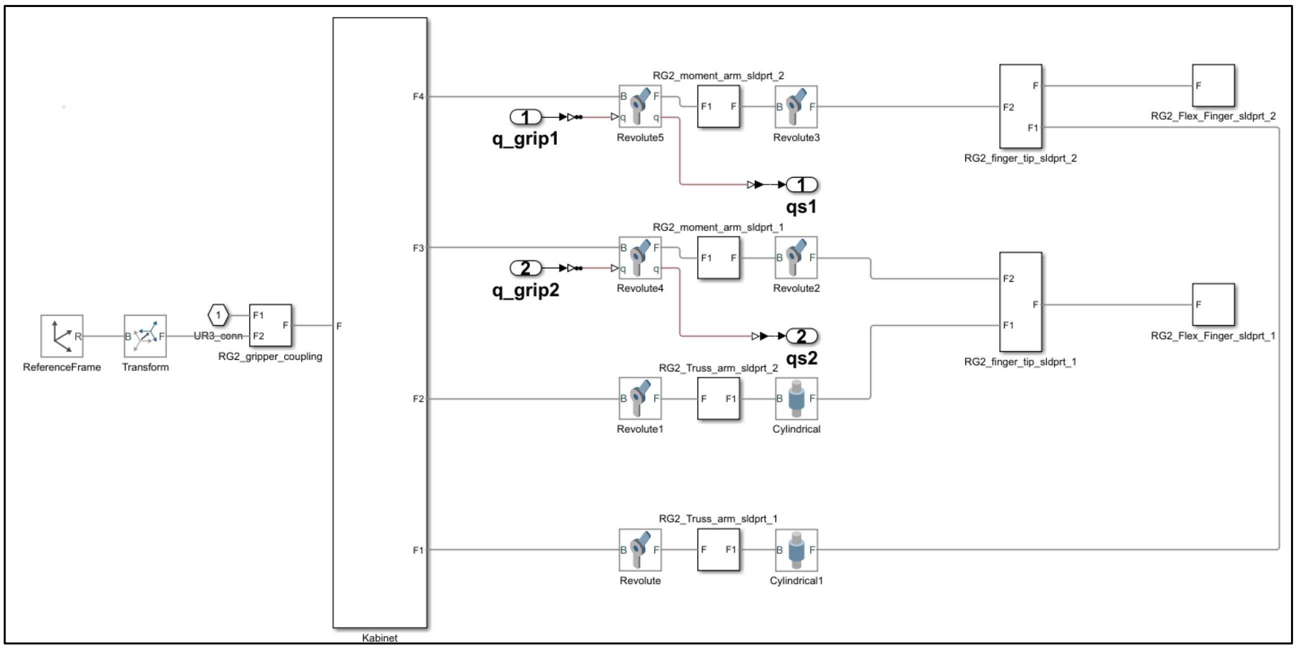


Figure 63: RG2 Gripper Simscape Multibody model.

5.3.3 Visualization of the Digital Model developed in Simscape Multibody

Below are some figures showcasing the implemented virtual 3D simulation environment. These images are captured from the Mechanics Explorer tool, which facilitates the visualization of multibody models.

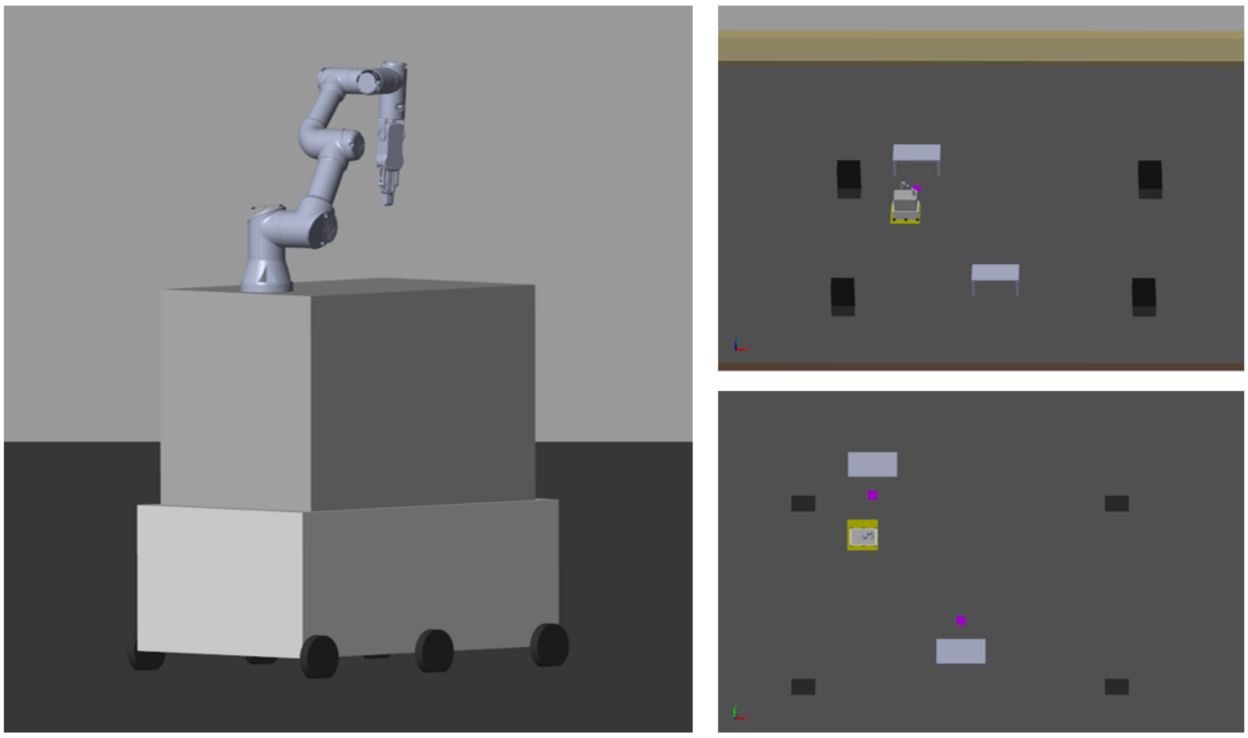


Figure 64: View of the digital model in Mechanics Explorer.

5.4 Results

As described in the dedicated case study chapter (Chapter 3), below are presented the results of two versions of the same task.

5.4.1 Case study: Orientation angles not specified

5.4.1.1 General settings and considerations

For the development of this version of the case study, the following parameters and error thresholds are set.

Table 14: Digital Model settings and threshold in Matlab and Simulink.

MiR100 PARAMETERS	
Pure Pursuit controller	
Desired Linear Velocity [m/s]	1.1
Maximum Angular Velocity [rad/s]	1
Lookahead distance [m]	0.4
Orientation phase threshold	
Maximum difference between angles $\Delta(\phi_{depart})$ [rad]	0.8
Orientation at goal station	
Maximum distance between angles at goal station Δ(ϕ) [rad]	0.0005 (= 0.029°)
AtStnRange thresholds	
Maximum distance between X coordinates Δ(X) [m]	0.1
Maximum distance between Y coordinates Δ(Y) [m]	0.3
AtGoal thresholds	
Maximum distance Δ(X,Y) [m]	0.1
UR3 PARAMETERS	
Trapezoidal Velocity Profile trajectory	
EndTime Pick task [s]	2.3
EndTime Place task [s]	1.9
UR3goalReached function	
UR3 configuration threshold [rad]	0.015

Since the data is obtained from different acquisition systems and processed on various machines or tools, an attempt at synchronization is made for the creation of the following charts. This is achieved by carefully observing the data and adjusting the time scale to ensure that the graphical representation for all systems roughly begins at the same moment in time (see *Figure 69*).

In this specific version of the case study, since the orientation angles at which the mobile robot departs from the stations within the created program are not specified, an attempt is made to approximate the algorithm used by the MiR to establish its orientation when moving from one station to the next in the implementation of this DM. This orientation angle is computed as the angle between the reference x-axis and the ideal trajectory joining the two stations (see *Figure 65*).

The procedure followed in Matlab is illustrated with the calculations that follow:

$$\theta_1 = \text{atan2}\left((LoadingStn(2) - StartingPoint(2)), (LoadingStn(1) - StartingPoint(1))\right)$$

$$\theta_2 = \text{atan2}\left((UnloadingStn(2) - LoadingStn(2)), (UnloadingStn(1) - LoadingStn(1))\right)$$

$$\theta_3 = \text{atan2}\left((ChargingStn(2) - UnloadingStn(2)), (ChargingStn(1) - UnloadingStn(1))\right)$$

$$\theta = [1.326, -0.955;\ 2.427] rad$$

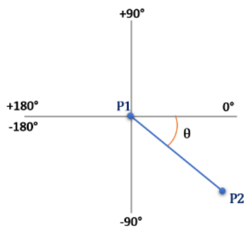


Figure 65: Orientation angle between two stations.

Based on the values obtained for the theta vector, by appropriately adjusting the 'orientation phase threshold' parameter (see *Table 14*), referred to the *Figure 50*, it is possible to achieve a realistic simulation of the trajectory followed by MoMa, accurately sampled using ROS.
The simulation of the trajectory is also affected by the parameters of the pure pursuit controller. Due to the difficulty in accurately determining the MiR's actual velocities, they are approximated by referring to the real-time ROS values, as shown in *Figure 66*.

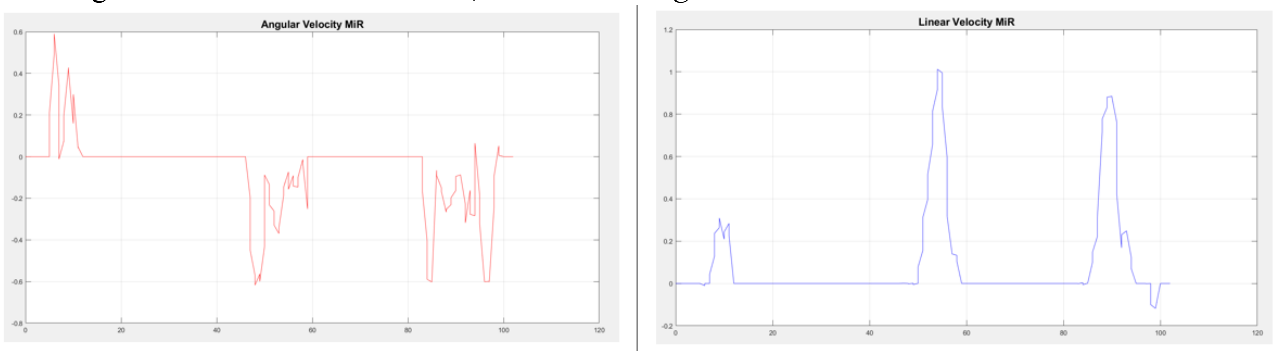


Figure 66: Angular and Linear velocities values acquired with ROS in Matlab.

The *Figure 67* shows the UR3 and UR_TCP reference systems respectively in Optitrack, real-world, and Matlab (Digital Model). These figures are useful to obtain the charts illustrated in section 5.4.1.4.

It is important to note that evaluating the UR_TCP data with respect to the UR base is functional for analysing the accuracy of the Digital Model, considering it as if it were not part of a MoMa but rather stable in a fixed position. This choice is driven by the fact that the MoMa, being affected by the position error of the MiR, inevitably leads to a significant error in the UR_TCP performance as well. Reproducing this situation could help quantify these errors. However, in a working environment with the characteristics of Industry 4.0, advanced technologies are available. The use of a wrist-mounted camera integrated into the manipulator would avoid this situation by accurately detecting the object's position and allowing the UR to successfully complete its task.

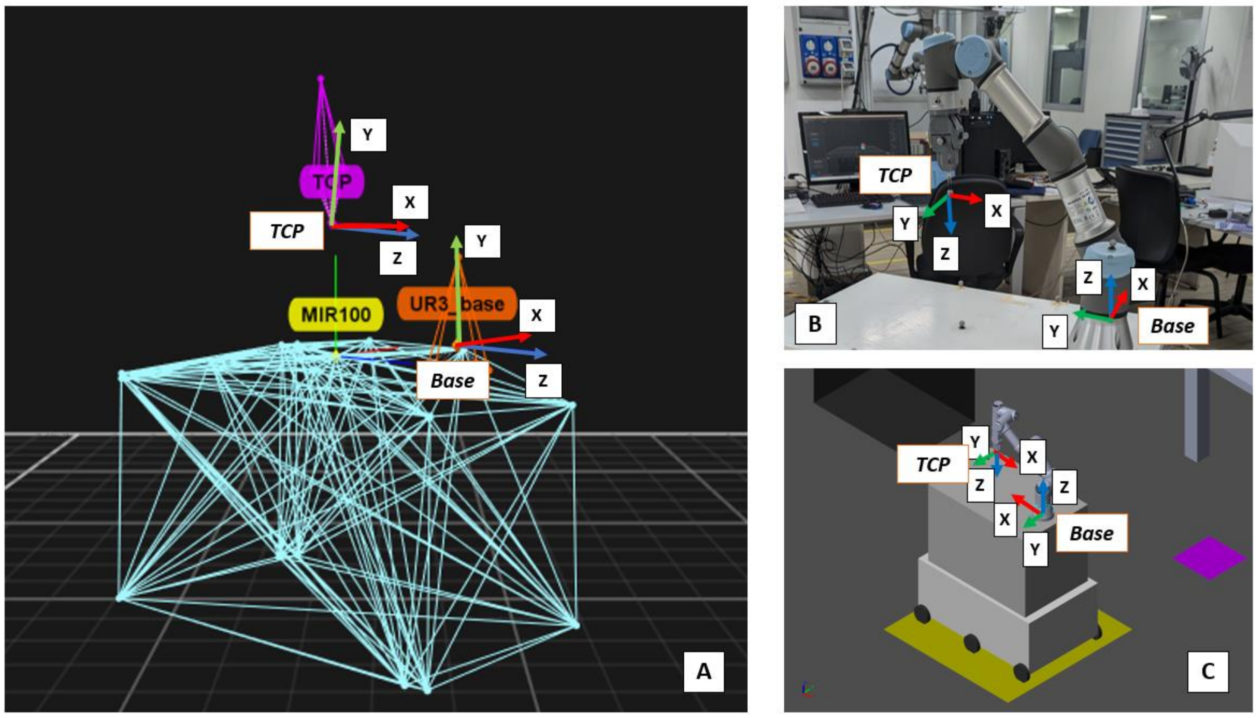


Figure 67: UR3 base and TCP reference systems in different environments.

5.4.1.2 *MiR100 results*

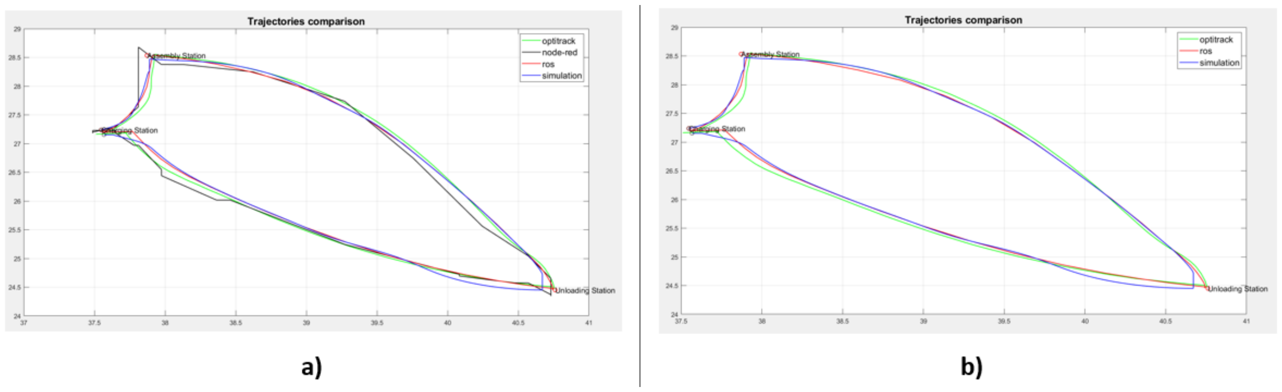


Figure 68: Trajectories of the different systems: a) with Node-RED, b) without Node-RED.

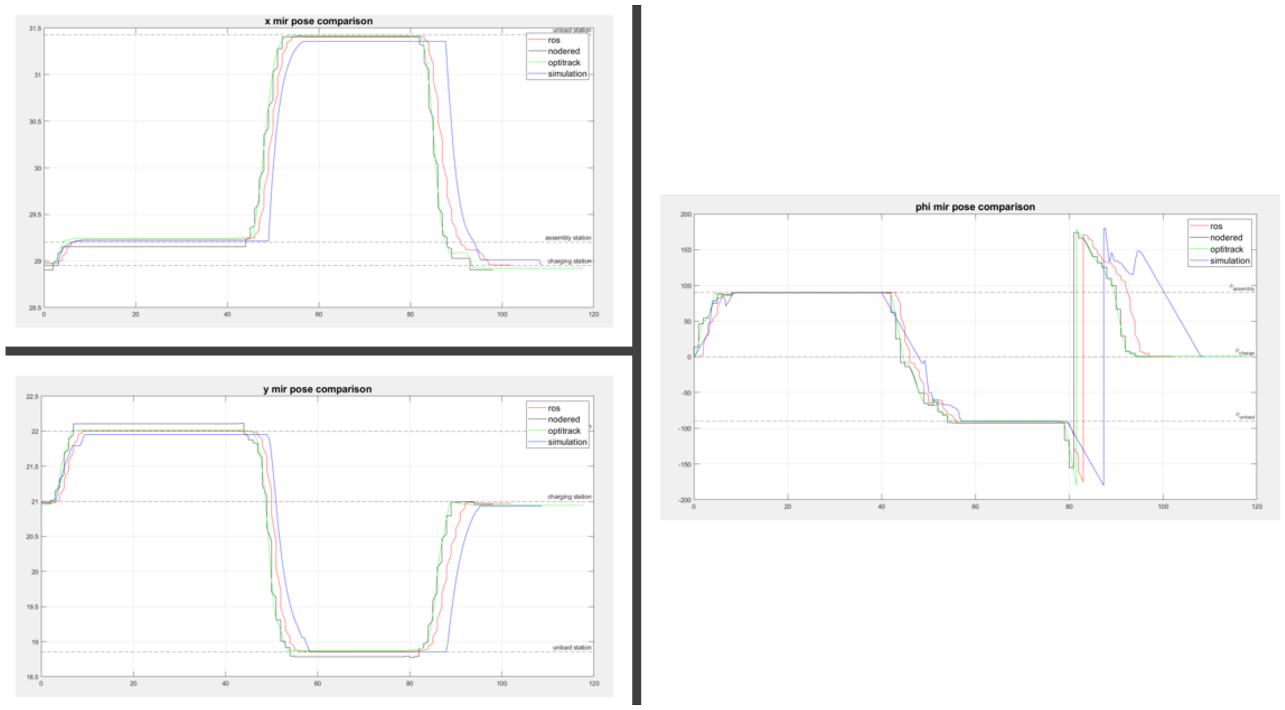


Figure 69: Position and orientation time plots for each system.

The waypoints provided as input to the pure pursuit controller are based on the ROS data. Therefore, the ROS data is utilized as the reference trajectory for the DM. As depicted in *Figure 68a*, the simulation model closely resembles the reference trajectory, except for the Unloading station where the distance is larger. The Optitrack system captures the actual trajectory followed by the physical MoMa, and it is used exclusively for comparison purposes. However, as demonstrated by the repeatability test, ROS exhibits a closer correlation with the Optitrack system, and the distance between them, also in this simulation, aligns with the repeatability test results. In contrast, Node-RED has been determined to be the least precise system based on the considerations discussed in previous chapters. It is probable that this system would be more suitable and accurate for acquiring data from long period processes that extend beyond approximately two minutes, as demonstrated in this case where a more variable trajectory is evident.

Figure 69 displays the time-based graphs of the position and orientation of the mobile robot for each acquisition system. These graphs visually depict the position errors compared to the reference values taken from the map created by the MiR, as well as the behaviour of the DM relative to the reference

trajectory provided by ROS. It can be observed that the DM's behaviour is fairly aligned with that of the ROS system. The slight time shift is related to the velocity values specified in the pure pursuit block in Simulink. However, as mentioned in subsection 5.4.1.1, the precise values are unknown, and an approximation is provided to achieve the most accurate simulation possible. The implemented model allows for linear velocity variations up to 1.5m/s, which is the maximum value provided by the datasheet, or even higher to better match the DM's trajectory with the reference and obtain a more accurate representation of the results. The most important aspect, regardless of time, is that the proposed DM successfully reaches all positions within the specified error thresholds. Several tests have been conducted to further reduce the threshold values. However, currently, for the developed model, the ones listed in *Table 14* represent the best solution.
In this regard, as can be observed from *Table 18*, the error values obtained for the DM during the simulation of this specific case are well within the thresholds reported in *Table 14*. Additionally, the errors on the DM's position are consistent or only slightly higher than the accuracy provided by the datasheet of the MiR100.

Regarding orientation, the behaviour of the DM appears to be more variable but exhibits excellent response at the stations, where the error is minimal. This performance is also influenced by the lack of orientation angle indications in this version of the case study, making the model more variable.

Table 15: Node-RED position errors.

NODE-RED							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.153	22.106	90.021	0.047	0.106	0.021	0.116
Unloading Stn	31.404	18.785	-92.444	0.023	0.067	2.444	0.071
Charging Stn	28.904	20.957	0.213	0.046	0.043	0.213	0.063

Table 16: ROS position errors.

ROS							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.231	22.005	90.361	0.031	0.005	0.361	0.032
Unloading Stn	31.410	18.861	-92.146	0.017	0.009	2.146	0.019
Charging Stn	28.958	20.971	0.213	0.008	0.029	0.213	0.030

Table 17: Optitrack position errors.

OPTITRACK							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.243	22.017	88.734	0.043	0.017	1.266	0.046
Unloading Stn	31.421	18.874	-89.326	0.006	0.022	0.674	0.023
Charging Stn	28.922	20.942	1.288	0.028	0.058	1.288	0.064

Table 18: Digital Model position errors.

SIMULATION (Digital Model)							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.214	21.952	89.971	0.014	0.048	0.029	0.050
Unloading Stn	31.357	18.852	-89.971	0.070	0.001	0.029	0.070
Charging Stn	28.972	20.935	0.029	0.022	0.065	0.029	0.069

5.4.1.3 UR3 results

The analysis of UR3 results is performed by comparing the data related to the joints configurations during the execution of the programmed pick and place task. However, the only available data for the manipulator are those acquired through Node-RED and the simulation of the Digital Model. Obtaining the configurations of the revolute joints in radians using the real Optitrack reference system is extremely complicated. Therefore, the analysis of the manipulator's results is incomplete, but it still serves the purpose of visualizing the behaviour of the Digital Model in comparison to the data collected with Node-RED. Furthermore, the ROS system is not available for UR and the UR_TCP, as collecting the data is more complex compared to Node-RED.

