Department of Electronics and Telecommunications
Master's Degree Course in Electronic Engineering

Integrated control of multiple robots using the ROS framework

Supervisor:
Prof. Dario Antonelli

Candidate:
Dario Maschio

Academic year 2023/2024

Contents

Abstract

Industrial automation has advanced quickly in recent years thanks to the creation of more intelligent and adaptable robots that can work in dynamic settings where people coexist. Mobile Manipulators (MOMA) are one of these new technologies that are vital to many industrial areas. With a MOMA, a single system may navigate its surroundings and manipulate things by combining the mobility of a robotic mobile platform with the accuracy and adaptability of a manipulator arm. Numerous tasks, including assembly, flexible manufacturing support, and internal logistics, call for the usage of this kind of robots, representing a significant advantage in terms of efficiency and productivity.
Proprietary software platforms are provided by several of the mobile robots and collaborative manipulators on the market, including the MiR100 and UR3. Despite their strength, these software systems are frequently closed and have a restricted range of features. Because of this, users are limited by the programming logic that manufacturers impose, which can limit their options when setting up and integrating the system in particular production settings, which is an issue for some new major requirements.
This problem can be solved by using an open-source robotic operating system like ROS (Robot Operating System). ROS is a very versatile platform that offers libraries and tools for creating unique control algorithms, making it easier to integrate sensors and actuators, and enabling high-level programming based on sophisticated logic, including motion planning and artificial intelligence.
In this thesis has been developed a ROS framework for the mobile manipulator, which is made up of 'Universal Robots' UR3 collaborative manipulator and the MiR100 industrial mobile robot. Making the system highly configurable while utilizing ROS's flexibility and modularity was the primary objective. has been established a software architecture that permits to have flexible and scalable programming, aiming to have a system that not only is able to execute the standard tasks, but that is able to handle predictable non standard situations that may occur in a line of production.
While the MOMA is performing the assigned duties, all the movements of it and of the operator that collaborates with it in the working environment are tracked, so that there is a complete vision of all the elements and situations in a working environment, on which some optimization can be done if needed.

1 Introduction

1.1 Mobile Manipulator in the industry

A recently developed kind of robotic systems known as mobile manipulators, or mobile manipulator robots (MOMA), combines the characteristics of mobile robots with those of fixed robotic arms. These systems' development can be attributed back to the growing need for increased automation job flexibility and efficiency, especially in contexts where navigation and manipulation skills are required[6].

1.1.1 Early Development of Mobile Manipulators

Early developments in robotics in the 1960s and 1970s[21], when researchers aimed to integrate mobile platforms with robotic arms to carry out complicated tasks in dynamic situations, can be traced back to the idea of mobile manipulators. By combining these two technologies, robot autonomy was increased and they were able to function outside of designated work areas.
Due to the requirement to combine mobility with manipulation in order to increase the capabilities of both systems, the first mobile manipulators were created. One of the first examples in the industrial field was the "Hilare 2bis" (fig.(1)), built in 1992 in France, by the 'Laboratory for Analysis and Architecture of Systems'[29]

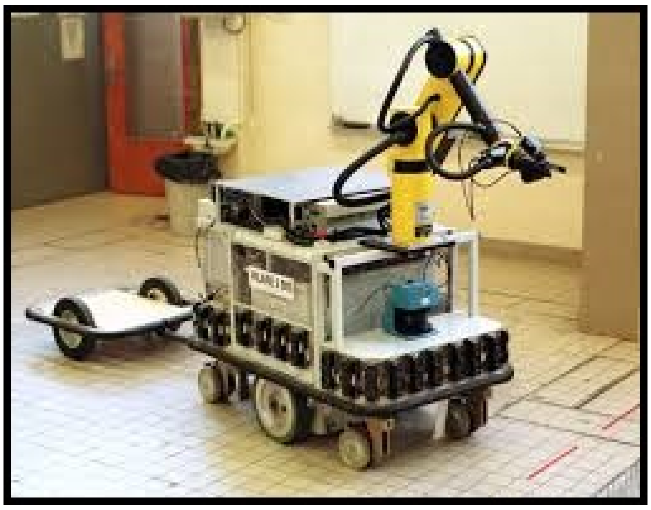


Figure 1: Hilare2bis mobile manipulator[10].