Figure 70 presents the charts depicting the DM joint configurations generated by the Simscape model in relation to the reference waypoint provided as inputs to the trapezoidal velocity profile trajectory block in Simulink (see *Figure 56*). These plots also indicate the gripper status, serving to verify whether the closing and opening actions occur correctly in accordance with the programmed UR configuration. The delay in the behaviour of the DM can be attributed to the involvement of a delay block in Simulink, which provides the feedback signal to the UR scheduler, thus avoiding the occurrence of infinite loop errors.

Figure 71 illustrates the comparison between the behaviour of the DM and the actual behaviour represented by the Node-RED acquired data. The DM's performance closely resembles the reference behaviour, despite some inaccuracies. It is worth noting that the majority of errors remain well below the established threshold of 0.015 rad. The highest values are observed in a few instances, specifically in joints 4 and 6, where they reach approximately 0.01rad. However, it is important to note that these errors in the UR joints configuration have an impact on the UR_TCP results, as discussed in the subsequent section.

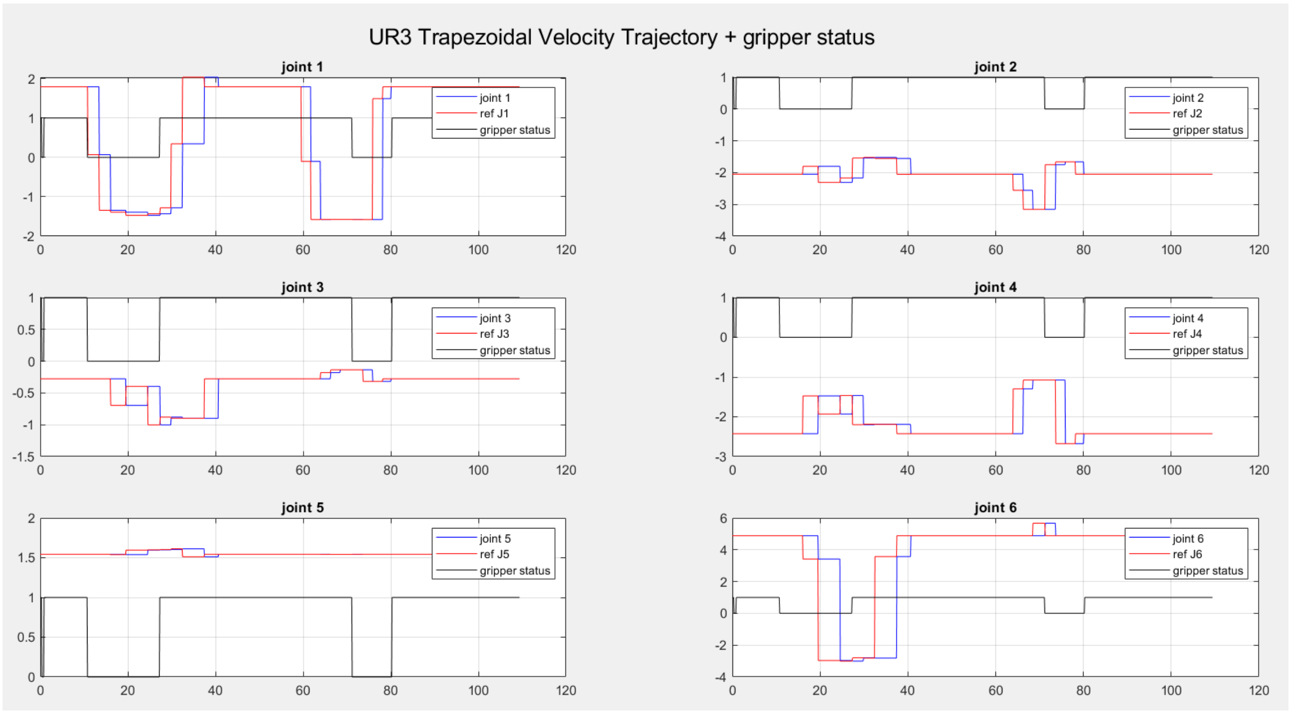


Figure 70: Trapezoidal Velocity Profile Trajectory and Gripper status results.

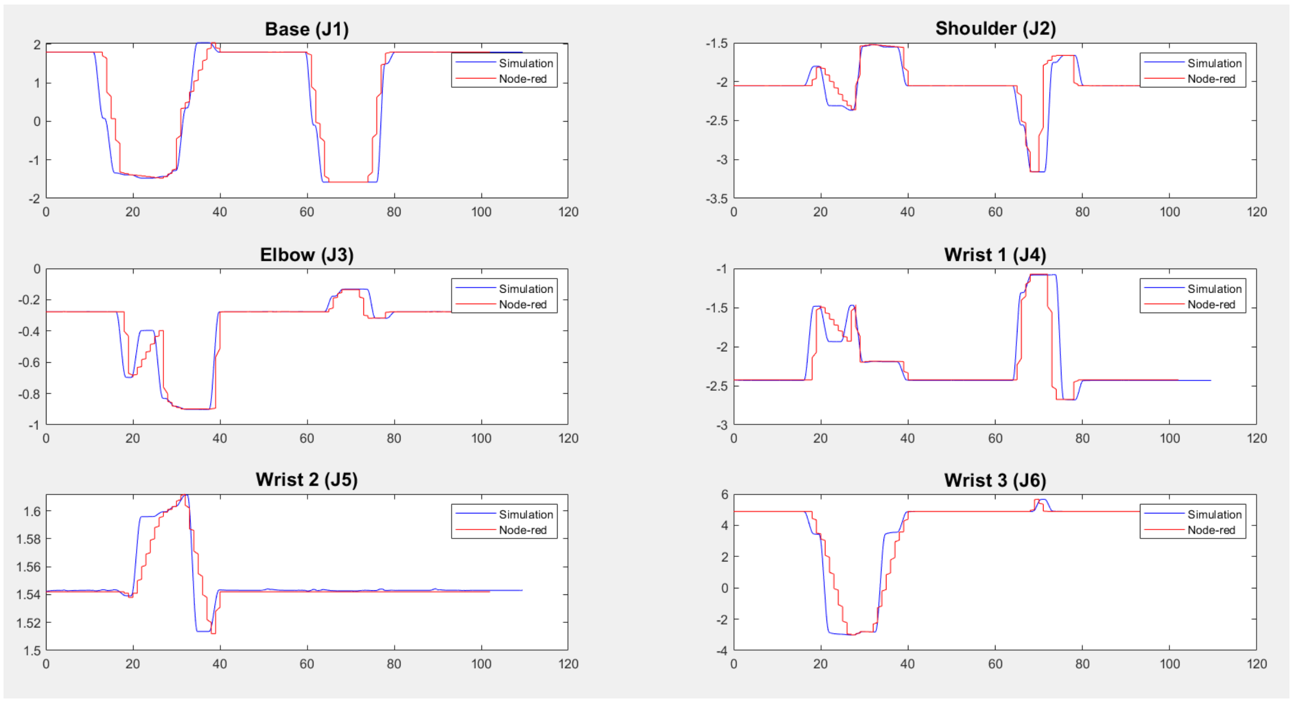


Figure 71: DM joints configurations compared with the Node-RED reference system.

Table 19: UR3 Pick task programmed waypoints from teach pendant.

UR3 PICK TASK PROGRAMMED WAYPOINTS							
Wpts\Joints	**Base [rad]**	**Shoulder [rad]**	**Elbow [rad]**	**Wrist 1 [rad]**	**Wrist 2 [rad]**	**Wrist 3 [rad]**	**Gripper status**
1	1.7935	-2.0515	-0.2770	-2.4262	1.5422	4.8880	0
2	0.0709	-2.0515	-0.2770	-2.4264	1.5422	4.8880	0
3	-1.3440	-2.0515	-0.2770	-2.4264	1.5422	4.8880	0
4	-1.3907	-1.8021	-0.6960	-1.4745	1.5382	3.4153	0
5	-1.4734	-2.3073	-0.3969	-1.9305	1.5958	-2.9723	0
6	-1.4319	-2.1694	-1.0022	-1.4643	1.5992	-3.0010	1
7	-1.2788	-1.5418	-0.8805	-2.1977	1.6033	-2.7946	1
8	0.3461	-1.5240	-0.8981	-2.1865	1.6116	-2.8143	1
9	2.0356	-1.5544	-0.8988	-2.1900	1.5122	3.5704	1
10	1.7937	-2.0555	-0.2768	-2.4264	1.5423	4.8880	1

Table 20: UR3 Pick task DM simulation errors.

UR3 PICK TASK SIMULATION ERRORS						
Wpts\Errors	**Δ(Base) [rad]**	**Δ(Shoulder) [rad]**	**Δ(Elbow) [rad]**	**Δ(Wrist 1) [rad]**	**Δ(Wrist 2) [rad]**	**Δ(Wrist 3) [rad]**
1	0.0004	0.0012	0.0006	0.0062	0.0008	0.0002
2	0.0036	0.0012	0.0005	0.0061	0.0012	0.0001
3	0.0035	0.0012	0.0005	0.0061	0.0012	0.0001
4	0.0001	0.0002	0.0015	0.0076	0.0010	0.0099
5	0.0002	0.0014	0.0006	0.0059	0.0001	0.0101
6	0.0002	0.0015	0.0006	0.0063	0.0000	0.0011
7	0.0007	0.0028	0.0021	0.0004	0.0000	0.0045
8	0.0006	0.0002	0.0029	0.0040	0.0003	0.0005
9	0.0003	0.0004	0.0029	0.0041	0.0014	0.0105
10	0.0006	0.0010	0.0004	0.0061	0.0007	0.0002

Table 21: UR3 Place task programmed waypoints from teach pendant.

UR3 PLACE TASK PROGRAMMED WAYPOINTS							
Wpts\Joints	**Base [rad]**	**Shoulder [rad]**	**Elbow [rad]**	**Wrist 1 [rad]**	**Wrist 2 [rad]**	**Wrist 3 [rad]**	**Gripper status**
1	1.7937	-2.0516	-0.2770	-2.4262	1.5422	4.8880	1
2	-0.1012	-2.0518	-0.2773	-2.4264	1.5422	4.8880	1
3	-1.5780	-2.0518	-0,2773	-2.4264	1.5422	4.8880	1
4	-1.5771	-2.5559	-0.1799	-1.2957	1.5418	4.8881	1
5	-1.5771	-3.1573	-0.1354	-1.0746	1.5418	4.8881	1
6	-1.5771	-3.1573	-0.1354	-1.0746	1.5418	5.6720	0
7	-1.5776	-1.7504	-0.1356	-1.0741	1.5423	4.8883	0
8	-1.5778	-1.6623	-0.3176	-2.6752	1.5420	4.8876	0
9	1.4924	-1.6624	-0.3175	-2.6751	1.5420	4.8876	0
10	1.7937	-2.0516	-0.2770	-2.4262	1.5422	4.8880	0

Table 22: UR3 Place task DM simulation errors.

UR3 PLACE TASK SIMULATION ERRORS						
Wpts\Errors	**Δ(Base) [rad]**	**Δ(Shoulder) [rad]**	**Δ(Elbow) [rad]**	**Δ(Wrist 1) [rad]**	**Δ(Wrist 2) [rad]**	**Δ(Wrist 3) [rad]**
1	0.0006	0.0010	0.0004	0.0061	0.0007	0.0002
2	0.0008	0.0011	0.0005	0.0060	0.0013	0.0001
3	0.0003	0.0010	0.0023	0.0131	0.0009	0.0000
4	0.0000	0.0018	0.0029	0.0097	0.0008	0.0002
5	0.0000	0.0026	0.0033	0.0084	0.0008	0.0086
6	0.0007	0.0020	0.0021	0.0059	0.0007	0.0100
7	0.0005	0.0003	0.0009	0.0013	0.0009	0.0011
8	0.0008	0.0004	0.0007	0.0059	0.0009	0.0005
9	0.0006	0.0010	0.0006	0.0061	0.0009	0.0002
10	0.0008	0.0011	0.0005	0.0060	0.0013	0.0001

5.4.1.4 UR3 TCP results

The UR_TCP data is captured relative to the base joint of the UR3. For result analysis, a comparison is made between the data collected with Node-RED and Optitrack. Similar to the UR3, the ROS system data is unavailable due to the challenges involved in obtaining it compared to Node-RED. The Optitrack detection of the position of the UR_TCP by utilizing a 3D-printed tool that holds a marker at the corresponding RG2 gripper's UR_TCP location. The Optitrack values are referenced to its own coordinate system, therefore, to obtain the UR_TCP data with respect to the UR base joint using Optitrack, the position of the latter is captured. Afterwards, a Matlab script is utilized to calculate the difference between the two sets of data (see Appendix 11.3.7). Furthermore, each system exhibits a varying distance between the UR_TCP and the UR3 wrist3 along the z-axis. This variation is due to different settings in the UR3 programs, Optitrack UR_TCP printed tool, and Simscape CAD. To account for this, an offset is introduced in the Matlab script to align the z-coordinate of the starting point of each system with the Node-RED starting point, which is considered the most accurate system for UR_TCP analysis. This procedure is performed for both the pick and place tasks.

A preliminary overview of the UR_TCP behaviour for both the pick task and the place task is provided by the UR3 teach pendant in the graphics section, as shown in *Figure 72*.

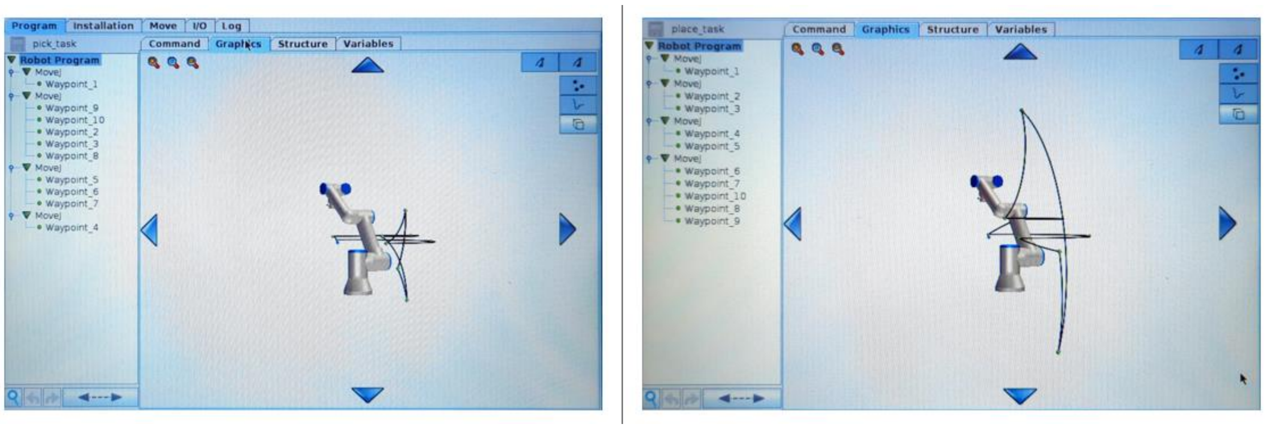


Figure 72: Left side: TCP Pick task graphic view; right side: TCP Place task graphic view.

The results obtained from the three systems for pick and place tasks are depicted in the figures below.

- **Pick task**

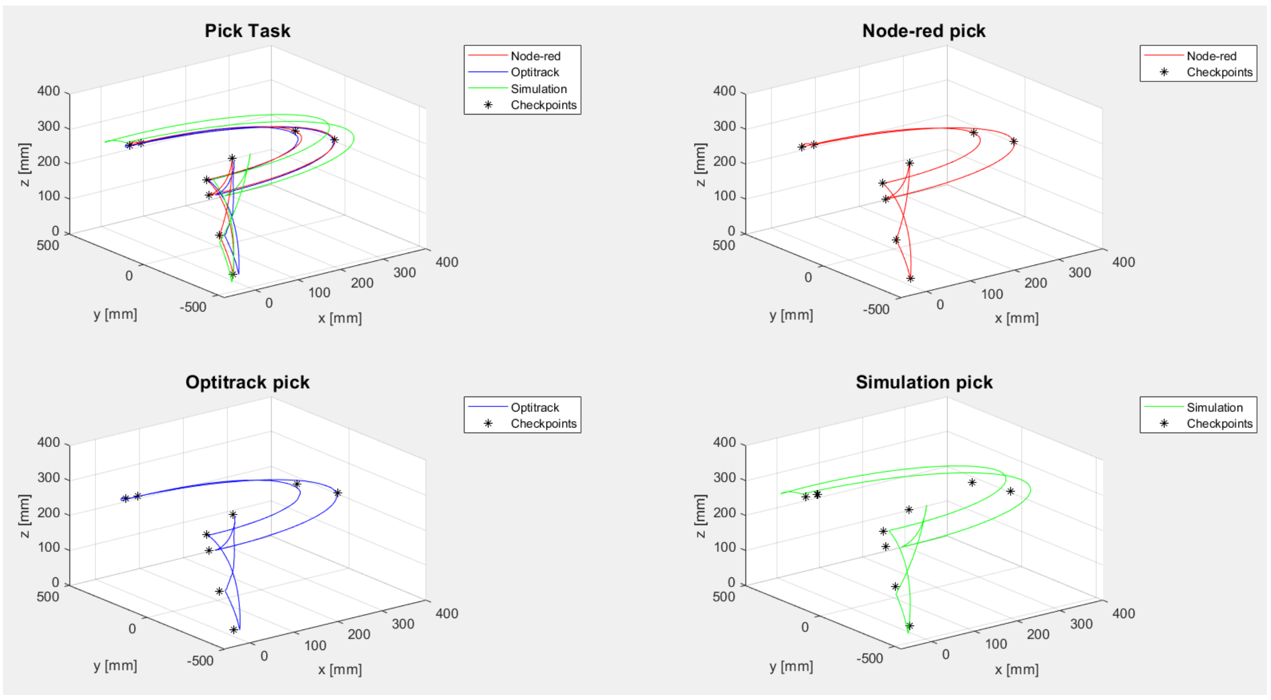


Figure 73: Comparison of the TCP graphs for the Pick task.

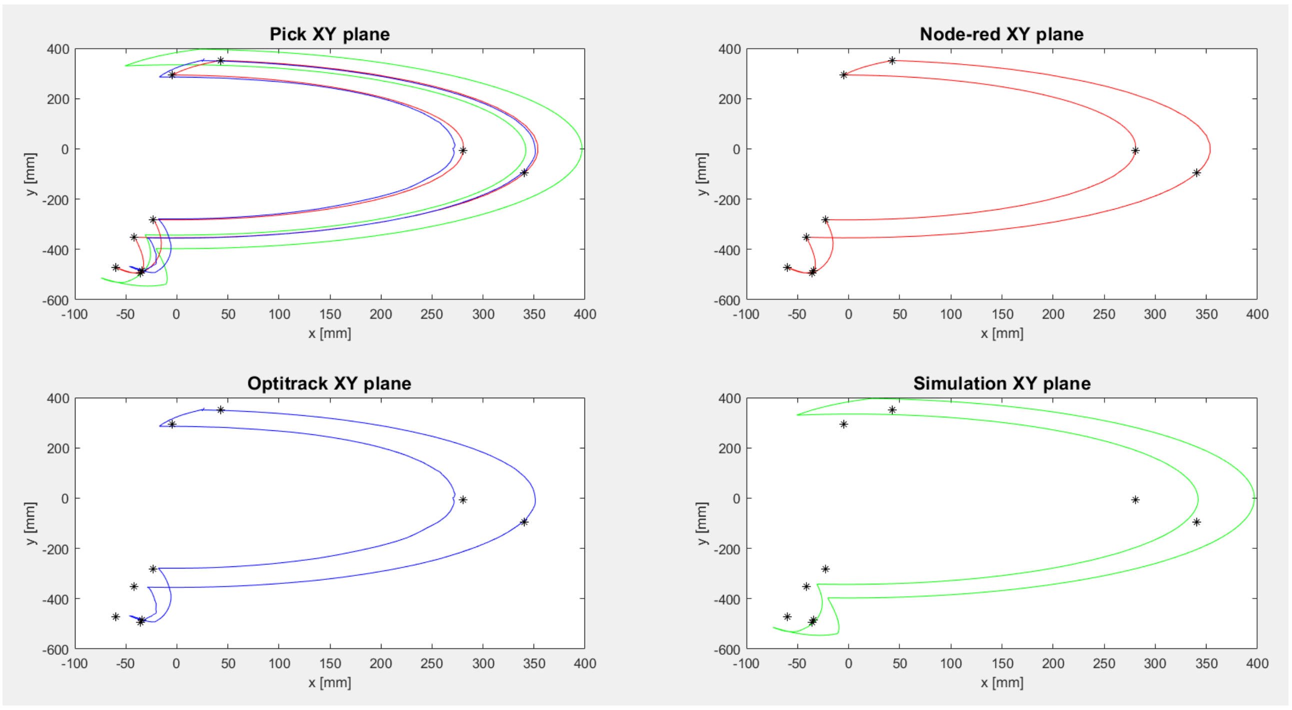


Figure 74: Focus on the XY plane for the Pick task.

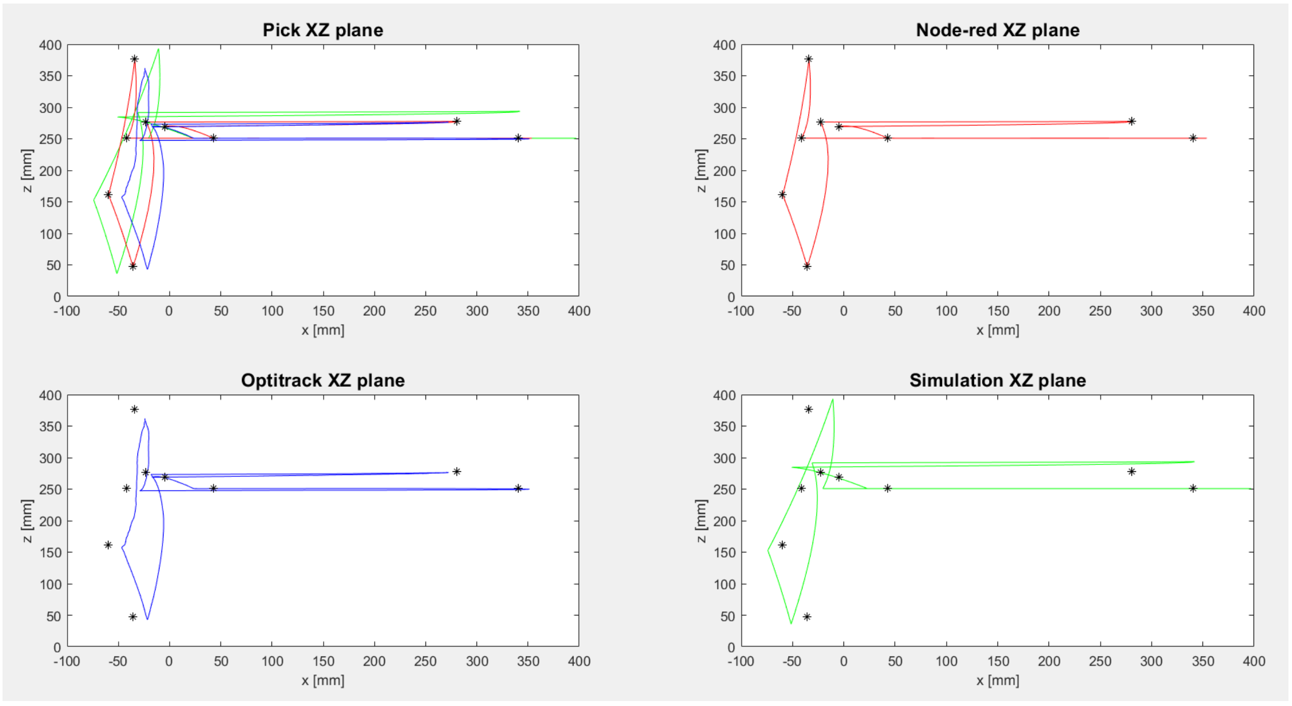


Figure 75: Focus on the XZ plane for the Pick task.