1.1.2 The Role of MOMAs in Industry 4.0

The advent of Industry 4.0 signified a momentous shift in the acceptance of mobile manipulators. The term "industry4.0" refers to the fourth industrial revolution, which is defined by the incorporation of advanced data analytics, digitalization, the Internet of Things (IoT), and cyber-physical systems into production and manufacturing processes. Given this, MOMAs' versatility and adaptability make them perfect for dynamic, automated settings that call for both mobility and highly accurate manipulation[6].
Mobile manipulators are not limited to a specific workplace, in contrast to conventional stationary robots. They are capable of independently navigating hospital rooms, warehouses, and industrial floors. They may move between workstations and interact with objects and machinery as needed. Because of this, they are extremely useful in just-in-time production settings, where the capacity to promptly adjust to shifting demands and processes is essential to preserving productivity. For instance, MOMAs can perform inspections, move parts to assembly lines, and even help with assembly in the automotive industry, greatly lowering the need for manual labor and boosting throughput.
Another key feature of MOMAs in Industry 4.0 is their collaborative nature. Equipped with advanced sensors, such as lidar, 3D cameras, and tactile sensors, MOMAs can safely operate alongside human workers. This collaborative capability enables human-robot interaction (HRI) in tasks that require both human intelligence and robotic precision. MOMAs can take over repetitive or physically demanding tasks, while human workers focus on more complex, decision-making aspects of the operation, fostering a symbiotic work environment[2].
Additionally, MOMAs support the integration of cutting-edge technologies like machine learning (ML) and artificial intelligence (AI). Robots can now optimize their movements, learn from previous interactions, and make

judgments in real time based on their environment thanks to these technologies. A mobile manipulator with a vision system, for example, may plot its trajectory, identify objects, and make real-time adjustments to avoid obstacles or reposition itself for more effective handling. AI, precision manipulation, and autonomous mobility come together to create a highly adaptable system that may be used in a variety of industrial applications[1].

1.1.3 Applications and Benefits in the Industries

Many different industries are adopting mobile manipulators because of their unique set of capabilities. MOMAs are used in warehousing and logistics (fig.2) for picking and packaging, inventory control, and product transportation. Their capacity to carry out activities usually associated with fixed robots while moving independently around big storage facilities improves operational efficiency.

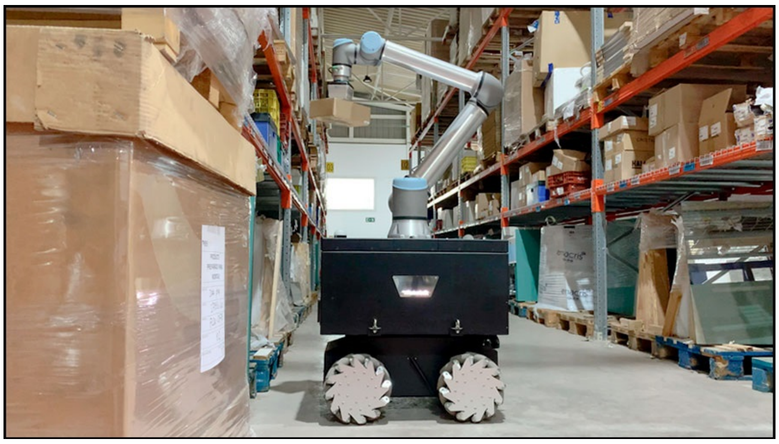


Figure 2: Application of a Mobile Manipulator in logistic application[23].

Another application of the MOMAs can be seen On assembly lines, where stationary manipulators are highly common. But occasionally, modifications to the production line are required. Since mobile manipulators are more adaptable and efficient than stationary manipulators, industry could benefit from increased production and lower costs. Additionally, the automobile industry relies heavily on assembly, and several mobile manipulators are made just for that purpose. which, at the moment, need human labor to move bulky or heavy components.[31].

1.2 Objectives of the thesis

The aim of this project is to configure an optimized environment where a MOMA works with more flexibility . Starting from a working environment, a Finite State Machine (FSM) has been created in Simulink, which is installed inside ROS, that has to execute all the tasks of a line of production, considering non-standard situations that may occur during the performance of the task, so the MOMA is able to correct those situations autonomously.
While doing that, in order to optimize the system to the best, an acquisition system has been set up to monitor and acquire data of the movements of the operator and the Mobile Manipulator, so that can be studied the interaction between the worker, the MOMA, and the overall conduct of work, in such a way that could also be possible to improve the arrangement of the working environment.
In order to do so, the FSM has to communicate with the mobile manipulator, so a network has been established between the Computer and the MOMA.
The Mobile Manipulator used to complete these tasks is composed by a Mobile Industrial Robot (MIR100) and by a collaborative arm robot (UR3), with an RG2 gripper, provided by OnRobot, attached to the end effector.

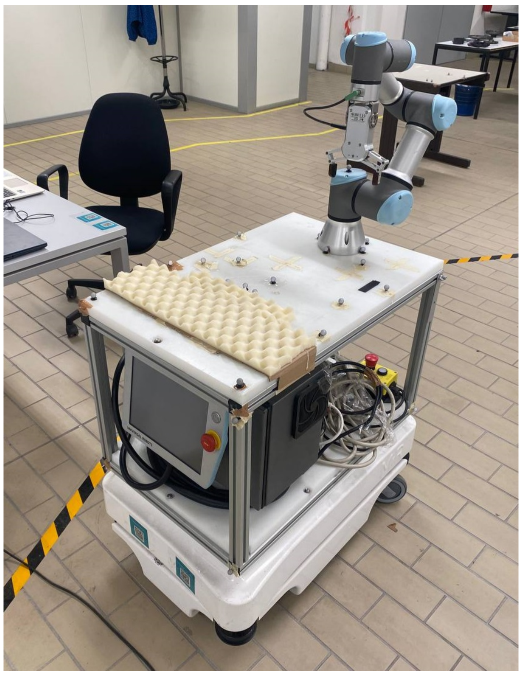


Figure 3: Mobile Manipulator (MoMa).