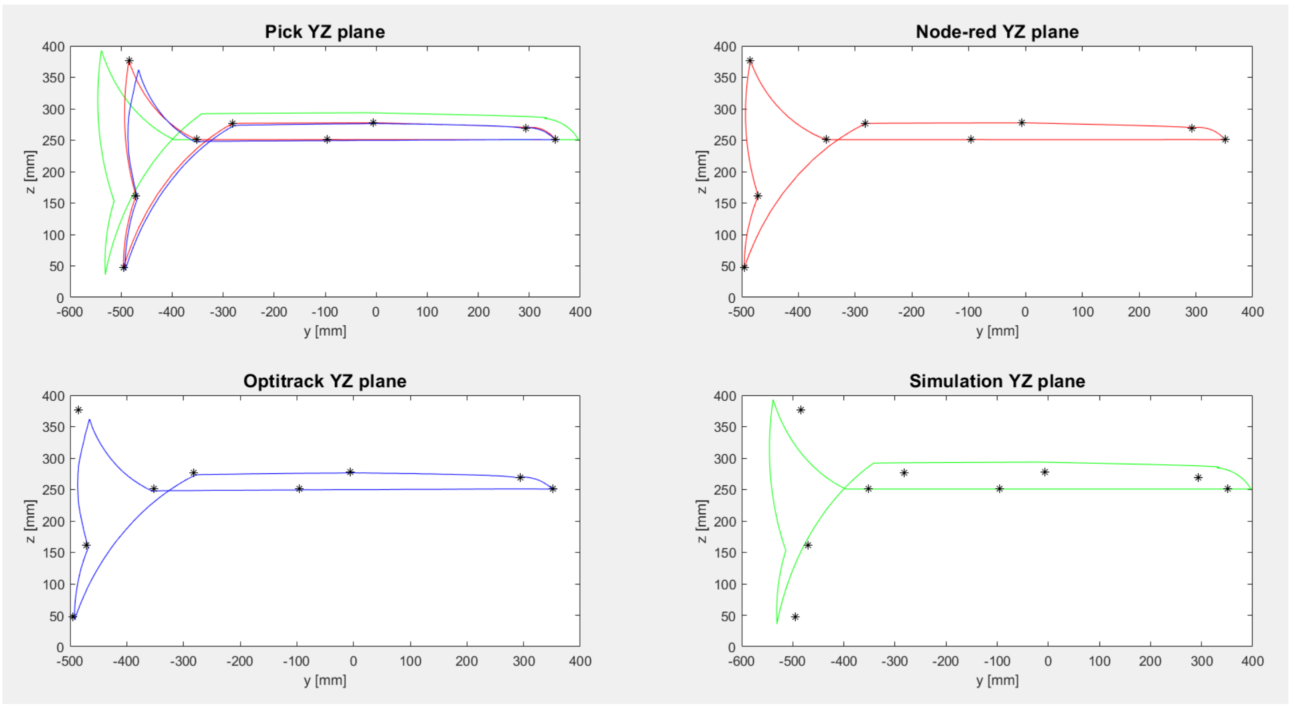


Figure 76: Focus on the YZ plane for the Pick task.

The tables below showcase the UR_TCP position errors in relation to the goal position, which represents the UR_TCP value at which the gripper is in the closed state.

Table 23: UR_TCP Pick task goal position.

TCP PICK TASK			
GOAL POSE	**X [mm]**	**Y [mm]**	**Z [mm]**
	-35.73	-495.44	46.95

Table 24: UR_TCP Pick task errors at goal.

TCP PICK TASK							
NODE-RED	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-35.70	-495.30	46.90	0.03	0.14	0.05	0.15
OPTITRACK	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-21.53	-491.81	42.88	14.2	3.63	4.07	15.21
SIMULATION	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-51.38	-530.98	36.03	15.65	35.54	10.92	40.34

As mentioned at the beginning of this section, an offset is applied to the z-coordinate of the data acquired from each system to align their initial points with the lowest vertex in *Figure 73*. However, significant errors are evident due to substantial discrepancies in the x and y coordinates at this point, both for Optitrack and the simulation results obtained with the DM. These errors contribute to the failure to achieve the goal. In the absence of data obtained with ROS, Node-RED represents the system that best matches the ideal behaviour, as demonstrated by the smallest error among all systems listed in *Table 24*.

On the other hand, the error at the goal position, relative to the simulation results, is the largest and highly significant, equal to approximately 4cm. One of the reasons for this outcome is the errors in the UR joints configuration for the DM.

- **Place task**

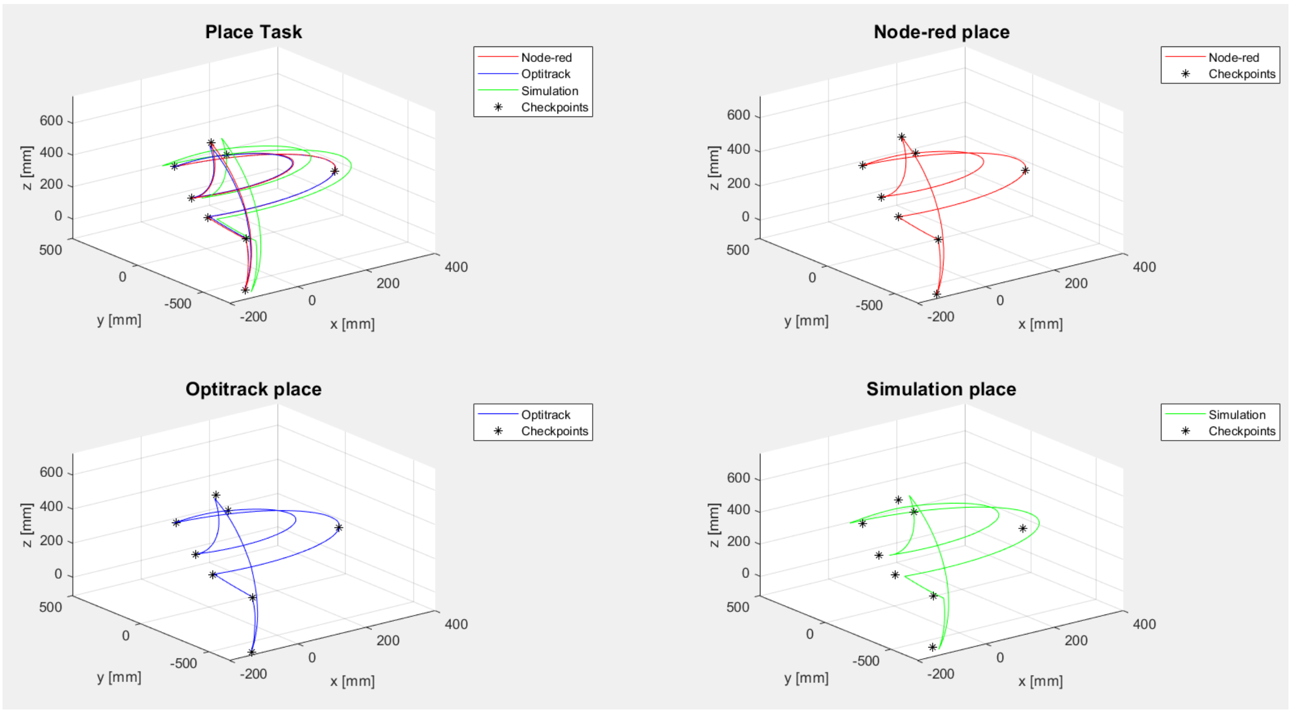


Figure 77: Comparison of the TCP graphs for the Place task.

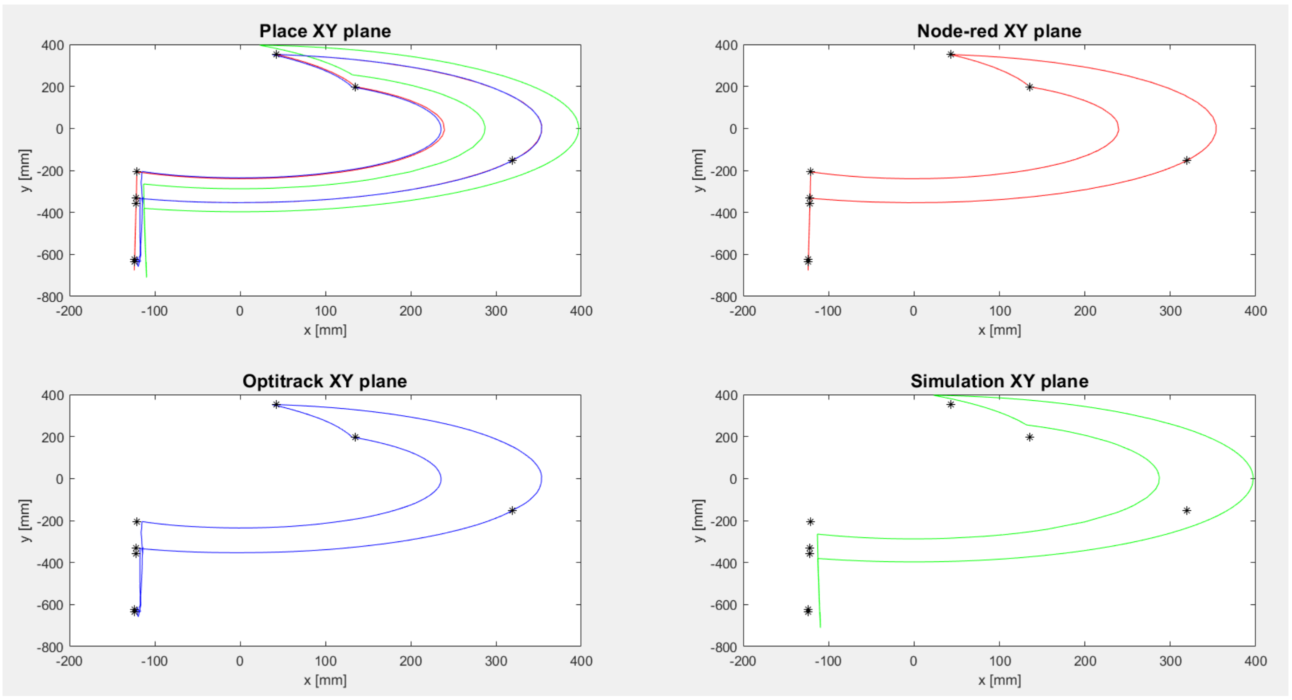


Figure 78: Focus on the XY plane for the Place task.

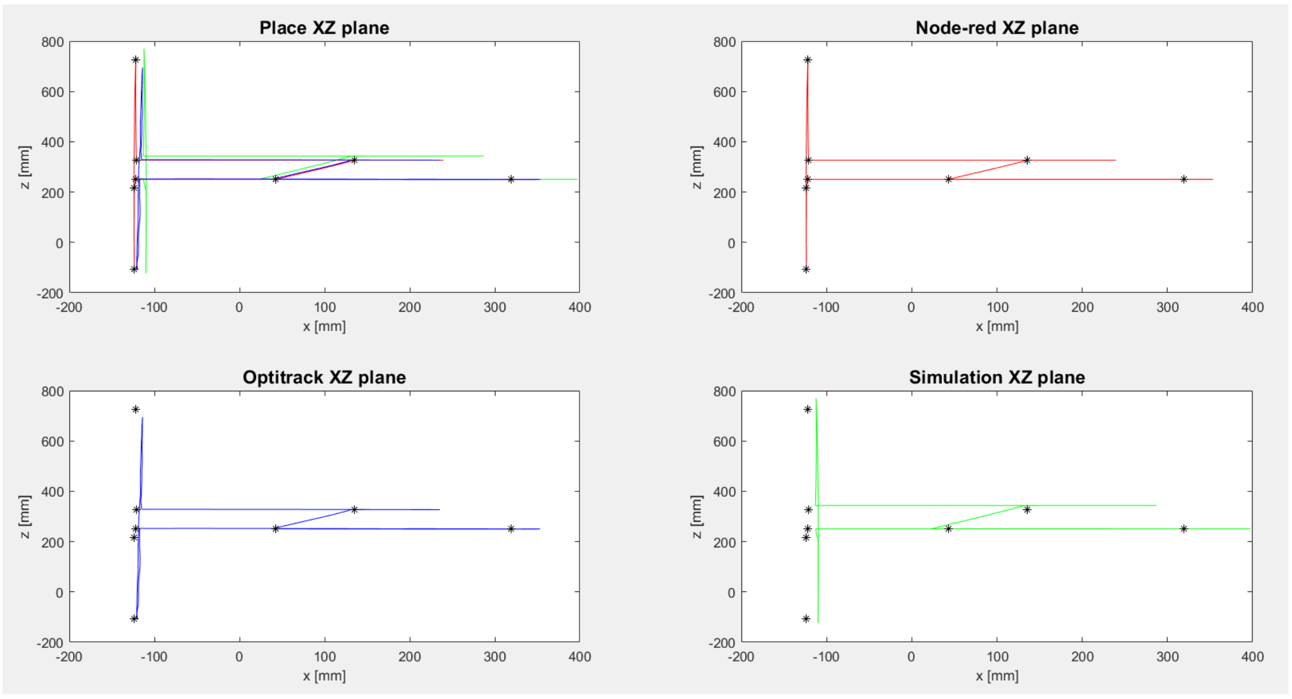


Figure 79: Focus on the XZ plane for the Place task.

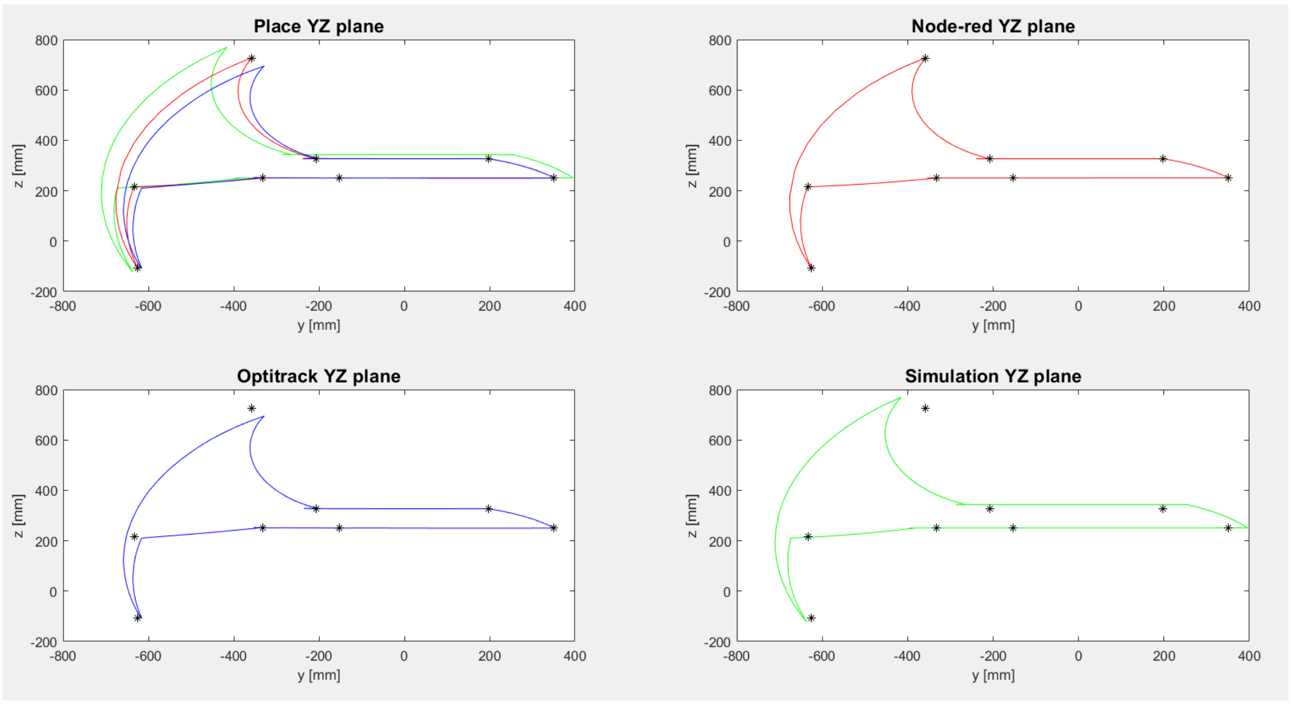


Figure 80: Focus on the YZ plane for the Place task.

Table 25: UR_TCP Place task goal position.

TCP PLACE TASK			
GOAL POSE	**X [mm]**	**Y [mm]**	**Z [mm]**
	-123.78	-624.76	-108.14

Table 26: UR_TCP Place task errors at goal.

TCP PLACE TASK							
NODE-RED	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-123.70	-624.50	-108.20	0.08	0.26	0.06	0.28
OPTITRACK	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-120.45	-616.85	-107.65	3.33	7.91	0.49	8.60
SIMULATION	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-109.98	-636.89	-121.86	13.80	12.13	13.72	22.93

The same considerations mentioned earlier for the pick task apply to the UR_TCP results related to the place task. However, a slight overall improvement is observed, particularly at the goal position, which corresponds to the designated position for releasing the object. At this point, the detected error is approximately 2.3cm.

5.4.2 Case study: Orientation angles specified

5.4.2.1 General settings and considerations

For the development of this version of the case study, the following parameters and error thresholds are set.

Table 27: Digital Model settings and threshold in Matlab and Simulink.

MiR100 PARAMETERS	
Pure Pursuit controller	
Desired Linear Velocity [m/s]	0.9
Maximum Angular Velocity [rad/s]	1
Lookahead distance [m]	0.4
Orientation phase threshold	
Maximum difference between angles $\Delta(\phi_{depart})$ [rad]	0.01
Orientation at goal station	
Maximum distance between angles at goal station Δ(ϕ) [rad]	0.0005 (= 0.029°)
AtStnRange thresholds	
Maximum distance between X coordinates Δ(X) [m]	0.08
Maximum distance between Y coordinates Δ(Y) [m]	0.3
AtGoal thresholds	
Maximum distance Δ(X,Y) [m]	0.08
UR3 PARAMETERS	
Trapezoidal Velocity Profile trajectory	
EndTime Pick task [s]	2.3
EndTime Place task [s]	1.9
UR3goalReached function	
UR3 configuration threshold [rad]	0.015

The initial considerations made for the first version of the analysed case study remain unchanged in this case as well. The crucial difference compared to the previous version lies in the orientation angles at which the mobile robot leaves the stations. In fact, for this version, there is no need to implement any algorithm that approximates the orientation of the MiR since the information regarding the orientation angles is directly provided by the program defined for this specific case study (see *Figure 26*). Since these angles are based on the relative MiR reference system (see *Figure 40*), they need to be transformed to obtain the angle values with respect to the reference x-axis, as depicted in *Figure 65*. The calculations are as follows:

$\theta_1 = \mathrm{deg2rad}(47°)$

$\theta_2 = \mathrm{deg2rad}(-144.7°) + \pi/2;$

$\theta_3 = \pi - \mathrm{abs}(\mathrm{deg2rad}(-139.1°) + \pi/2);$

$\theta = [0.8203; -0.954;, 2.2846]rad$

5.4.2.2 *MiR100 results*

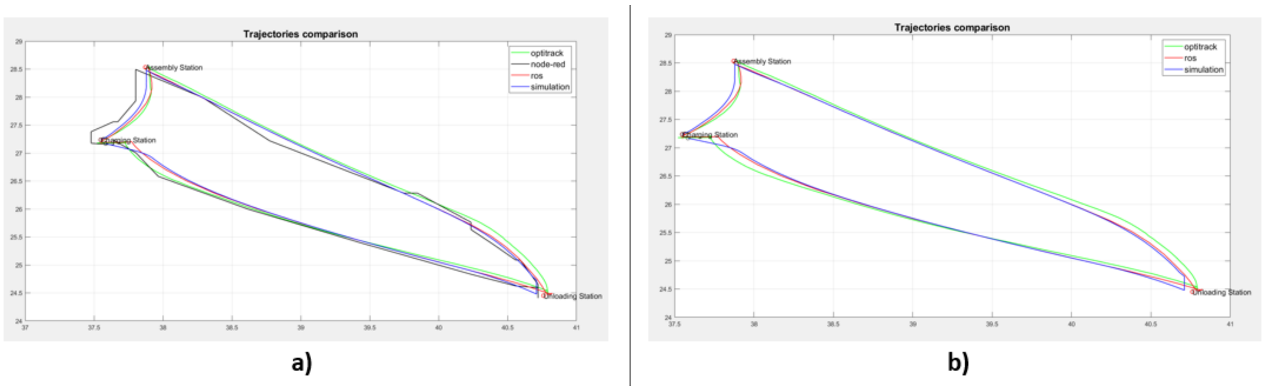


Figure 81: Trajectories of the different systems: a) with Node-RED, b) without Node-RED.

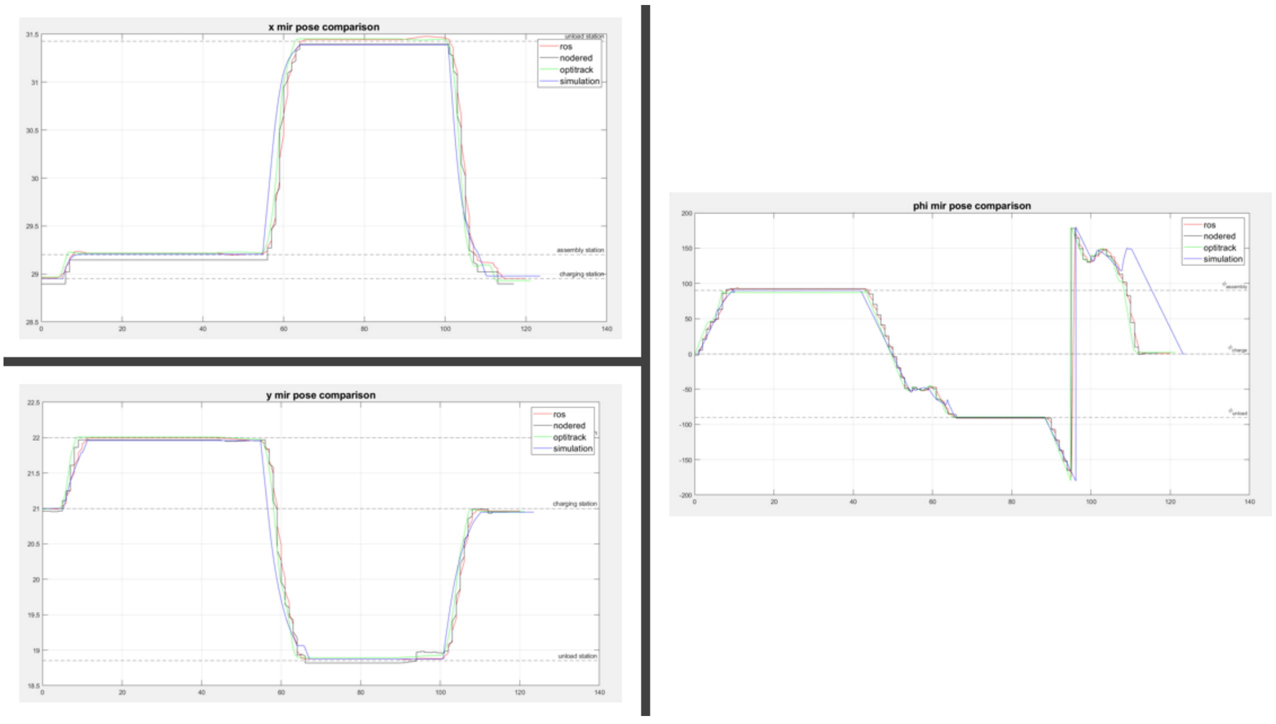


Figure 82: Position and orientation time plots for each system.

Table 28: Node-RED position errors.

NODE-RED							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.146	21.968	92.448	0.054	0.032	2.448	0.063
Unloading Stn	31.396	18.819	-90.995	0.031	0.033	0.995	0.045
Charging Stn	28.897	20.953	0.655	0.053	0.047	0.655	0.071

Table 29: ROS position errors.

ROS							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.218	21.997	92.236	0.018	0.003	2.236	0.018
Unloading Stn	31.444	18.874	-90.669	0.017	0.022	0.669	0.028
Charging Stn	28.953	20.964	0.655	0.003	0.036	0.655	0.036

Table 30: Optitrack position errors.

OPTITRACK							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.216	22.007	88.103	0.016	0.007	2.897	0.017
Unloading Stn	31.451	18.894	-89.482	0.024	0.042	0.518	0.048
Charging Stn	28.931	20.955	2.233	0.019	0.045	2.233	0.049

Table 31: Digital Model position errors.

SIMULATION (Digital Model)							
STATIONS	**X**	**Y**	**ϕ**	**Δ(X)**	**Δ(Y)**	**Δ(ϕ)**	**Δ(X,Y)**
Loading Stn	29.206	21.959	89.971	0.006	0.041	0.029	0.041
Unloading Stn	31.388	18.874	-89.971	0.039	0.022	0.029	0.045
Charging Stn	28.978	20.948	0.029	0.028	0.052	0.029	0.059

As can be observed from the graphs above and the results illustrated in the tables, generally better results are obtained compared to the previous case. *Figure 82* shows a very similar trend for all systems, without significant time shifts. The errors in position at the designated stations for the task are generally smaller compared to the previous version, especially for the Loading station. The results from the simulation performed with the DM are consistent with the accuracy value stated in the MiR datasheet. The only slightly higher value, equal to 0.059m, is found at the Charging station when the mobile robot returns to this position after completing the entire task. Therefore, this slight difference from the datasheet is not relevant for the correct execution of the pick and place task.

The graph related to orientation also shows excellent performance, with much fewer variations compared to the previous case. This improvement can be attributed to the fact that this version indicates the orientation angles with which the MiR leaves the stations.

5.4.2.3 *UR3 results*

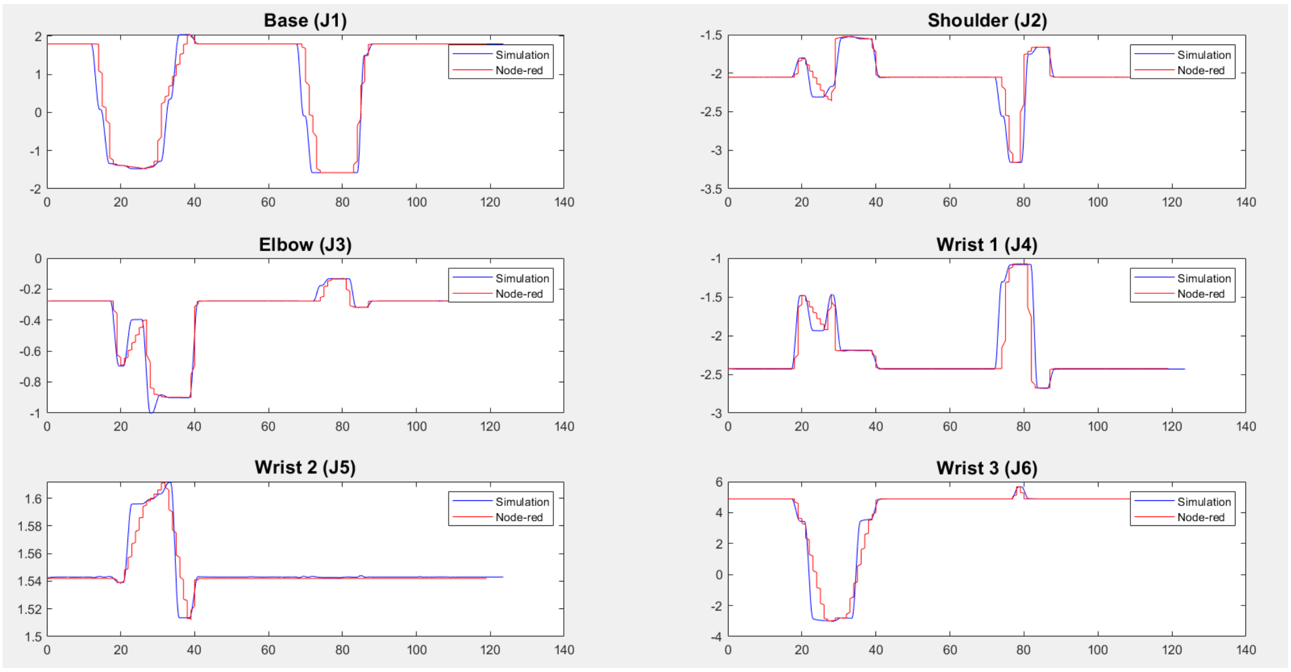


Figure 83: DM joints configurations compared with the Node-RED reference system.

Table 32: UR3 Pick task programmed waypoints from teach pendant.

UR3 PICK TASK SIMULATION ERRORS						
Wpts\Errors	**Δ(Base) [rad]**	**Δ(Shoulder) [rad]**	**Δ(Elbow) [rad]**	**Δ(Wrist 1) [rad]**	**Δ(Wrist 2) [rad]**	**Δ(Wrist 3) [rad]**
1	0.0004	0.0012	0.0006	0.0062	0.0008	0.0002
2	0.0039	0.0012	0.0005	0.0061	0.0012	0.0001
3	0.0034	0.0012	0.0005	0.0061	0.0012	0.0001
4	0.0002	0.0002	0.0014	0.0077	0.0010	0.0100
5	0.0002	0.0014	0.0006	0.0059	0.0001	0.0100
6	0.0002	0.0015	0.0006	0.0061	0.0000	0.0008
7	0.0007	0.0028	0.0021	0.0004	0.0001	0.0045
8	0.0036	0.0002	0.0028	0.0041	0.0003	0.0005
9	0.0003	0.0004	0.0029	0.0041	0.0014	0.0103
10	0.0006	0.0010	0.0004	0.0060	0.0007	0.0002

Table 33: UR3 Place task programmed waypoints from teach pendant.

UR3 PLACE TASK SIMULATION ERRORS						
Wpts\Errors	**Δ(Base) [rad]**	**Δ(Shoulder) [rad]**	**Δ(Elbow) [rad]**	**Δ(Wrist 1) [rad]**	**Δ(Wrist 2) [rad]**	**Δ(Wrist 3) [rad]**
1	0.0006	0.0010	0.0004	0.0060	0.0007	0.0002
2	0.0015	0.0011	0.0005	0.0060	0.0014	0.0001
3	0.0009	0.0011	0.0005	0.0060	0.0013	0.0001
4	0.0002	0.0016	0.0025	0.0118	0.0009	0.0000
5	0.0001	0.0020	0.0027	0.0110	0.0008	0.0002
6	0.0000	0.0023	0.0032	0.0085	0.0008	0.0086
7	0.0007	0.0021	0.0021	0.0059	0.0007	0.0102
8	0.0005	0.0002	0.0011	0.0029	0.0009	0.0011
9	0.0001	0.0004	0.0007	0.0060	0.0009	0.0005
10	0.0006	0.0010	0.0006	0.0061	0.0009	0.0002

The analysis of the UR3 results follows the same waypoints and approach as the previous case. In this version of the case study, the errors in the UR joints configuration are recorded in *Table 32* and *Table 33*. The majority of errors remain comfortably below the predefined threshold of 0.015 radians, with only a few instances where higher values are observed, primarily in joints 4 and 6.

5.4.2.4 UR3 TCP results

The results of UR_TCP are based on the same premise as in the previous case. The procedure to generate graphs for both the pick and place tasks in Matlab remains unchanged. The proposed results are related to the goal positions for both tasks, which are the same as those shown in *Table 23* and *Table 25*. The location errors align with the values found in the analysis of the first version of the case study. This lack of improvement can be attributed to the fact that, as mentioned earlier, the error values obtained from simulating the UR with the DM do not show significant improvements compared to the previous case. Consequently, there are no variations in the errors associated with UR_TCP. The only difference observed is with the Optitrack acquisition system, which exhibits worse behaviour compared to the results from version 1.

- **Pick task**

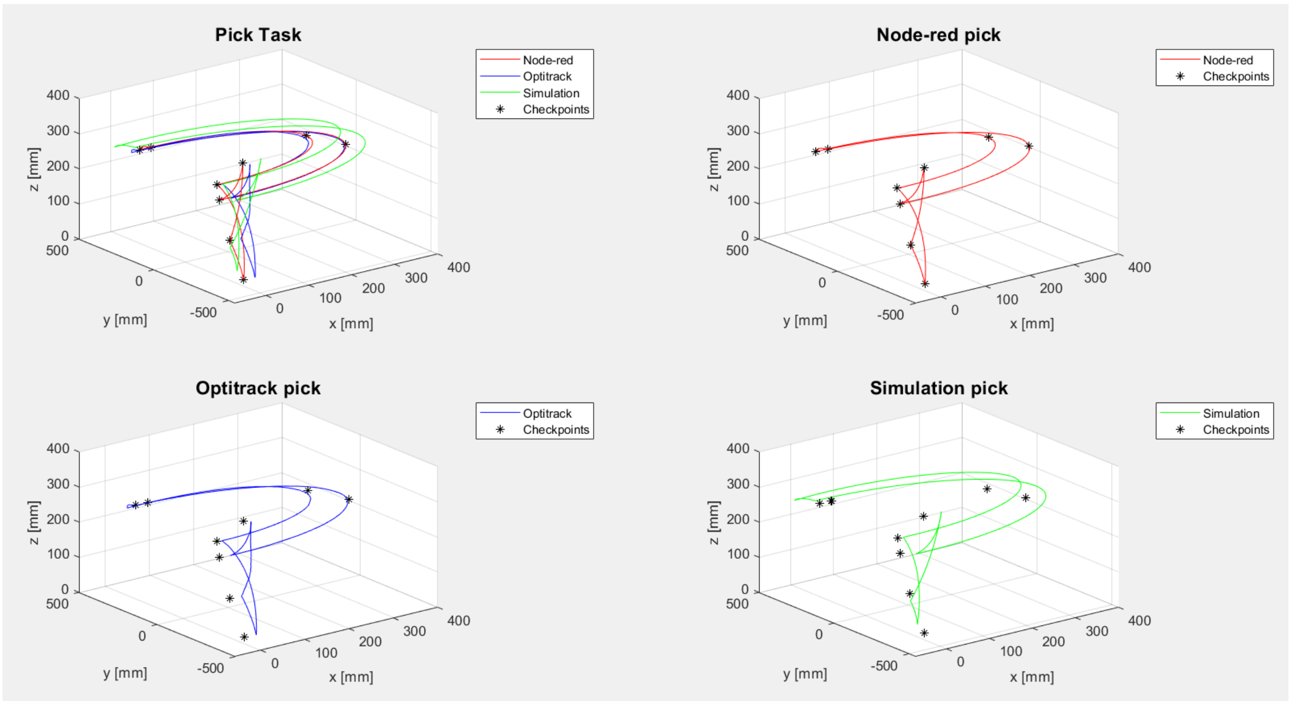


Figure 84: Comparison of the TCP graphs for the Pick task.

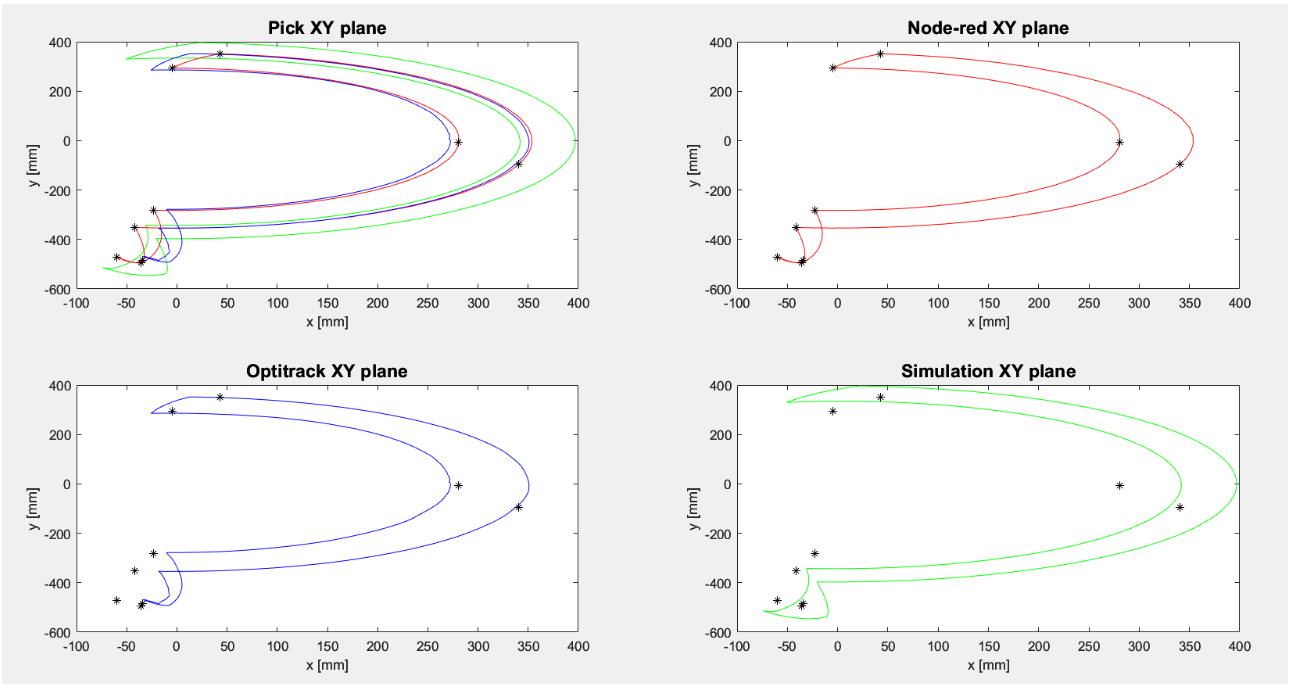


Figure 85: Focus on the XY plane for the Pick task.

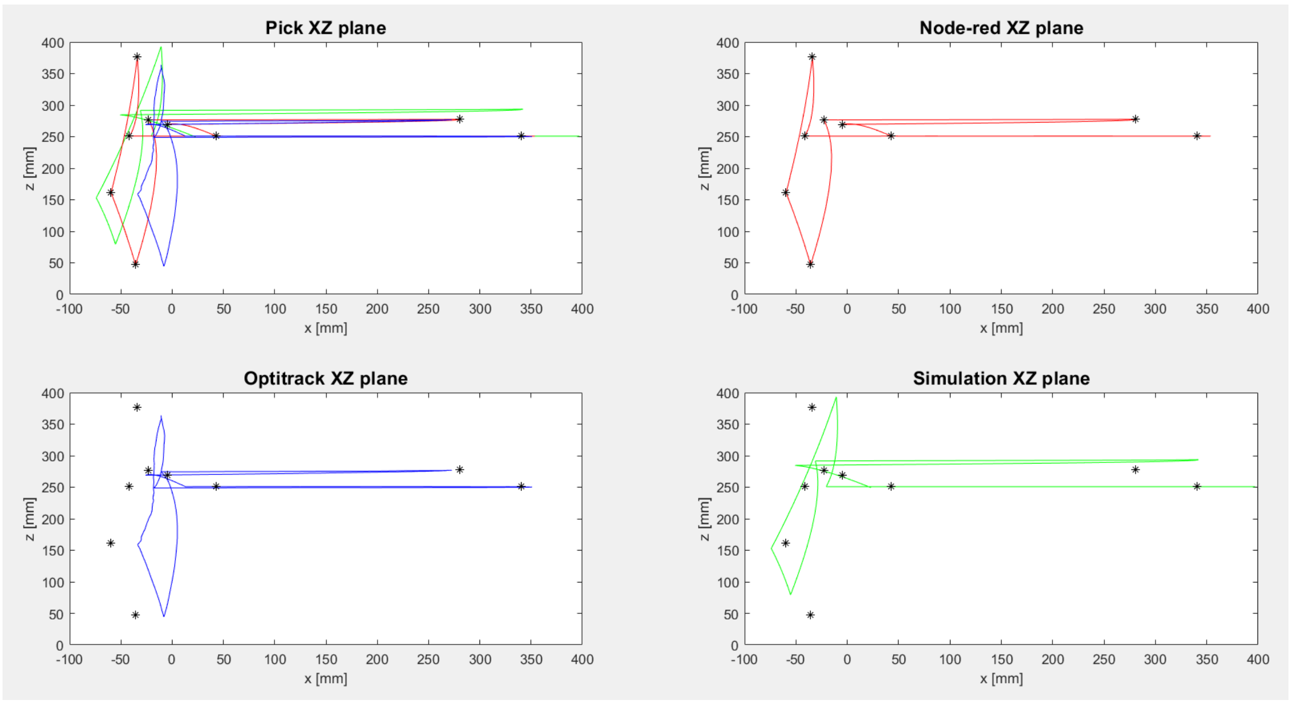


Figure 86: Focus on the XZ plane for the Pick task.

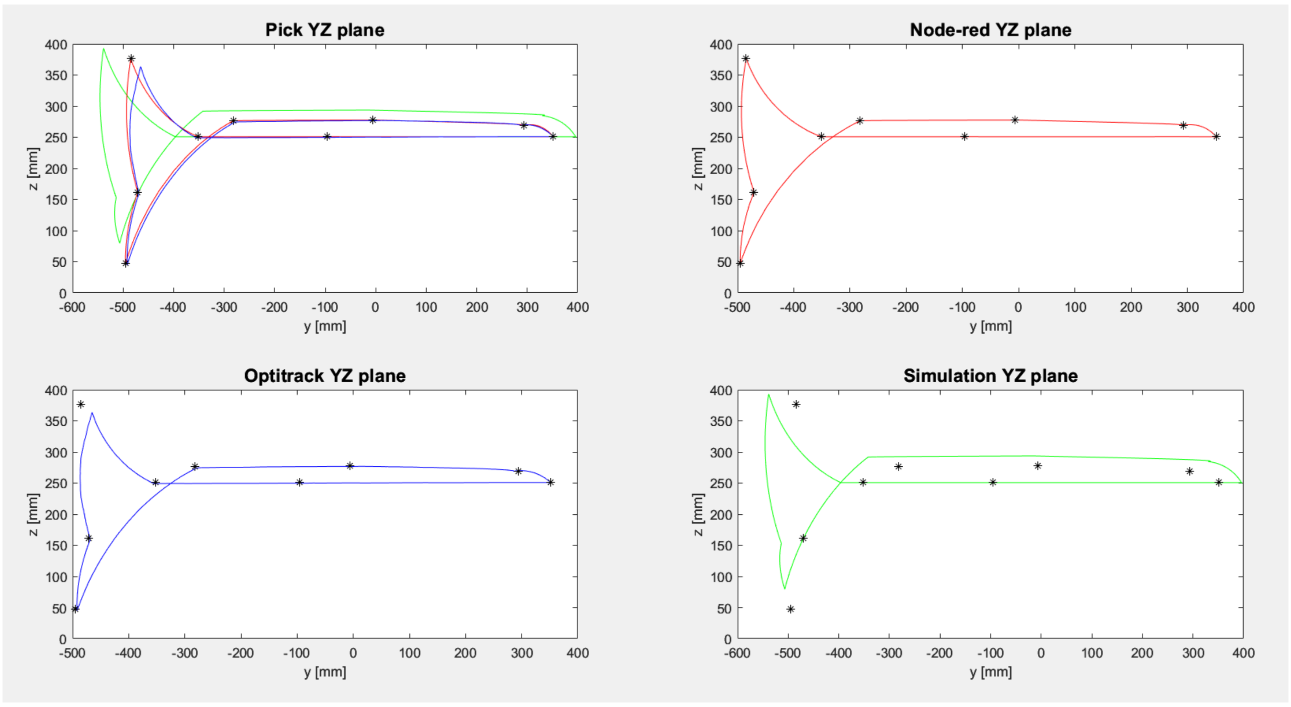


Figure 87: Focus on the YZ plane for the Pick task.

Table 34: UR_TCP Pick task errors at goal.