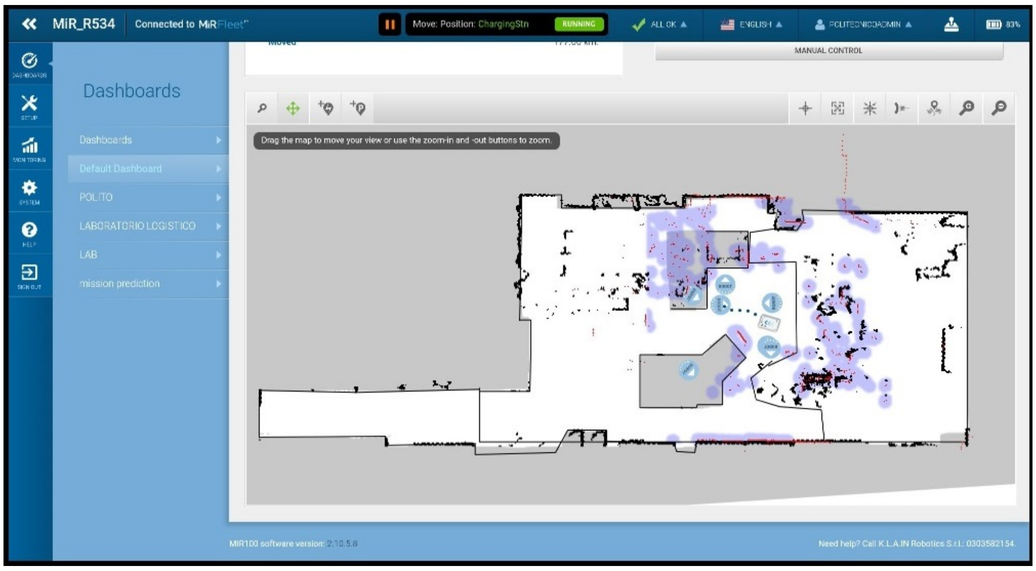
1.2.1 MIR100

The MiR100 is a mobile robot that automates transportation and logistics within the company quickly and affordably. It streamlines processes by distributing resources among employees, which contributes to higher output and lower expenses.

Figure 4: Mobile Industrial Robot MIR100.

The MIR100 main features of interest for this project are:

- Driving in a populated workspace: the robot is able to operate safely in dynamic environments;
- Overall route planning and local adjustments: the robot's movement is focused on finding the most optimized path to its destination; it can also adjust the trajectory when an obstacle occurs in the current path;
- Efficient transportation of heavy loads: the robot is able to transport loads up to 100 kg;
- Internal map: the robot uses a map that is created by manually driving it around the working environment. While is mapping the robot can detect walls, doors, furniture and other obstacles, in the mean time it creates, based on the previous detection, a map, which is then editable in the MIR software.



Drag the map to move your view or use the zoom-in and -out buttons to zoom.

MIR100 software version: 2.10.5.6

Need help? Call K.L.A.IN Robotics S.r.l.: 0303582154.

Figure 5: MIR100 live map.

1.2.2 UR3

Compact and versatile, the Universal Robot UR3 collaborative robot (cobot) is intended for automation and precision work in a variety of industries.
The UR3, created by Universal Robots, is noted for its lightweight construction and adaptable deployment, which makes it the perfect choice for applications needing fine detail or small-scale production situations.

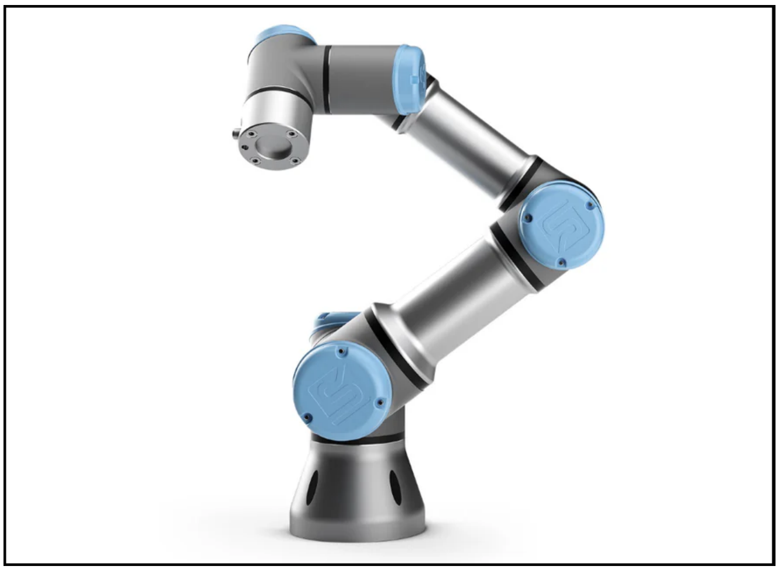


Figure 6: Universal Robot UR3 CB3-series.

The UR3 is designed for high-precision operations as testing, assembly, and pick-and-place procedures. With its six degrees of freedom, it can replicate the handiness of a human arm, and its integrated safety features facilitate close cooperation with human workers.
Additionally, the UR3 may be programmed via an easy-to-use interface that considerably lowers the operator learning curve and facilitates rapid and effective automation integration through the teach pendant. This interface is called Polyscope (fig.(7)), which is the graphical interface that allows the user to program, control and monitor the robot.