TCP PICK TASK							
NODE-RED	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-35.70	-495.30	46.90	0.03	0.14	0.05	0.15
OPTITRACK	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-8.13	-492.06	44.54	27.60	3.38	2.41	27.91
SIMULATION	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-51.39	-530.98	36.07	15.66	35.54	10.88	40.33

- **Place task**

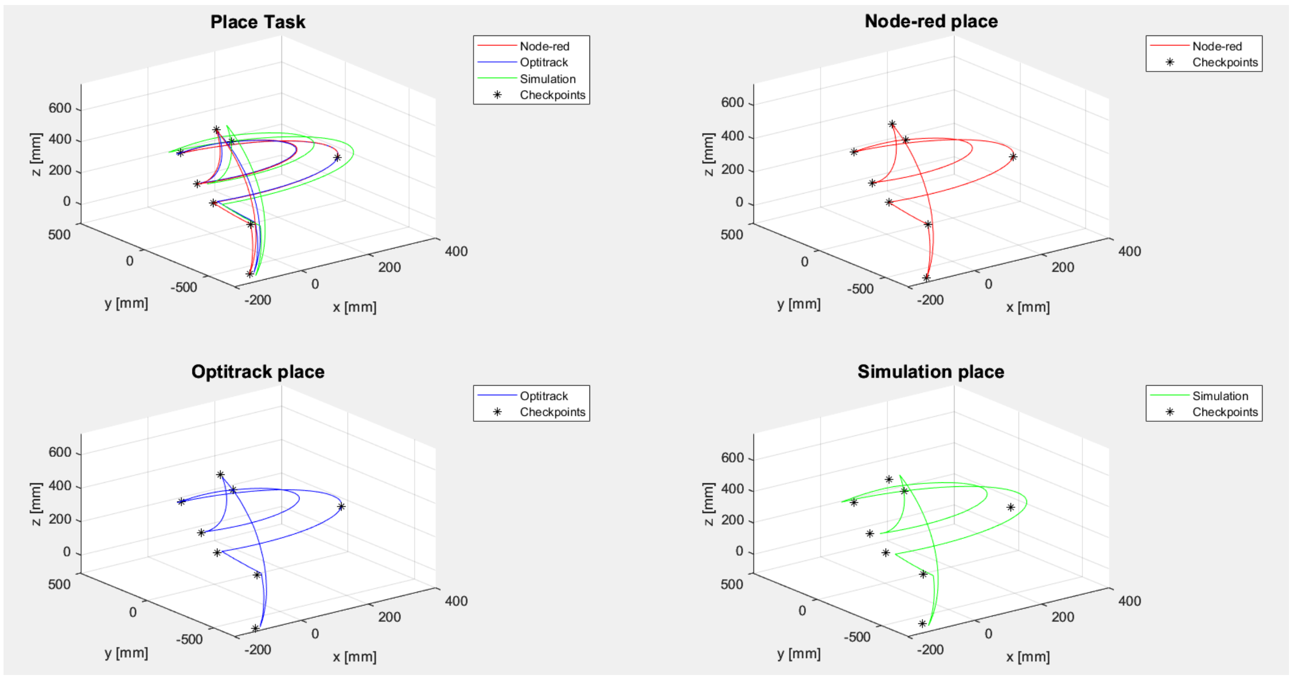


Figure 88: Comparison of the TCP graphs for the Place task.

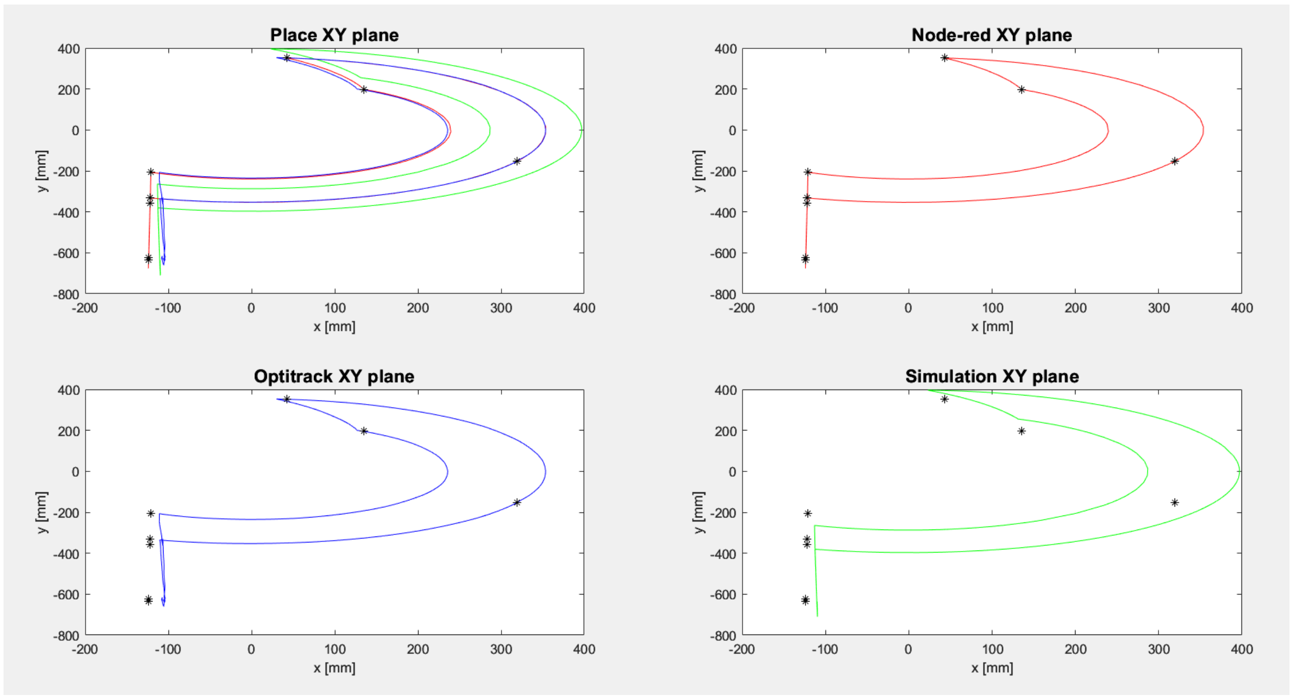


Figure 89: Focus on the XY plane for the Place task.

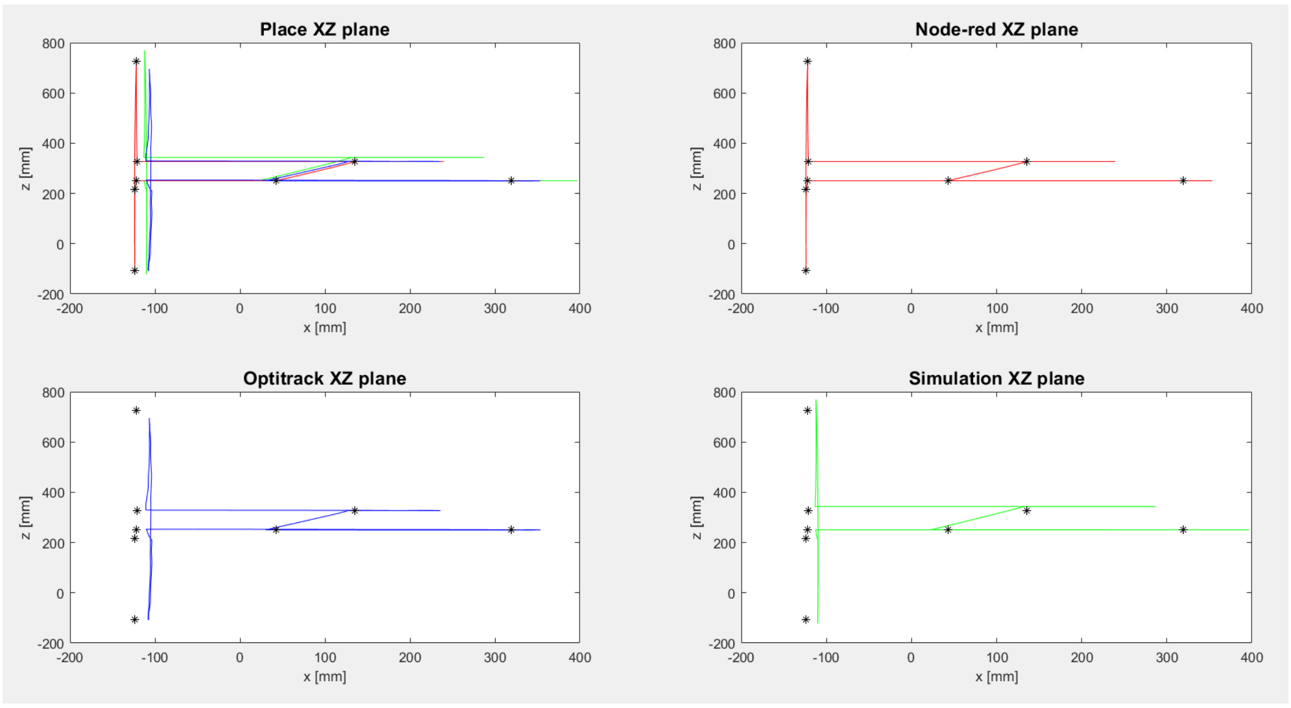


Figure 90: Focus on the XZ plane for the Place task.

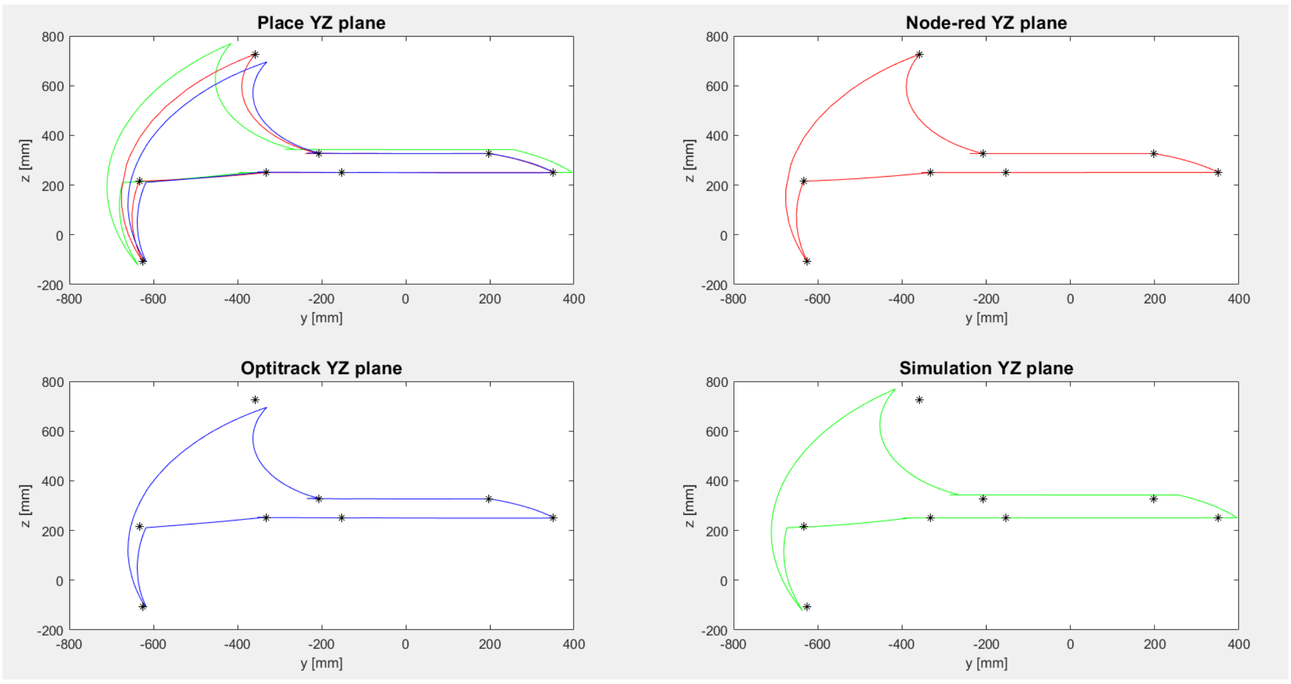


Figure 91: Focus on the YZ plane for the Place task.

Table 35: UR_TCP Place task errors at goal.

TCP PLACE TASK							
NODE-RED	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-123.70	-624.50	-108.20	0.08	0.26	0.06	0.28
OPTITRACK	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-107.48	-617.20	-107.87	16.3	7.56	0.27	17.97
SIMULATION	**X [mm]**	**Y [mm]**	**Z [mm]**	**Δ(X) [mm]**	**Δ(Y) [mm]**	**Δ(Z) [mm]**	**Δ(X,Y,Z) [mm]**
	-109.98	-636.54	-121.88	13.80	11.78	13.74	22.93

6 Digital Shadow

The digital shadow represents a significant advancement over the previous Digital Model, as outlined in section 1.1, by enabling real-time monitoring of a physical object in motion within the real-world. It receives data directly from the object and facilitates the replication of its movements in a virtual model.

6.1 Method

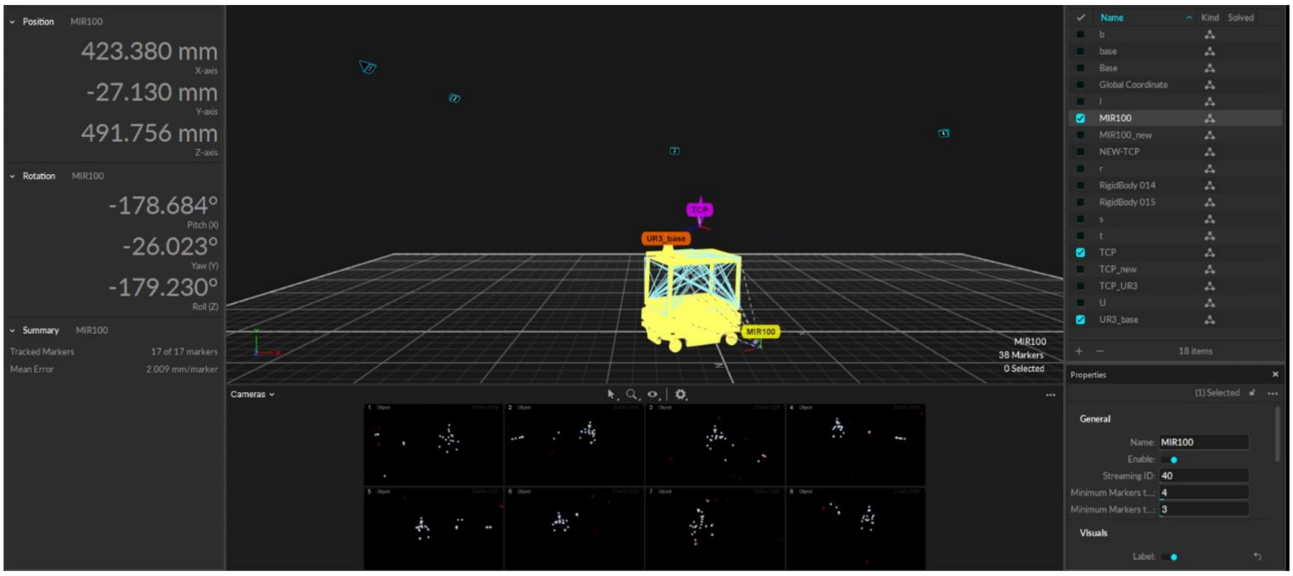
6.1.1 Utilized software and tools

The following tools and software are used to implement the DM:

- Matlab 2022b, Simulink
- Matlab Toolbox: Mobile Robotics Simulation Toolbox, ROS Toolbox
- MiR100 and UR3 software
- Ubuntu 20.04.6 LTS with ROS Noetic version
- Oracle VM VirtualBox 7.0.8
- Optitrack Motive
- TP-Link Switch Ethernet

The features and applications of the Node-RED, ROS, and Optitrack systems remain unchanged in accordance with the explanations presented in Chapter 2.

It is important to highlight that the Optitrack system itself operates as a Digital Shadow. In fact, it has the ability to track the position of a moving rigid body in real-time and render it in a 3D format using specialized markers and CAD files (see *Figure 92*). This comprehensive approach ensures a highly detailed representation of the 3D model and its surrounding environment.



Position MIR100
423.380 mm X-axis
-27.130 mm Y-axis
491.756 mm Z-axis

Rotation MIR100
-178.684° Pitch (X)
-26.023° Yaw (Y)
-179.230° Roll (Z)

Summary MIR100
Tracked Markers 17 of 17 markers
Mean Error 2.009 mm/marker

✓	Name	Kind	Solved
	b		
	base		
	Base		
	Global Coordinate		
	l		
✓	MIR100		
	MIR100_new		
	NEW-TCP		
	r		
	RigidBody 014		
	RigidBody 015		
	s		
	t		
✓	TCP		
	TCP_new		
	TCP_UR3		
	U		
✓	UR3_base		

18 items

Properties
(1) Selected
General
Name: MIR100
Enable: ●
Streaming ID: 40
Minimum Markers t...: 4
Minimum Markers t...: 3
Visuals
Label: ●

Figure 92: MoMa Digital Shadow in Optitrack with CAD model.

6.1.2 Main idea

To implement the Digital Shadow of the MoMa described in this thesis, the MiR100 and UR3 are treated as separate entities. They never simultaneously perform actions during the execution of programmed tasks for the MoMa. When the MiR is in motion, the UR remains inactive, and vice versa. The core concept behind this DS, which also serves as the foundation for the Digital Twin discussed in the subsequent chapter, revolves around employing a single computer that operates two virtual machines dedicated to the MiR and UR, respectively. The decision to employ two virtual machines is motivated by the fact that the drivers used for communication between ROS and the robots share packages with identical names. Running these packages concurrently causes a disruption in the connection. Since the drivers originate from two different projects available on GitHub[17][18], modifying the package content without a comprehensive understanding of the file and variable interdependencies proves to be challenging. Hence, for the initial development of the DS for both robots, a cautious and straightforward approach was chosen.

Specifically, the connection scheme implemented to integrate the MiR and UR robots in the same network is illustrated in *Figure 93*. An Ethernet switch is used to connect the UR3 and the host PC, on which the two virtual machines run, to the same MiR network. The ethernet connection represents a physical limitation because the host PC needs to be fixed on the MoMa and move with it. As explained in section 6.1.3, the Digital Shadow of the MiR can also be developed with a wireless connection, but it is not possible for the UR case due to an unstable wireless connection that disconnects the driver.

The procedure followed for both models is based on the tutorials provided in the respective GitHub repositories mentioned earlier.

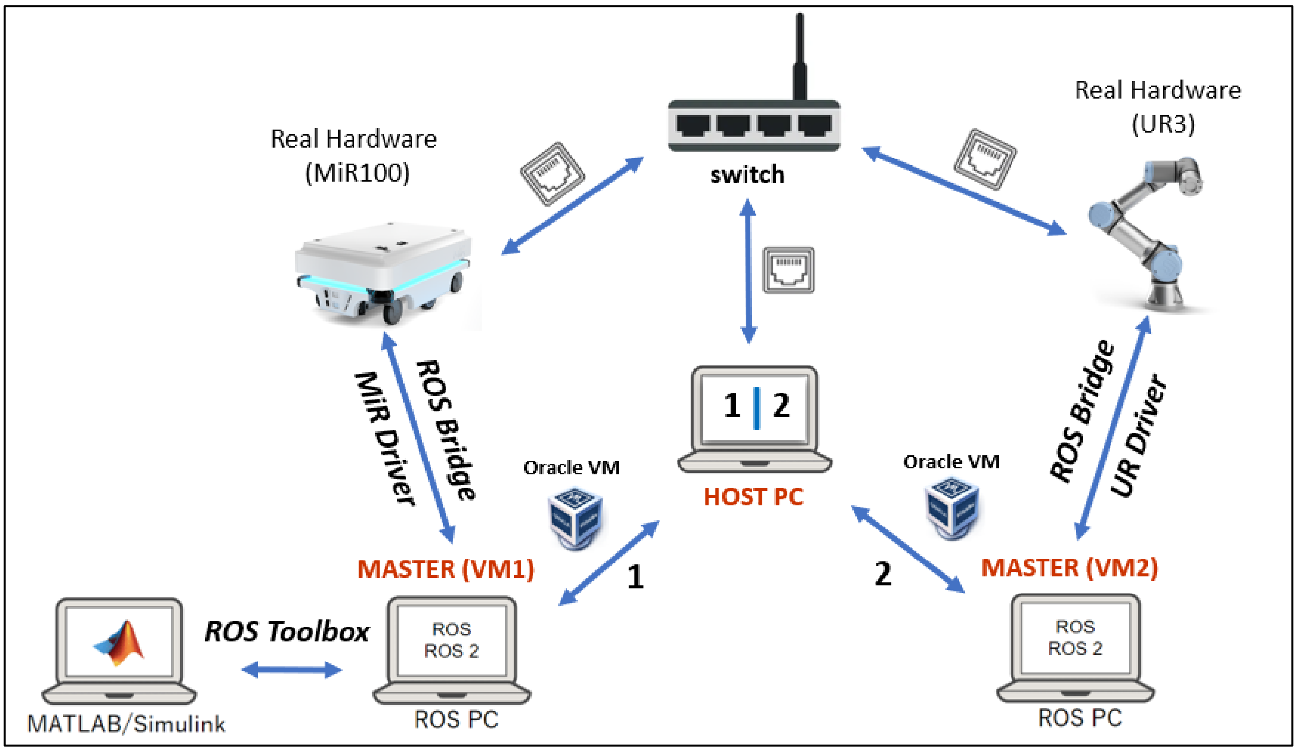


Figure 93: Network scheme with switch ethernet.

[17] MiR driver for ROS connection: https://github.com/DFKI-NI/mir_robot
[18] UR3 CB3 driver for ROS connection: https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

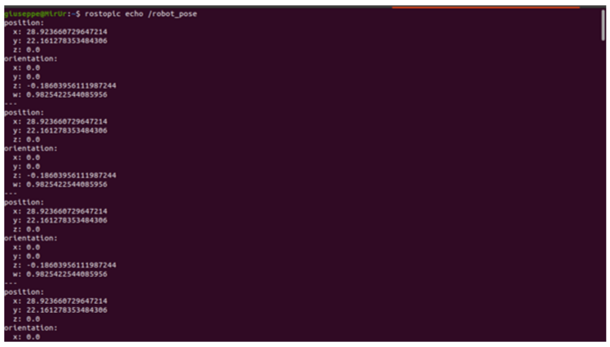
6.1.3 Development in Matlab, Simulink, and ROS systems

The development of the Digital Shadows in Matlab, Simulink, and ROS systems assumes that all the settings described in Chapter 2 have been correctly executed, including the configuration of the ROS Toolbox in Simulink and the synchronization procedure for communication between ROS and the MiR. In the case of communication between ROS and the UR3, it is exclusively achieved through Ethernet due to the unstable connection and frequent disconnections encountered with wireless communication. Once the entire system is properly configured, the next step is to install the RVIZ packages and plugins on the two virtual machines. As mentioned in section 2.3, RVIZ is a 3D simulation tool that allows real-time visualization of movements. It also facilitates transmitting commands from the digital device to the physical one directly through its ROS interface, as described for the DT in the following chapter. In this thesis, RVIZ is used for both the MiR and the UR.