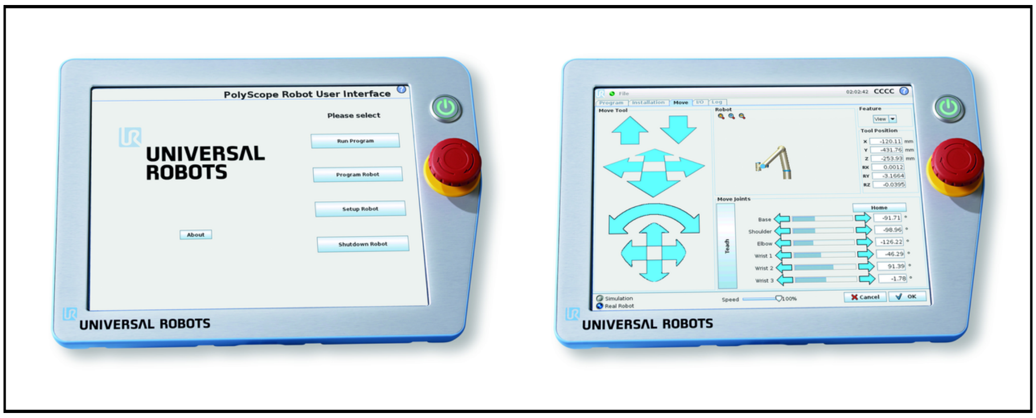


Figure 7: Teach pendant of UR3 CB3-series robot with Polyscope interface[3].

Because of its small size and simplicity of use, it is a well-liked option for businesses trying to streamline production procedures while preserving flexibility and worker safety.
The UR3 main features for this project are[1]:

- Reach: 500 mm from base joint;
- Payload: 3kg;
- Weight: 11.2 kg;
- Every joint rotate in a $\pm 360^{\circ}$ range;

[1]UR3 CB-series datasheet:https://www.universalrobots.com/media/1828034/ur3_tech_spec_web_en.pdf

1.2.3 GR2 Gripper

The OnRobot GR2 Gripper is a flexible and accurate end-effector made specifically to be used with collaborative robots. With the focus on adaptability and user-friendliness, this two-finger electric gripper is designed to handle a variety of tasks, including pick-and-place, assembly, and machine tending.

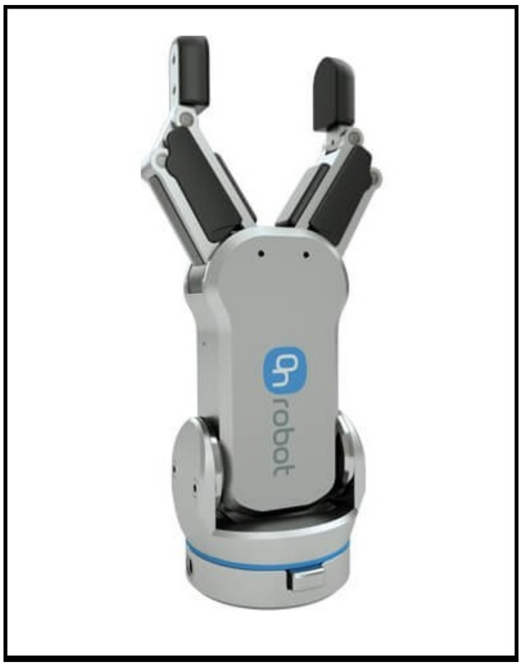


Figure 8: OnRobot GR2 Gripper.

The GR2's plug-and-play connection, which enables speedy setup with a variety of robotic arms, including those from Universal Robots, is one of its best features. The gripper doesn't need complicated control systems or a great deal of modification in order to be installed and configured.
It is perfect for handling objects of various sizes and fragilities, from delicate components to more robust things, thanks to its adjustable gripping force and stroke range. With the precise control it offers over the gripping action, the GR2 Gripper guarantees that objects are held securely and unharmed.
Furthermore, because the gripper is electricly powered, it does not require complicated pneumatic systems or an external air supply, which lowers maintenance and operating expenses. It is ideal for many industrial settings because of its small size, which also makes it space-efficient.