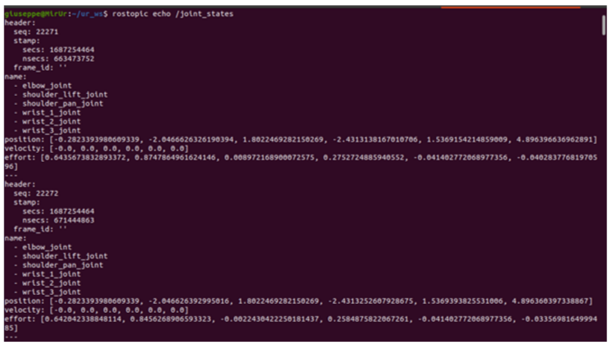
For the MiR, it utilizes messages containing information sent by the physical robot and acquired by the sensors and laser scanner to reproduce the same map created by the mobile robot using the same procedure. The map is updated in real-time based on environmental variations detected by the MiR. Once reproduced, this map is saved using the command 'rosrun map_server map_saver -f name_of_the_map'. The visualization of the laboratory map and the MiR on the RVIZ interface is displayed in 2D, which is sufficient for the purpose of this implementation.

For the UR, once the UR3 driver specifically designed for communication with ROS is running, it becomes possible to visualize a virtual 3D replica of the robot using RVIZ and the integrated Motion Planning plug-in. This is achieved by utilizing the UR3.URDF files included in the GitHub folders dedicated to this purpose. These files provide the necessary geometric and physical characteristics required for an accurate representation of the robot within the simulation environment. With the UR3 driver active, executing a program from the UR's teach pendant or performing simple movements enables real-time visualization in RVIZ, with minimal latency in the connection with the virtual machine.

For both robots, it is possible to observe and monitor real-time information during the execution of tasks in the real-world by utilizing appropriate commands typed on the Ubuntu command line in the form of 'rostopic echo <topic_name>'. The figures below depict the MiR pose and the UR3 joint state by launching the previous command on the Ubuntu terminal.



```
giuseppe@MirUr:~$ rostopic echo /robot_pose
position:
  x: 28.923660729647214
  y: 22.161278353484306
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: -0.18603956111987244
  w: 0.9825422544085956
---
position:
  x: 28.923660729647214
  y: 22.161278353484306
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: -0.18603956111987244
  w: 0.9825422544085956
---
position:
  x: 28.923660729647214
  y: 22.161278353484306
  z: 0.0
orientation:
  x: 0.0
  y: 0.0
  z: -0.18603956111987244
  w: 0.9825422544085956
---
position:
  x: 28.923660729647214
  y: 22.161278353484306
  z: 0.0
orientation:
  x: 0.0
```



```
giuseppe@MirUr:~/ur_ws$ rostopic echo /joint_states
header:
  seq: 22271
  stamp:
    secs: 1687254464
    nsecs: 663473752
  frame_id: ''
name:
  - elbow_joint
  - shoulder_lift_joint
  - shoulder_pan_joint
  - wrist_1_joint
  - wrist_2_joint
  - wrist_3_joint
position: [-0.2823393980609339, -2.0466626326190394, 1.8022469282150269, -2.4313138167010706, 1.5369154214859009, 4.896396636962891]
velocity: [-0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
effort: [0.6435673832893372, 0.8747864961624146, 0.008972168900072575, 0.2752724885940552, -0.041402772068977356, -0.04028377681970596]
---
header:
  seq: 22272
  stamp:
    secs: 1687254464
    nsecs: 671444863
  frame_id: ''
name:
  - elbow_joint
  - shoulder_lift_joint
  - shoulder_pan_joint
  - wrist_1_joint
  - wrist_2_joint
  - wrist_3_joint
position: [-0.2823393980609339, -2.046626392995016, 1.8022469282150269, -2.4313252607928675, 1.5369393825531006, 4.896360397338867]
velocity: [-0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
effort: [0.642042338848114, 0.8456268906593323, -0.0022430422250181437, 0.2584875822067261, -0.041402772068977356, -0.0335698164999485]
---
```

Figure 94: Left side: robot pose in real-time from ROS terminal; right side: UR joint states in real-time from ROS terminal.

In Matlab and Simulink, the DS is only implemented for the MiR, as managing the message types related to the UR topics proves to be difficult in Matlab and Simulink. For this reason, real-time information from the UR3 is monitored using Node-RED and the Ubuntu terminal, as shown in *Figure 94*. The procedure and blocks used for the MiR DS implementation in Simulink are the same as illustrated in section 2.3.1 and *Figure 21* and *Figure 22*, allowing for real-time data monitoring.

The only addition is the 'Robot Visualizer' block, available through the Mobile Robotics Simulation Toolbox. This block takes a map as input, in this case, the same map generated with the 'BinaryOccupancyMap' function for the DM, along with the real-time position of the mobile robot. It represents the current trajectory executed by the physical device in a stylized manner, providing a useful visual representation of the robot's movement (see *Figure 96*).

6.2 Results

6.2.1 MIR100 Digital Shadow

Figure 95 showcases the RVIZ environment, which demonstrates the real-time replication of the MiR's map. In this figure, the movement of the MiR near the Charging station is observable. The green line represents a segment of the trajectory followed by the physical robot, while the red points correspond to the laser scanner data from the MiR, providing information about obstacle detection. It is noteworthy that the virtual map displayed on the right side of the figure below perfectly replicates the map of the MiR, shown on the left side.

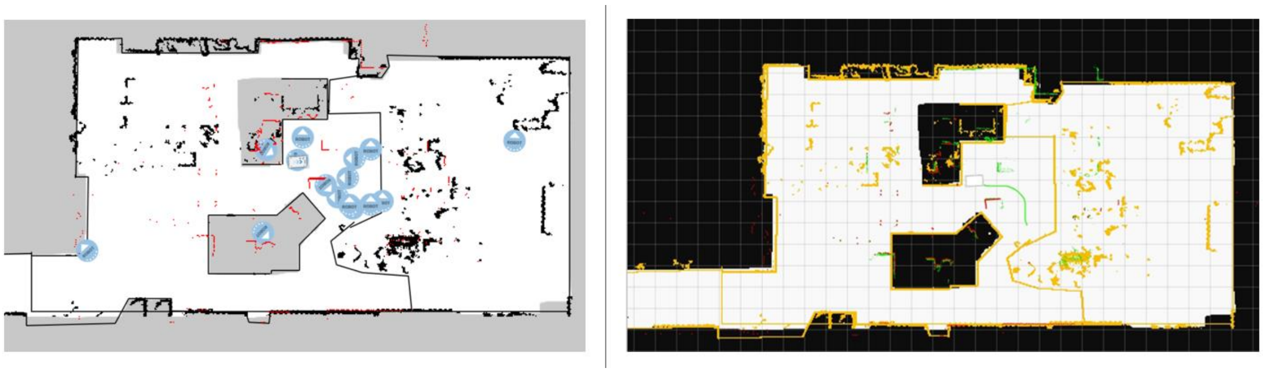


Figure 95: RVIZ 2D MiR Simulation comparison with the MiR system map.

Regarding the DS developed in Matlab and Simulink, the results are depicted in the figure below. It showcases the Robot Visualizer block in Simulink, along with the real-time trajectory executed by the MiR.

The trajectory is also visualized through the XY graphs, which are inserted in the grey panel as illustrated in *Figure 22*.

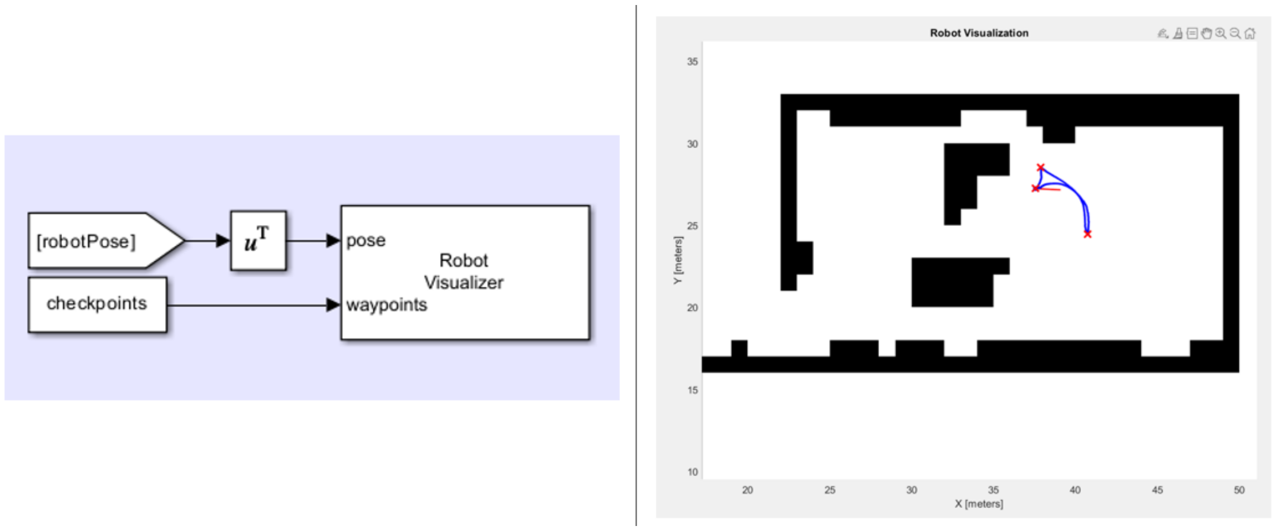


Figure 96: MiR Digital Shadow in Matlab and Simulink.

6.2.2 UR3 Digital Shadow

The Digital Shadow of the UR3 is depicted in *Figure 97*, showcasing the 3D model within the RVIZ interface running in the ROS virtual machine. When a program or movement is launched through the teach pendant, this virtual model replicates the motions, taking into account the minimal delay caused by the connection between ROS and the physical device.

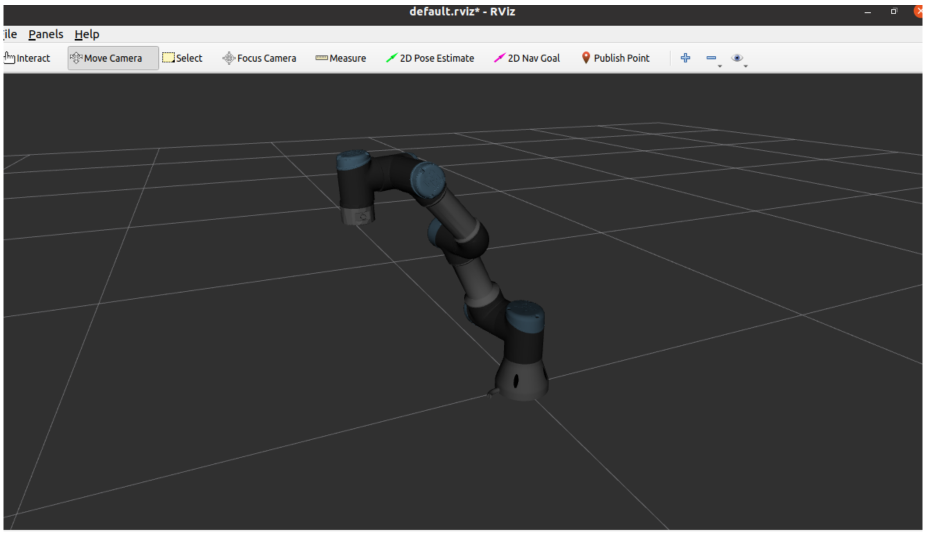


Figure 97: UR3 Digital Shadow in the RVIZ environment.

6.2.3 MoMa Digital Shadow

As described in [section 6.1.2](#), the Digital Shadow of the MoMa consists of the two previously developed models placed side by side and connected to the same network. With the drivers correctly running in their respective virtual machines, it becomes possible to perform the pick and place task, which was already involved in the Digital Model and preliminary tests. Additionally, real-time monitoring of the actual behaviour of the two robots in the RVIZ environment and obtaining their position and configuration information from the Ubuntu command line are feasible, as shown in *Figure 94*.

The figure below depicts an example situation of the pick and place task, specifically illustrating the moment when the MiR reaches the Unloading station and the UR3 starts executing the place task.

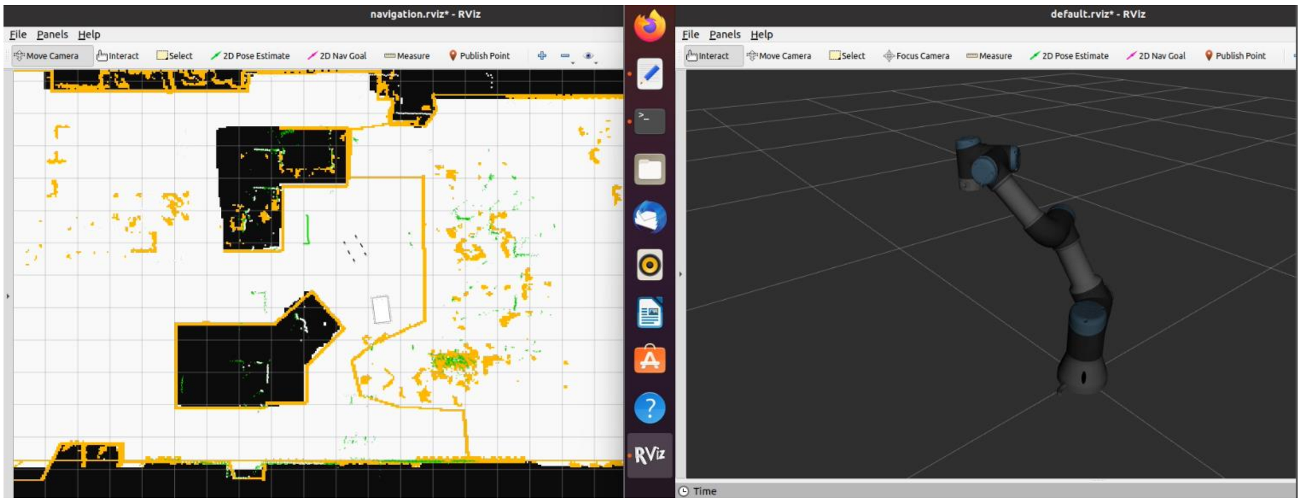


Figure 98: MoMa Digital Shadow in RVIZ, placing MiR DS and UR DS side by side.

7 DIGITAL TWIN

The ultimate objective of this thesis is to establish a comprehensive digital environment that ensures complete control over the physical robots within the laboratory space. This objective is accomplished by developing two Digital Twins, one for the MiR100 and the other for the UR3, respectively.

7.1 METHOD

7.1.1 Utilized software and tools

The following tools and software are used to implement the DM:

- Matlab 2022b, Simulink
- Matlab Toolbox: Mobile Robotics Simulation Toolbox, ROS Toolbox
- MiR100 and UR3 software
- Ubuntu 20.04.6 LTS with ROS Noetic version
- Oracle VM VirtualBox 7.0.8
- Optitrack Motive
- TP-Link Switch Ethernet

To initiate the connection between ROS and the physical robots, the same scheme illustrated in *Figure 93* is utilized for this purpose. However, since the DT aims to enable command transmission from the digital device to the real-world, the constraint of connecting the UR3 with an Ethernet cable, and consequently keeping the ROS host PC in close proximity to the robot, prevents wireless control of the entire MoMa. One possible solution could be to remotely control the host PC fixed on the MoMa, similar to what is done for the Digital Shadow, by utilizing appropriate software or tools. This would allow managing the ROS environment of the host PC wirelessly from another PC placed anywhere within the workspace environment. Although this solution is not developed and tested in this thesis project, it could serve as a starting point for a possible future project.

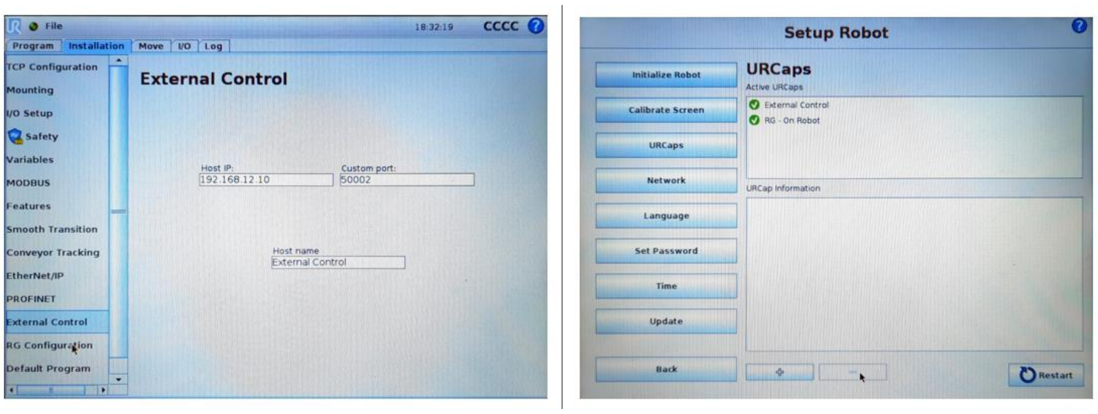
Furthermore, the ROS host PC could be replaced with a Raspberry Pi to avoid the need for a PC on the MoMa, which can be more challenging to manage and bulkier in size.

7.1.2 Development of the Digital Twin

The development of the MoMa's DT follows the same procedure and considerations as the DS discussed in the previous section. The only difference lies in the UR. Specifically, the UR software requires the installation of a plug-in that enables control of the physical device using an external device other than the teach pendant. The External Control URCaps plug-in for communication with ROS can be downloaded from the GitHub folder, which already contains the driver used for the DS. The installation process involves inserting a USB pen drive into the teach pendant. The following figures illustrate the settings required for this purpose.

The host IP refers to the IP address of the ROS virtual machine (VM2 in *Figure 93*), which should be configured as the IPv4 address in the network settings. It is crucial for this address to align with the network ID of the UR3 (192.168.12) to ensure that both devices are connected to the same network.

In order to facilitate control of the UR3 robot from the digital world to the real-world, it is essential to successfully launch the UR drivers. Once this step is completed, the program with the URCaps external control installed has to be executed from the teach pendant. At this stage, within the RVIZ environment, the Motion Planning plug-in can be leveraged to command the physical robot.



Host IP: 192.168.12.10
Custom port: 50002
Host name: External Control

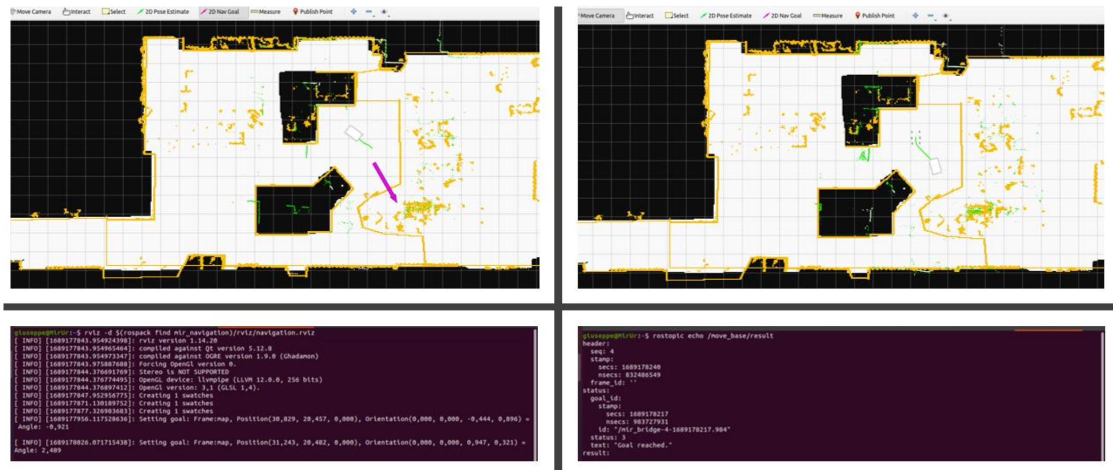
Figure 99: UR3 External Control configuration on the teach pendant.

7.2 RESULTS

7.2.1 MiR100 Digital Twin

Regarding the MiR DT, the required steps are exactly the same as described for the DS and in Chapter 2. As mentioned earlier, the RVIZ tool allows for direct control of the mobile. This is achieved by utilizing the '2D Nav Goal' command. The motion of the MiR in the real-world is executed by selecting a desired position on the RVIZ MiR map (see *Figure 100*). If the chosen location is not reachable, the robot does not complete the programmed task. Additionally, if an obstacle suddenly appears along the MiR's trajectory during its movement, the robot is capable of generating a new trajectory to reach the goal pose if it is still feasible. Concurrently, the actual movement of the mobile robot is replicated in real-time within the RVIZ simulation environment.

Figure 100 illustrates the RVIZ procedure for commanding the real mobile robot to reach a new goal position, which is programmed using the 2D Nav Goal tool and indicated on the map by a pink arrow. The published goal position is also visible in the Ubuntu terminal where the RVIZ command is executed, as depicted in the third figure below. Moreover, the figure on the right side demonstrates the successful achievement of the goal by the MiR robot. This is further confirmed by the fourth figure, displaying a feedback message indicating the task status, which in this case reports a 'Goal reached' message.



```
giuseppe@MirUr:~$ rviz -d $(rospack find mir_navigation)/rviz/navigation.rviz
[ INFO] [1689177843.954924398]: rviz version 1.14.20
[ INFO] [1689177843.954965464]: compiled against Qt version 5.12.8
[ INFO] [1689177843.954973347]: compiled against OGRE version 1.9.0 (Ghadamon)
[ INFO] [1689177843.975887688]: Forcing OpenGl version 0.
[ INFO] [1689177844.376691769]: Stereo is NOT SUPPORTED
[ INFO] [1689177844.376774495]: OpenGL device: llvmpipe (LLVM 12.0.0, 256 bits)
[ INFO] [1689177844.376897412]: OpenGl version: 3,1 (GLSL 1,4).
[ INFO] [1689177847.952956775]: Creating 1 swatches
[ INFO] [1689177871.130189752]: Creating 1 swatches
[ INFO] [1689177877.326983683]: Creating 1 swatches
[ INFO] [1689177956.117528636]: Setting goal: Frame:map, Position(30,829, 20,457, 0,000), Orientation(0,000, 0,000, -0,444, 0,896) =
 Angle: -0,921

[ INFO] [1689178026.071715438]: Setting goal: Frame:map, Position(31,243, 20,482, 0,000), Orientation(0,000, 0,000, 0,947, 0,321) =
Angle: 2,489
```

```
giuseppe@MirUr:~$ rostopic echo /move_base/result
header:
  seq: 4
  stamp:
    secs: 1689178240
    nsecs: 832486549
  frame_id: ''
status:
  goal_id:
    stamp:
      secs: 1689178217
      nsecs: 983727931
    id: "/mir_bridge-4-1689178217.984"
  status: 3
  text: "Goal reached."
result:
```

Figure 100: MiR DT using 2D Nav Goal and Ubuntu terminal.