2 Software and Tools utilized

The Software and Tools used for this project are the following:

- ROS (Robot Operating System)
- MATLAB
- Simulink
- Optitrack Motive

2.1 Robot Operating System

The open-source software package known as ROS, or Robot Operating System, was created to facilitate the creation of robotic applications. It is a framework that offers a collection of tools and libraries for robot development rather than an operating system in the conventional sense. Its primary characteristics are as follows[24]:

- Middleware: Within a robotic system, ROS serves as a mediator by enabling communication between various nodes, or code modules. Python and C++ are the most common programming languages used to create these nodes;
- Development Tools: ROS comes with a number of tools to make data visualization, debugging, and programming easier. For instance, rqt offers a graphical user interface (GUI) for managing and displaying data, while rviz is a visualization tool that lets you view real-time sensor and node data;
- Package Management: Software is arranged and managed by ROS via a package system. Nodes, libraries, configuration files, and other tools required for a particular capability may be included in each package;
- Hardware Abstraction: ROS offers hardware abstractions, allowing you to develop code that interacts with many kinds of sensors and actuators without having to deal with the intricacies of the hardware itself;
- Community and Ecosystem: ROS is benefited by an important development and research community that provides packages, tools, and documentation because it is an open-source project. This ecosystem speeds up the integration of new technologies and cuts down on development time;
- Interoperability: ROS facilitates communication between nodes via a range of message and service protocols, enabling disparate system components to cooperate with one another.

2.1.1 ROS Architecture

The main elements that compose the architecture of ROS are:

- **Nodes**: The nodes are processes used to execute the codes in ROS. Every node compute different functions and they can communicate because of the ROS network.
  A node may be in charge of executing a planning algorithm, managing an actuator, or reading data from a sensor. One node may, for instance, read data from a camera, and a another node could process it to recognize objects.
  The communication between different nodes takes place through mechanisms as Topics, Services and Actions.
- **Topics**: The topics are communications channels that permits to publish and subscribe messages between the Nodes.
  Those are the features of the Topics:
  - Publish-Subscribe model: A Node can send (publish) or receive (subscribe) a message on or from a Topic. A Topic can have different Nodes publishing and subscribing messages into it.
  - Anonymous Communication: During the exchange of messages, the Nodes communicating through the same channel don't know who is sending or receiving the messages, decoupling the Nodes and making the network more scalable.
  - Message Type: every Topic is marked by his own message type, so every message sent to a specific topic must adhere to his message type.

- Asynchronous communication: The messages can be published and subscribed at different times and at different frequencies, so they operate independently.

- **Services**: The Services are used when a Remote Procedure Call interaction is needed, which is a request/reply interaction. A Service is defined by a pair of messages, one for the request and one for the reply. A ROS Node provides a service, where the client can call it by sending a request message and waiting for the response [25].