Additionally, it is also possible to provide new goals or specify velocities to the real robot through the Ubuntu command line. This can be achieved by using commands such as ‘rostopic pub /cmd_vel geometry_msgs/Twist -- '[0.0,0.0,0.0]' '[0.0,0.0,0.5]'’, which assigns a constant angular velocity of 0.5 rad/s, for example, to the MiR robot.

7.2.2 UR3 Digital Twin

The Motion Planning plug-in simplifies robot motion programming by manipulating a virtual shadow, depicted in orange in *Figure 103*, to achieve the desired configuration. If the visualized movement aligns with the safety conditions and the intended task, it can be executed. This feature proves to be a valuable asset as it allows for controlling the actions of the real UR3 while avoiding potential security issues. By experimenting with the shadow's movements, already incorporated within the RVIZ motion planning feature, users can conduct preliminary trials before executing actions with the physical robot. Additionally, by selecting the ‘Use Collision-Aware IK’ option, the solver is able to avoid the collision configurations.

The Motion Planning interface offers two different ways to program the joint configuration:

- Directly moving the end effector of the orange UR shadow to the desired configuration, as shown in *Figure 102*. The end effector motion can be performed by utilizing a blue sphere or by dragging the dedicated arrows.
- Setting specific configuration values for each UR joint from the dedicated tab depicted in *Figure 101*.

Finally, *Figure 103* illustrates the motion sequence performed by the UR DT. It begins with the new configuration planning, represented by the movement of the orange shadow. This is followed by the execution of the planned motion, where the virtual robot aligns with the configuration of the orange shadow. This alignment is then reflected in the movement of the real robot.

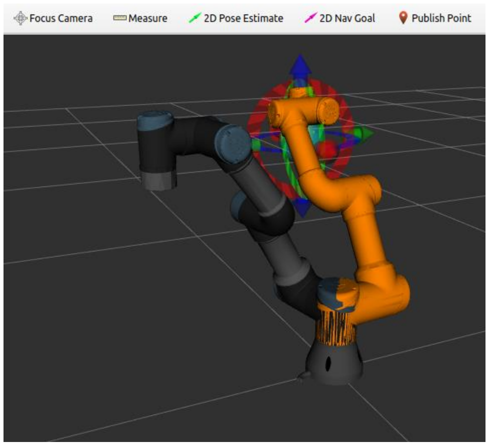
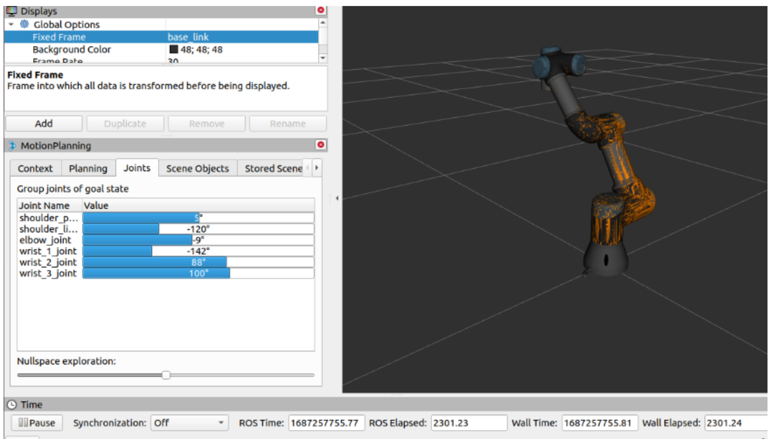


Figure 102: Moving UR by using End Effector tool.


Displays
Global Options
Fixed Frame: base_link
Background Color: ■ 48; 48; 48
Frame Rate: 30

Fixed Frame
Frame into which all data is transformed before being displayed.

Group joints of goal state

Joint Name	Value
shoulder_p...	5°
shoulder_li...	-120°
elbow_joint	-9°
wrist_1_joint	-142°
wrist_2_joint	88°
wrist_3_joint	100°

Nullspace exploration:

Time
Pause Synchronization: Off ROS Time: 1687257755.77 ROS Elapsed: 2301.23 Wall Time: 1687257755.81 Wall Elapsed: 2301.24

Figure 101: Joints configurations in RVIZ to planning UR motion.

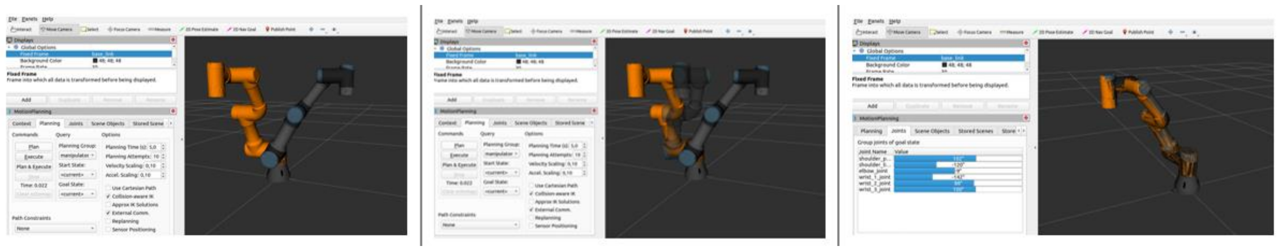


Figure 103: Sequence of UR DT movements in the RVIZ Motion Planning environment.

8 CONCLUSION

By reviewing the arguments developed in this thesis and the results obtained, the following conclusions can be drawn.

Among the acquisition systems ROS and Node-RED, the preliminary tests and subsequent results related to the Digital Model indicate that ROS offers superior precision and accuracy when comparing the information obtained with that derived directly from the mobile robot. Furthermore, ROS provides a range of tools, plugins, and resources for simulation, along with its ability to interface with various software installed on external devices such as Matlab and Simulink, enabling the development of applications for robots. It is important to note that the analysis of the digital models developed in this thesis represents only a fraction of the potential that this system offers.
Optitrack Motive on the other hand, functions as a real-time tracking system for moving rigid bodies, ensuring high accuracy compared to the real reference system. However, it is important to emphasize that in a hypothetical work environment, such as the manufacturing sector of an industry, it is impractical to monitor a mobile manipulator that covers the entire environment due to the unrealistic installation of cameras to cover such large workspaces. Consequently, its usage is limited to specific purposes and analyses involving a circumscribed area, as demonstrated in this thesis.
Based on the conducted tests, a distance of approximately 0.055m is observed between the values acquired with ROS and those captured with Optitrack. Considering the variability of the MiR system and the calibration results obtained for the Optitrack system, it can be concluded that this positional error of the mobile robot relative to the real values aligns with expectations and can be considered acceptable for the purposes presented in this thesis.

Focusing on the results obtained from the development and simulations of different digital models, they undoubtedly constitute a highly useful tool for optimizing processes in the context of Industry 4.0. Specifically:

- The Digital Model serves as an important tool for simulating moderately complex scenarios, such as the pick-and-place between two studied workstations in this thesis. This allows the prevention of physical device usage and resource allocation by virtually estimating and analysing performance and potential risks associated with executing a certain process in the real work environment. Furthermore, as demonstrated in this case study, it is possible to acquire data from a previously executed physical process to conduct in-depth evaluations and analyses of the obtained performance, evaluating potential optimization solutions.
  The Digital Model implemented in this thesis proves to be sufficiently accurate and reliable for the mentioned applications, with position errors slightly higher than the accuracy indicated in the mobile robot's datasheet when compared to the programmed reference positions. This error can be significantly reduced by approximately 1cm by providing more information during the process programming phase, such as the orientation angles of the MiR, thereby limiting the degrees of freedom for the robot.
  Lastly, it is important to acknowledge that the Digital Model represents a unique and non-general solution for each specific case study. Any modifications in the process require the adjustment of the current model or, in the worst-case scenario, the development of a new one.
- The Digital Shadow and the Digital Twin represent the digital models that best leverage the potential offered by the technologies constituting Industry 4.0. By employing these digital models within the work environment, it becomes possible to monitor the behaviour of robots involved in various industrial processes in real time. The bidirectional communication

implemented with the Digital Twin facilitates direct interaction with physical devices, enabling the programming of new actions or the implementation of corrective measures in case the current behaviour does not align with the desired one or unforeseen events occur.
Regarding the proposed solutions in this case study, as previously mentioned, they represent only an introductory development of these technologies within the context of the Politecnico di Torino laboratory where this thesis project was conducted. Currently, the most significant limitation for these technologies in this specific case arises from the instability of the wireless connection for implementing the digital models for the UR3 CB3. This necessitates the use of an Ethernet cable connection, which physically restricts the full utilization of the Digital Twin. The obtained latency results demonstrate a value of approximately 0.08s, making these models widely applicable for the purposes demonstrated in this thesis.
Moreover, the availability of the ROS Toolbox in the Matlab library allows for the implementation of real-time control systems, accounting for the latency caused by the connection between devices. This can be achieved by utilizing ROS functions in Matlab, callback functions, and dedicated Simulink blocks.

There are numerous scenarios and developments possible for these technologies. Some future research and project ideas aimed at improving and contributing to the development of the digital technologies presented in this thesis include:

- Conducting a more extensive and detailed analysis of the repeatability and accuracy of the MoMa, encompassing different scenarios and employing a greater number of repetitions to obtain more reliable and precise results.
- Implementing a fully wireless connection for the development of the Digital Twins of different robots, integrating them into a unified system and eliminating the need for physical constraints like Ethernet cables. Additionally, exploring the utilization of a Raspberry Pi for the connection with ROS could be an interesting avenue.
- Developing control systems in Matlab or the Ubuntu environment, leveraging the Python programming system, to detect specific situations and subsequently directing the physical devices to perform specific actions.
- Implementing communication between RVIZ and the 3D simulation environment offered by Gazebo to enable the simulation of scenarios and real-time as well as offline performance monitoring.
- Implementing the Navigation Stack in ROS and RVIZ for the mobile robot, allowing for the sending of multiple goal positions to enable the execution of complete trajectories consisting of different target points.

9 REFERENCES

[1] CULOT, G., NASSIMBENI, G., ORZES, G., & SARTOR, M. (2020). BEHIND THE DEFINITION OF INDUSTRY 4.0: ANALYSIS AND OPEN QUESTIONS. INTERNATIONAL JOURNAL OF PRODUCTION ECONOMICS, 226, 107617.

[2] SCHWAB, K. (2017). THE FOURTH INDUSTRIAL REVOLUTION. CURRENCY.

[3] RADZIWON, A., BILBERG, A., BOGERS, M., & MADSEN, E. S. (2014). THE SMART FACTORY: EXPLORING ADAPTIVE AND FLEXIBLE MANUFACTURING SOLUTIONS. PROCEDIA ENGINEERING, 69, 1184-1190.

[4] ACHOUCH, M., DIMITROVA, M., ZIANE, K., SATTARPANAH KARGANROUDI, S., DHOUIB, R., IBRAHIM, H., & ADDA, M. (2022). ON PREDICTIVE MAINTENANCE IN INDUSTRY 4.0: OVERVIEW, MODELS, AND CHALLENGES. APPLIED SCIENCES, 12(16), 8081.

[5] LI, Z., WANG, Y., & WANG, K. S. (2017). INTELLIGENT PREDICTIVE MAINTENANCE FOR FAULT DIAGNOSIS AND PROGNOSIS IN MACHINE CENTERS: INDUSTRY 4.0 SCENARIO. ADVANCES IN MANUFACTURING, 5, 377-387.

[6] SISINNI, E., SAIFULLAH, A., HAN, S., JENNEHAG, U., & GIDLUND, M. (2018). INDUSTRIAL INTERNET OF THINGS: CHALLENGES, OPPORTUNITIES, AND DIRECTIONS. IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, 14(11), 4724-4734.

[7] DAI, H. N., WANG, H., XU, G., WAN, J., & IMRAN, M. (2020). BIG DATA ANALYTICS FOR MANUFACTURING INTERNET OF THINGS: OPPORTUNITIES, CHALLENGES AND ENABLING TECHNOLOGIES. ENTERPRISE INFORMATION SYSTEMS, 14(9-10), 1279-1303.

[8] MELL, P., & GRANCE, T. (2011). THE NIST DEFINITION OF CLOUD COMPUTING.

[9] HELO, P., SUORSA, M., HAO, Y., & ANUSSORNNITISARN, P. (2014). TOWARD A CLOUD-BASED MANUFACTURING EXECUTION SYSTEM FOR DISTRIBUTED MANUFACTURING. COMPUTERS IN INDUSTRY, 65(4), 646-656.

[10] ÇINAR, Z. M., ABDUSSALAM NUHU, A., ZEESHAN, Q., KORHAN, O., ASMAEL, M., & SAFAEI, B. (2020). MACHINE LEARNING IN PREDICTIVE MAINTENANCE TOWARDS SUSTAINABLE SMART MANUFACTURING IN INDUSTRY 4.0. SUSTAINABILITY, 12(19), 8211.

[11] LEE, J., BAGHERI, B., & KAO, H. A. (2015). A CYBER-PHYSICAL SYSTEMS ARCHITECTURE FOR INDUSTRY 4.0-BASED MANUFACTURING SYSTEMS. MANUFACTURING LETTERS, 3, 18-23.

[12] BAHRIN, M. A. K., OTHMAN, M. F., AZLI, N. H. N., & TALIB, M. F. (2016). INDUSTRY 4.0: A REVIEW ON INDUSTRIAL AUTOMATION AND ROBOTIC. JURNAL TEKNOLOGI, 78(6-13).

[13] RÜßMANN, M., LORENZ, M., GERBERT, P., WALDNER, M., JUSTUS, J., ENGEL, P., & HARNISCH, M. (2015). INDUSTRY 4.0: THE FUTURE OF PRODUCTIVITY AND GROWTH IN MANUFACTURING INDUSTRIES. BOSTON CONSULTING GROUP, 9(1), 54-89.

[14] NGO, T. D., KASHANI, A., IMBALZANO, G., NGUYEN, K. T., & HUI, D. (2018). ADDITIVE MANUFACTURING (3D PRINTING): A REVIEW OF MATERIALS, METHODS, APPLICATIONS AND CHALLENGES. COMPOSITES PART B: ENGINEERING, 143, 172-196.

[15] Masood, T., & Egger, J. (2019). Augmented reality in support of Industry 4.0 Implementation challenges and success factors. Robotics and Computer-Integrated Manufacturing, 58, 181-195.

[16] de Paula Ferreira, W., Armellini, F., & De Santa-Eulalia, L. A. (2020). Simulation in industry 4.0: A state-of-the-art review. Computers & Industrial Engineering, 149, 106868.

[17] Kritzinger, W., Karner, M., Traar, G., Henjes, J., & Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. Ifac-PapersOnline, 51(11), 1016-1022.

[18] Vachálek, J., Bartalský, L., Rovný, O., Šišmišová, D., Morháč, M., & Lokšík, M. (2017, June). The digital twin of an industrial production line within the industry 4.0 concept. In 2017 21st international conference on process control (PC) (pp. 258-262). IEEE.

[19] Lee, J. H., & Hashimoto, H. (2003). Controlling mobile robots in distributed intelligent sensor network. IEEE Transactions on Industrial Electronics, 50(5), 890-902.

[20] Martínez, A., Díez, J., Verde, P., Ferrero, R., Álvarez, R., Perez, H., & Vizán, A. (2021, October). Digital twin for the integration of the automatic transport and manufacturing processes. In IOP Conference Series: Materials Science and Engineering (Vol. 1193, No. 1, p. 012107). IOP Publishing.

[21] Oyekanlu, E. A., Smith, A. C., Thomas, W. P., Mulroy, G., Hitesh, D., Ramsey, M., ... & Sun, D. (2020). A review of recent advances in automated guided vehicle technologies: Integration challenges and research areas for 5G-based smart manufacturing applications. IEEE access, 8, 202312-202353.

[22] Martínez-Gutiérrez, A., Díez-González, J., Ferrero-Guillén, R., Verde, P., Álvarez, R., & Perez, H. (2021). Digital twin for automatic transportation in industry 4.0. Sensors, 21(10), 3344.

[23] Stączek, P., Pizoń, J., Danilczuk, W., & Gola, A. (2021). A digital twin approach for the improvement of an autonomous mobile robots (AMR's) operating environment—A case study. Sensors, 21(23), 7830.

[24] Malik, A. A., & Bilberg, A. (2018). Digital twins of human robot collaboration in a production setting. Procedia manufacturing, 17, 278-285.

[25] Levenson, M. S., Banks, D. L., Eberhardt, K. R., Gill, L. M., Guthrie, W. F., Liu, H. K., ... & Zhang, N. F. (2000). An approach to combining results from multiple methods motivated by the ISO GUM. Journal of research of the National Institute of Standards and Technology, 105(4), 571.

[26] Swales, A. (1999). Open modbus/tcp specification. Schneider Electric, 29, 3-19.

[27] Megalingam, R. K., Teja, C. R., Sreekanth, S., & Raj, A. (2018). ROS based autonomous indoor navigation simulation using SLAM algorithm. International Journal of Pure and Applied Mathematics, 118(7), 199-205.

[28] Soriano, M. (2022). Realizzazione e valutazione della fedeltà del Gemello Digitale di un Manipolatore Mobile.= Design and fidelity evaluation of a Digital Twin of a Mobile Manipulator (Doctoral dissertation, Politecnico di Torino).

10 Figures References

Figure 1: Data flow in Digital Model, Digital Shadow, and Digital Twin.
Source: ResearchGate
https://www.researchgate.net/publication/343885041_Towards_online_adaptation_of_digital_twins/figures?lo=1

Figure 2: Technological pillars of Industry 4.0.
Source: AuraQuatic
Industry 4.0 technologies - AuraQuantic

Figure 4: Mobile Industrial Robot MiR100.
Source: Mobile Industrial Robot
MiR100 (mobile-industrial-robots.com)

Figure 5: The global path is indicated with the dotted blue line and is visible on the map. The local path is indicated with the blue arrow, showing the robot driving around a dynamic obstacle.
Source: MiR100 User's Guide
MiR100 user guide_3.1_en SW2.8.3 rev.3.1 (gibas.nl)

Figure 7: OnRobot RG2 gripper.
Source: OnRobot
Pinza di presa robotica RG2 | OnRobot

Figure 8: Universal Robot UR3, CB3-series.
Source: Universal Robot
UR3e - Robot collaborativo per Automazione | Fino a 3 kg (universal-robots.com)

Figure 11: NatNet SDK network block diagram.
Source: NatNet API User's Guide
NatNet User's Guide (mocap.jp)

Figure 19: Summary diagram of communication between ROS devices and Matlab.
Source: MathWorks
ROS Toolbox Documentation - MathWorks Italia

Figure 41: Differential drive robot and the corresponding block in Simulink. (Left side)
Source: ResearchGate
(PDF) Trajectory-tracking and discrete-time sliding-mode control of wheeled mobile robots (researchgate.net)

Figure 47: 'Pure Pursuit Controller' block in Simulink and Look Ahead parameter limit situations. (Right side)
Source: MathWorks
Pure Pursuit Controller - MATLAB & Simulink - MathWorks Italia

11 Appendix

11.1 Optitrack

11.1.1 NATNET SDK Matlab script

```
period = 20; % ms
task_duration = 3500; % acquisition time = task_duration*100ms

% Create table to save data:
rb_names = ["MIR100","TCP","UR3_base"];
empty_table = table;
name_row = [];
coord_row = [];
for i=1:length(rb_names)
        %name_row = cat(2, name_row, [rb_names(i),'','','','']);
        name_row = cat(2, name_row, [rb_names(i),'','','','','','']);
        %coord_row = cat(2, coord_row,["X","Y","Z","R1","R2","R3","TIME"]);
        coord_row = cat(2, coord_row,["X","Y","Z","R1","R2","R3","TIME"]);
end

intestazione = cat(1, name_row, coord_row);
intestazione = array2table(intestazione);
empty_table = [empty_table; intestazione];

% Create table to save data:
[rigid1,rigid2,rigid3,data_Optitrack] = NatNetPollingSampleCustom_xyphi(length(rb_names), freq,
empty_table_durata_task);
```

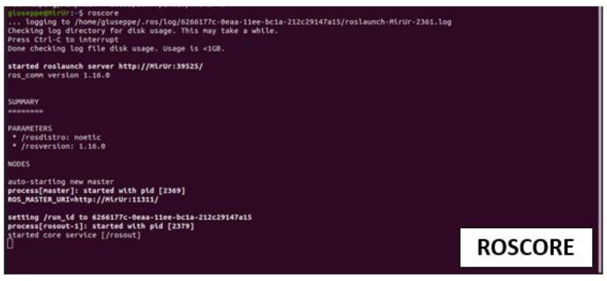
```
function [rb1,rb2,rb3,final_table] = NatNetPollingSampleCustom_xyphi(nbodies, sample_time,
data_table,task_time)

rb1=rigid_Body();
rb2=rigid_Body();
rb3=rigid_Body();
%% [... Optitrack System Configuration
% create an instance of the natnet client class
natnetclient = natnet;
% connect the client to the server (multicast over local loopback) - modify for your network
natnetclient.HostIP = '[...]';
natnetclient.ClientIP = '[...]';
natnetclient.ConnectionType = 'Multicast';
natnetclient.connect;
% ...]

%% Actions:
rb_pose = [];
pose_row = [];
%Timestamp:
timestamp = datestr(clock, "HH:MM:SS.FFF");
split_time = split(timestamp, ':');
h_in_sec = str2double(split_time(1)).*60^2;
m_in_sec = str2double(split_time(2)).*60;
sec=str2double(split_time(3));
poll_time = h_in_sec+m_in_sec+sec;
for k=1:nbodies
            %Orientation:
            q_x = data.RigidBodies(k).qx;
            q_y = data.RigidBodies(k).qy;
            q_z = data.RigidBodies(k).qz;
            q_w = data.RigidBodies(k).qw;
            q = quaternion(q_x, q_y, q_z, q_w);
            angles = EulerAngles(q,'xyz')*360/(2*pi);
            phi = [angles(1), angles(2), angles(3)];
            %Pose:
            pose = [data.RigidBodies(k).x*1000,data.RigidBodies(k).y*1000,data.RigidBodies(k).z*1000, phi];
            pose_row = cat(2, pose_row, pose, poll_time);
end
rb_pose = cat(1, rb_pose, pose_row);
end
```