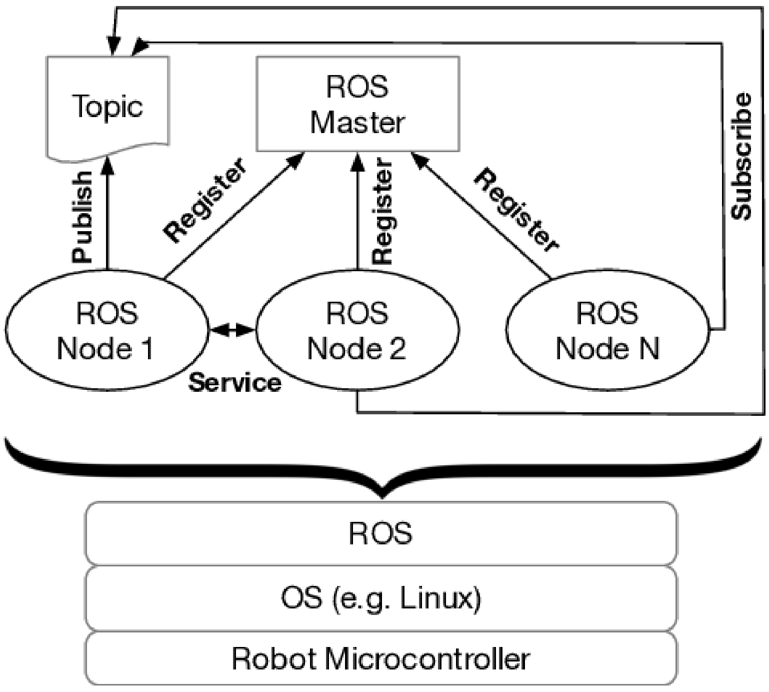
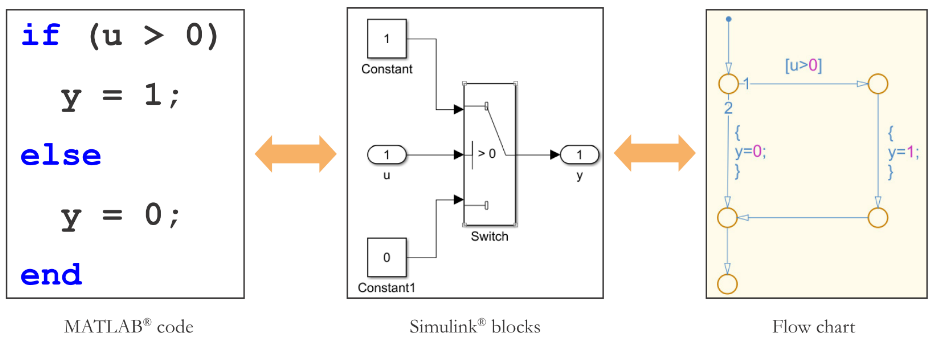


Figure 9: Communication between Nodes [18].

2.2 MATLAB and Simulink

MATLAB is a C-written environment for statistical analysis and numerical computation that also comes with the same-named programming language developed by MathWorks. With MATLAB, one may work with matrices, visualize data and functions, apply algorithms, design user interfaces, and communicate with other software. With the aid of special toolboxes to be presented later, it is able to communicate with external devices, as Mobile Robots and Collaborative ones used in this project.
Simulink is a block diagram environment that is used to design systems with multi-domain models, which means that before moving on to hardware and deploying the system without writing code, the user can represent different physical models, such as electrical and mechanical[17].
For this project, the most important design environment included in Simulink is StateFlow.



```
if (u > 0)
  y = 1;
else
  y = 0;
end
```

Figure 10: Modeling Flow charts [27].

State flow permits to model and simulate systems with control logic and state-based behavior. It allows to combine continuous or discrete systems (simulated in Simulink) with event- or state-based control logic (modeled in Stateflow).
Here are the main features of StateFlow:

- Finite State Machine Modeling: It can represent systems based on state transitions in response to circumstances or events. Flow charts, states, and transitions can all be used to explain how the system behaves.
- Control Logic: It assists in implementing intricate control logic in Simulink models, such as systems that are event- or condition-based. Systems with conditional reactions frequently employ it.
- Events and Timed Actions: You can designate timed actions, or prerequisites that must be satisfied, in order for a state change to take place, as well as events that cause transitions between states.
- Visual Diagrams: It is possible to build the behavior of the systems utilizing State diagrams and flow charts, which are visually straightforward and simple to comprehend. this leads to an easier system to analyse and debug.
- Automation and Simulation: Once the StateFlow model is created is possible to simulate it in Simulink. It is also possible to generate a C or HDL code from the model, with which can be simulated in other platforms.

As already mentioned, the previous applications are able to communicate with the MOMA using the Toolboxes, that will be explained in the hereafter.