11.1.2 Optitrack data convertion Matlab script

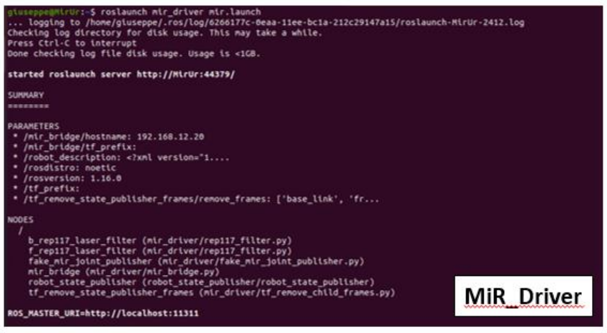
```matlab
scaling_factor = 1.297;
load('C:\Users\Desktop\test4_pose_opt.mat');

x_opt = data_Optitrack.intestazione10(3:end);
y_opt = data_Optitrack.intestazione8(3:end);
phi_opt = data_Optitrack.intestazione12(3:end);
t_opt = data_Optitrack.intestazione14(3:end);
x_opt = str2double(x_opt)/1000;
y_opt = str2double(y_opt)/1000;
x_opt_offset = 30.9085;
y_opt_offset = 19.3607;
x_opt = (x_opt + x_opt_offset)*scaling_factor;
y_opt = (y_opt + y_opt_offset)*scaling_factor;
x_opt = smooth(x_opt);
y_opt = smooth(y_opt);
phi_opt = str2double(phi_opt);
phi_opt = -phi_opt;
t_opt = str2double(t_opt);
t_opt = t_opt-t_opt(1);
table_opt = table(x_opt,y_opt,phi_opt,t_opt);
%writetable(table_opt,'C:\Users\Desktop\test_data_opt.xlsx')

figure(5)
plot(x_opt,y_opt,'b')
grid on
figure(6)
plot(t_opt,x_opt,'r')
grid on
figure(7)
plot(t_opt,y_opt,'k')
grid on
figure(8)
plot(t_opt,phi_opt,'g')
grid on
```

11.2 ROS INITIAL SETTINGS

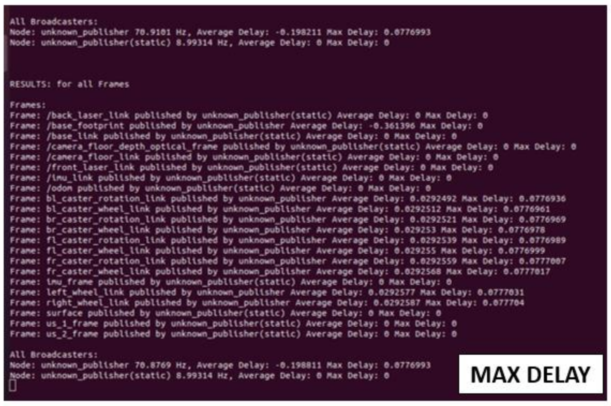
11.2.1 Time Synchronization between ROS machine and MiR100



```
giuseppe@MirUr:~$ roscore
... logging to /home/giuseppe/.ros/log/6266177c-0eaa-11ee-bc1a-212c29147a15/roslaunch-MirUr-2361.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://MirUr:39525/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [2369]
ROS_MASTER_URI=http://MirUr:11311/

setting /run_id to 6266177c-0eaa-11ee-bc1a-212c29147a15
process[rosout-1]: started with pid [2379]
started core service [/rosout]
```



```
giuseppe@MirUr:~$ roslaunch mir_driver mir.launch
... logging to /home/giuseppe/.ros/log/6266177c-0eaa-11ee-bc1a-212c29147a15/roslaunch-MirUr-2412.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://MirUr:44379/

SUMMARY
========

PARAMETERS
 * /mir_bridge/hostname: 192.168.12.20
 * /mir_bridge/tf_prefix:
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /tf_prefix:
 * /tf_remove_state_publisher_frames/remove_frames: ['base_link', 'fr...

NODES
  /
    b_rep117_laser_filter (mir_driver/rep117_filter.py)
    f_rep117_laser_filter (mir_driver/rep117_filter.py)
    fake_mir_joint_publisher (mir_driver/fake_mir_joint_publisher.py)
    mir_bridge (mir_driver/mir_bridge.py)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    tf_remove_state_publisher_frames (mir_driver/tf_remove_child_frames.py)

ROS_MASTER_URI=http://localhost:11311
```



```
All Broadcasters:
Node: unknown_publisher 70.9101 Hz, Average Delay: -0.198211 Max Delay: 0.0776993
Node: unknown_publisher(static) 8.99314 Hz, Average Delay: 0 Max Delay: 0



RESULTS: for all Frames

Frames:
Frame: /back_laser_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /base_footprint published by unknown_publisher Average Delay: -0.361396 Max Delay: 0
Frame: /base_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /camera_floor_depth_optical_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /camera_floor_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /front_laser_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /imu_link published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: /odom published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: bl_caster_rotation_link published by unknown_publisher Average Delay: 0.0292492 Max Delay: 0.0776936
Frame: bl_caster_wheel_link published by unknown_publisher Average Delay: 0.0292512 Max Delay: 0.0776961
Frame: br_caster_rotation_link published by unknown_publisher Average Delay: 0.0292521 Max Delay: 0.0776969
Frame: br_caster_wheel_link published by unknown_publisher Average Delay: 0.029253 Max Delay: 0.0776978
Frame: fl_caster_rotation_link published by unknown_publisher Average Delay: 0.0292539 Max Delay: 0.0776989
Frame: fl_caster_wheel_link published by unknown_publisher Average Delay: 0.029255 Max Delay: 0.0776999
Frame: fr_caster_rotation_link published by unknown_publisher Average Delay: 0.0292559 Max Delay: 0.0777007
Frame: fr_caster_wheel_link published by unknown_publisher Average Delay: 0.0292568 Max Delay: 0.0777017
Frame: imu_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: left_wheel_link published by unknown_publisher Average Delay: 0.0292577 Max Delay: 0.0777031
Frame: right_wheel_link published by unknown_publisher Average Delay: 0.0292587 Max Delay: 0.077704
Frame: surface published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: us_1_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: us_2_frame published by unknown_publisher(static) Average Delay: 0 Max Delay: 0

All Broadcasters:
Node: unknown_publisher 70.8769 Hz, Average Delay: -0.198811 Max Delay: 0.0776993
Node: unknown_publisher(static) 8.99314 Hz, Average Delay: 0 Max Delay: 0
```

11.2.2 ROS nodes initialization Matlab script

```
rosshutdown
% rosinit(rosIP);
setenv('ROS_IP','[...]') % Substitute the square brackets with the IP Address of the Matlab host PC
setenv('ROS_MASTER_URI', '[...]') % Substitute the square brackets with the IP Address of the MIR or VM
with ROS. The IP address is equal to 'http://192.168.xx.xx:11311'
rosinit

% For MiR Digital Shadow in Simulink, see block Robot Visualizer
logicalMap = scaledimage;
checkpoints = [starting_point(1:2); assemblyStn(1:2);unloadingStn(1:2);chargingStn(1:2)];
```

11.2.3 Simulink function to extract MiR position and velocity

```
function [map_position, notscaled_position, orientation] = extractPose(model_states)

X = model_states.Position.X;
Y = model_states.Position.Y;
Xo = model_states.Orientation.X;
Yo = model_states.Orientation.Y;
Zo = model_states.Orientation.Z;
Wo = model_states.Orientation.W;
orientation = [Xo,Yo,Zo,Wo];
eul = quat2eul([Wo,Xo,Yo,Zo]);
yaw = eul(1);

notscaled_position = [X,Y,yaw];
map_position = [X*1.297,Y*1.297,yaw];
end
```

```
function cmd_vel = extractVel(model_states)
% con IP_MIR --> geometry_msg/TwistStamped
vx = model_states.Twist.Linear.X;
vy = model_states.Twist.Linear.Y;
vz = model_states.Twist.Linear.Z;
wx = model_states.Twist.Angular.X;
wy = model_states.Twist.Angular.Y;
wz = model_states.Twist.Angular.Z;

cmd_vel = [vx,vy,vz,wx,wy,wz];


% con IP_VM --> geometry_msg/Twist
% vx = model_states.Linear.X;
% vy = model_states.Linear.Y;
% vz = model_states.Linear.Z;
% wx = model_states.Angular.X;
% wy = model_states.Angular.Y;
% wz = model_states.Angular.Z;
%
% cmd_vel = [vx,vy,vz,wx,wy,wz];
end
```

11.2.4 ROS Bag file management Matlab script

```
bag = rosbag("ros_realTime_MOMAsim_071223_18_53_05.bag");
%bag.MessageList

% split data from the bag
% pose
mir_rosBagPose = select(bag,"Topic","/subsystem_model_states");
% read messages
msgs_pose = readMessages(mir_rosBagPose,"DataFormat","struct");
% timeseries with topic components
% X-Y pose
timeseries_rosBagPose = timeseries(mir_rosBagPose,"Position.X","Position.Y");
%timeseries_rosBagPose.Data;
% Orientation
timeseries_rosBagOrient =
timeseries(mir_rosBagPose,"Orientation.X","Orientation.Y","Orientation.Z","Orientation.W");
%size(msgs_pose);
eul =
quat2eul([timeseries_rosBagOrient.Data(:,4),timeseries_rosBagOrient.Data(:,1),timeseries_rosBagOrient.Data(
:,2),timeseries_rosBagOrient.Data(:,3)]);
phi = eul(:,1);
% x-y position (from /robot_pose topic)
pose_ros_x = timeseries_rosBagPose.Data(:,1);
pose_ros_y = timeseries_rosBagPose.Data(:,2);
mir_pose_ros = cat(2,pose_ros_x,pose_ros_y);

% velocities
mir_rosBagCmdVel = select(bag,"Topic","/subsystem1_model_states");
% timeseries for velocities
timeseries_rosBagCmdVel = timeseries(mir_rosBagCmdVel,"Twist.Linear.X","Twist.Angular.Z");
% lin and ang velocities (from /cmd_vel topic)
lin_vel_ros = timeseries_rosBagCmdVel.Data(:,1);
ang_vel_ros = timeseries_rosBagCmdVel.Data(:,2);
```

11.3 Digital Model

11.3.1 Binary Occupancy Grid Matlab script

```
%CREATION OF THE LAB MATLAB MAP
image = imread('new_lab_4.png');
grayimage = im2gray(image);
bwimage = grayimage < 9;
scaledimage = imresize(bwimage, 0.06); % Tuned it measuring the real width of the corridor (3m) 0.06
map = binaryOccupancyMap(scaledimage);
```

11.3.2 Task Class

```
classdef Tasks < Simulink.IntEnumType
   enumeration
      Arm_Pick (0)
      Arm_Place (1)
      Arm_Home (2)
      Robot_Navigate (3)
      Idle (4)
   end
end
```

11.3.3 CheckAtStnRange Simulink function

```
function atStnRange = checkAtStnRange(targetPose,currentPose)
atStnRange = 0;
    if(targetPose(3)==0||abs(targetPose(3))==pi)
        if(abs(currentPose(2)-targetPose(2))<0.08 && abs(currentPose(1)-targetPose(1))<0.3)
            atStnRange = 1;
        else
            atStnRange = 0;
        end
    elseif(abs(targetPose(3))==pi/2)
        if(abs(currentPose(1)-targetPose(1))<0.08 && abs(currentPose(2)-targetPose(2))<0.3)
            atStnRange = 1;
        else
            atStnRange = 0;
        end
    end
end
```

11.3.4 ThetaQuarterSelection and AngVelSignController Simulink functions

```
function situation = thetaQuarterSelection(scaledPhi,theta)
sit = 0;
if(scaledPhi*theta>=0)
    if(theta>scaledPhi)
        sit=100; % w>0
    elseif(theta<scaledPhi)
        sit=200; % w<0
    elseif(theta==scaledPhi)
        sit=300; % oriented
    end
else
    if(theta<pi/4 && theta>=0) % first half - first quarter
        sit=11;
    elseif(theta<pi/2 && theta>=pi/4) % second half - first quarter
        sit=12;
    elseif(theta<3*pi/4 && theta>=pi/2) % first half - second quarter
        sit=21;
    elseif(theta<pi && theta>=3*pi/4) % second half - second quarter
        sit=22;
    elseif(theta<-3*pi/4 && theta>=-pi) % first half - third quarter
        sit=31;
    elseif(theta<-pi/2 && theta>=-3*pi/4) % second half - third quarter
        sit=32;
    elseif(theta<-pi/4 && theta>=-pi/2) % first half - fourth quarter
        sit=41;
    elseif(theta<0 && theta>=-pi/4) % second half - fourth quarter
        sit=42;
    elseif((theta==pi && scaledPhi==-pi) || (theta==-pi && scaledPhi==pi))
        sit=400;
    end
end

situation = sit;
end
```

```
function [w_pos, w_neg, oriented] = angVelSignController(scaledPhi,sit,theta)

w_pos=0;
w_neg=0;
oriented=0;

switch(sit)
    case 100
        w_pos=1;
        w_neg=0;
        oriented=0;
    case 200
        w_pos=0;
        w_neg=1;
        oriented=0;
    case 300
        w_pos=0;
        w_neg=0;
        oriented=1;
    case 400
        w_pos=0;
        w_neg=0;
        oriented=1;
    case 11
        if(scaledPhi<0 && scaledPhi>=-pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi<-pi/4 && scaledPhi>=-pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-pi/2 && scaledPhi>=-3*pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi<-3*pi/4 && scaledPhi>=-pi)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi==0 && theta~=0)
            w_pos=1;
            w_neg=0;
            oriented=0;
        end
    case 12
        if(scaledPhi<=0 && scaledPhi>=-pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi<-pi/4 && scaledPhi>=-pi/2)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi<-pi/2 && scaledPhi>=-3*pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-3*pi/4 && scaledPhi>=-pi)
            w_pos=0;
            w_neg=1;
            oriented=0;
        end
```

```
case 21
        if(scaledPhi<=0 && scaledPhi>=-pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi<-pi/4 && scaledPhi>=-pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-pi/2 && scaledPhi>=-3*pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-3*pi/4 && scaledPhi>=-pi)
            w_pos=0;
            w_neg=1;
            oriented=0;
        end
    case 22
        if(scaledPhi<=0 && scaledPhi>=-pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-pi/4 && scaledPhi>=-pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-pi/2 && scaledPhi>=-3*pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi<-3*pi/4 && scaledPhi>=-pi)
            w_pos=0;
            w_neg=1;
            oriented=0;
        end
    case 31
        if(scaledPhi>=0 && scaledPhi<pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/4 && scaledPhi<pi/2)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi>=pi/2 && scaledPhi<3*pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi>=3*pi/4 && scaledPhi<pi)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi==-pi && theta~=-pi)
            w_pos=1;
            w_neg=0;
            oriented=0;
        end
```

```
case 32
        if(scaledPhi>=0 && scaledPhi<pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/4 && scaledPhi<pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/2 && scaledPhi<3*pi/4)
            w_pos=1;
            w_neg=0;
            oriented=0;
        elseif(scaledPhi>=3*pi/4 && scaledPhi<=pi)
            w_pos=1;
            w_neg=0;
            oriented=0;
        end
    case 41
        if(scaledPhi>=0 && scaledPhi<pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/4 && scaledPhi<pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/2 && scaledPhi<3*pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=3*pi/4 && scaledPhi<=pi)
            w_pos=1;
            w_neg=0;
            oriented=0;
        end
    case 42
        if(scaledPhi>=0 && scaledPhi<pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/4 && scaledPhi<pi/2)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=pi/2 && scaledPhi<3*pi/4)
            w_pos=0;
            w_neg=1;
            oriented=0;
        elseif(scaledPhi>=3*pi/4 && scaledPhi<=pi)
            w_pos=0;
            w_neg=1;
            oriented=0;
        end
end
end
```

11.3.5 SetSignLinVel Simulink function

```
function sign_v = setSignLinVel(perpend,currentPose,targetPose)
sign_v = 1;
if(perpend==1)
    if(targetPose(3)==0)
        if(currentPose(1)<targetPose(1))
            sign_v = 1;
        elseif(currentPose(1)>targetPose(1))
            sign_v = -1;
        else
            sign_v = 1;
        end
    elseif(abs(targetPose(3))==pi)
        if(currentPose(1)<targetPose(1))
            sign_v = -1;
        elseif(currentPose(1)>targetPose(1))
            sign_v = 1;
        else
            sign_v = 1;
        end
    elseif(targetPose(3)==pi/2)
        if(currentPose(2)<targetPose(2))
            sign_v = 1;
        elseif(currentPose(2)>targetPose(2))
            sign_v = -1;
        else
            sign_v = 1;
        end
    elseif(targetPose(3)==-pi/2)
        if(currentPose(2)<targetPose(2))
            sign_v = -1;
        elseif(currentPose(2)>targetPose(2))
            sign_v = 1;
        else
            sign_v = 1;
        end
    end
else
    sign_v = 1;
end
end
```

11.3.6 UR3goalReached and GripperGoalReached Simulink function

```
function successLocal = UR3goalReached(taskCurrentLocal, taskTargetLocal, tolerance)
    successLocal = false;

    diffWpts = zeros(length(taskCurrentLocal(:,1)),1);
    diffWpts(:,:) = abs(taskCurrentLocal-taskTargetLocal);
    errorStatus = diffWpts < tolerance;
    if all(errorStatus)
            successLocal = true; % goal achieved
    end
end
```

```
function successLocal = GripperGoalReached(gripperCurrentState, gripperTargetState)
    successLocal = false;

    grip_check = abs(gripperCurrentState-gripperTargetState);

    if(grip_check==0)
        successLocal = true;
    end
end
```

11.3.7 UR3_TCP Matlab Script

```matlab
Pick_checkpoints = readtable('TCP_pick_checkpoints_new2.xlsx');
Place_checkpoints = readtable('TCP_place_checkpoints_new2.xlsx');
% Checkpoints
x_check_pick = (Pick_checkpoints.X);
y_check_pick = (Pick_checkpoints.Y);
z_check_pick = (Pick_checkpoints.Z);
x_check_place = (Place_checkpoints.X);
y_check_place = (Place_checkpoints.Y);
z_check_place = (Place_checkpoints.Z);

% Simulation
% PICK:
x_pick_sim = -(simulation.x_TCP.signals.values(indx_pick(1):indx_mir(2)-1));
y_pick_sim = -(simulation.y_TCP.signals.values(indx_pick(1):indx_mir(2)-1));
z_pick_sim = (simulation.z_TCP.signals.values(indx_pick(1):indx_mir(2)-1));
x_pick_sim = x_pick_sim.*1000;
y_pick_sim = y_pick_sim.*1000;
z_pick_sim = z_pick_sim.*1000;
% PLACE :
x_place_sim = (simulation.x_TCP.signals.values(indx_place(1):indx_mir(3)-1));
y_place_sim = (simulation.y_TCP.signals.values(indx_place(1):indx_mir(3)-1));
z_place_sim = (simulation.z_TCP.signals.values(indx_place(1):indx_mir(3)-1));
x_place_sim = x_place_sim.*1000;
y_place_sim = y_place_sim.*1000;
z_place_sim = z_place_sim.*1000;

% Node-red
nodered_data_tcp = readtable('C:\Users \Desktop\ nr_tcp.xlsx');
nodered_pick_tcp = [nodered_data_tcp.Var1(115:608), nodered_data_tcp.Var2(115:608),
nodered_data_tcp.Var3(115:608)];
nodered_place_tcp = [nodered_data_tcp.Var1(609:end), nodered_data_tcp.Var2(609:end),
nodered_data_tcp.Var3(609:end)];
% Pick:
x_pick_nr = (nodered_pick_tcp(:,1));
y_pick_nr = (nodered_pick_tcp(:,2));
z_pick_nr = (nodered_pick_tcp(:,3));
nr_pose_pick = [x_pick_nr, y_pick_nr, z_pick_nr];
% Place:
x_place_nr = (nodered_place_tcp(:,1));
y_place_nr = (nodered_place_tcp(:,2));
z_place_nr = (nodered_place_tcp(:,3));
nr_pose_place = [x_place_nr, y_place_nr, z_place_nr];

% nr_offset:
dz_check_pick = z_pick_nr(1) - z_check_pick(1);
z_check_pick = z_check_pick + dz_check_pick;
dz_check_place = z_place_nr(1) - z_check_place(1);
z_check_place = z_check_place + dz_check_place;

% Optitrack:
x_tcp_opt = data_Optitrack.intestazione15(3:end);
y_tcp_opt = data_Optitrack.intestazione16(3:end);
z_tcp_opt = data_Optitrack.intestazione17(3:end);
% Pick:
x_pick_opt = smooth(str2double(x_tcp_opt(368:1443)));
y_pick_opt = smooth(str2double(y_tcp_opt(368:1443)));
z_pick_opt = smooth(str2double(z_tcp_opt(368:1443)));

x_base_pick_opt = (smooth(str2double(data_Optitrack.intestazione1(494))));
y_base_pick_opt = (smooth(str2double(data_Optitrack.intestazione2(494))));
z_base_pick_opt = (smooth(str2double(data_Optitrack.intestazione3(494))));
% Place:
x_place_opt = smooth(str2double(x_tcp_opt(2165:2932)));
y_place_opt = smooth(str2double(y_tcp_opt(2165:2932)));
z_place_opt = smooth(str2double(z_tcp_opt(2165:2932)));

x_base_place_opt = (smooth(str2double(data_Optitrack.intestazione1(2165))));
y_base_place_opt = (smooth(str2double(data_Optitrack.intestazione2(2165))));
z_base_place_opt = (smooth(str2double(data_Optitrack.intestazione3(2165))));
```

```
% Erease Base offset:
x_tcp_pick_opt = -(x_pick_opt - x_base_pick_opt);
y_tcp_pick_opt = y_pick_opt - y_base_pick_opt;
dz_opt_pick = z_pick_nr(1) - y_tcp_pick_opt(1);
y_tcp_pick_opt = y_tcp_pick_opt + dz_opt_pick;
z_tcp_pick_opt = -(z_pick_opt - z_base_pick_opt);
%
x_tcp_place_opt = x_place_opt - x_base_place_opt;
y_tcp_place_opt = y_place_opt - y_base_place_opt;
dz_opt_place = z_place_nr(1) - y_tcp_place_opt(1);
y_tcp_place_opt = y_tcp_place_opt + dz_opt_place;
z_tcp_place_opt = z_place_opt - z_base_place_opt;

% sim offset:
dz_sim_pick = z_pick_nr(1) - z_pick_sim(1);
z_pick_sim = z_pick_sim + dz_sim_pick;

dz_sim_place = z_place_nr(1) - z_place_sim(1);
z_place_sim = z_place_sim + dz_sim_place;
```