2.2.1 MATLAB Toolboxes

The MATLAB Toolboxes utilized for this project are:

- **ROS Toolbox**: It provides an interface between MATLAB, Simulink and ROS. It is possible to use the toolbox to create a network of ROS nodes and merge your current ROS network with nodes created using Simulink or MATLAB[14].

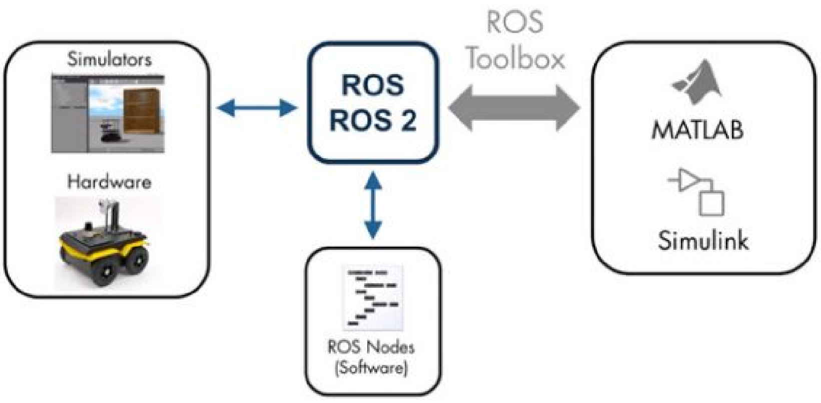


Figure 11: Connection between MATLAB/Simulink and ROS through ROS Toolbox.[14]

By recording, importing, and playing back rosbag files, users can see and analyze ROS data using the toolbox's MATLAB functions and Simulink blocks. To access ROS messages, one can also establish a connection with an active ROS network.

- **Robotics System Toolbox**: The Robotics System Toolbox offers tools and techniques for manipulator and mobile robot applications, including design, simulation, testing, and deployment. The toolkit for manipulators consists of rigid-body tree representation-based algorithms for dynamics, path planning, trajectory development, direct and inverse kinematics, and collision control. It comprises mapping, localization, path planning, path following, and motion control algorithms for mobile robots.
  It is possible to test the design on hardware. To accomplish this, just produce and distribute the code (using MATLAB Coder) and connect to robotic platforms like the Universal Robots UR series, which is used in this project[13].

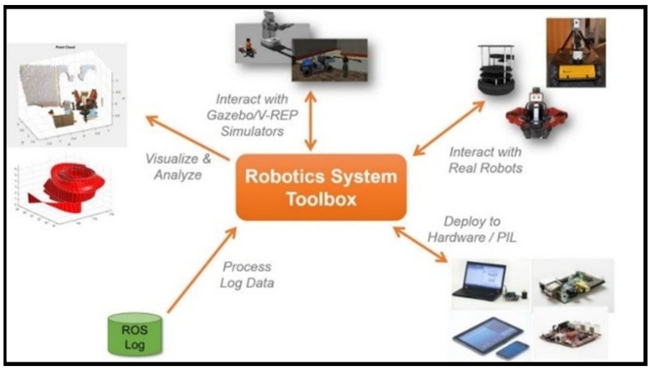


Figure 12: Connection betweeb MATLAB/Simulink to external devices through the Robotics System Toolbox.[4]

- **Robotics System Toolbox™ Support Package for Universal Robots UR Series Manipulators**: It is possible to develop algorithms and run simulations with the Robotics System Toolbox Support Package for Universal Robots UR Series Manipulators. This package works with the rigid body tree environment, Gazebo, and URSim simulators. Additionally, is possible to test and evaluate your algorithms by connecting to Universal Robots hardware with this companion package. tha user is also able to use urROSNode to automatically generate C++ code and build an independent ROS node that you can install on an Ubuntu Linux host system[16]. The supported Hardware with this Toolbox are:
  - Universal Robots E Series Manipulators;
  - Universal Robots CB Series Manipulators, which are the one used in this project.

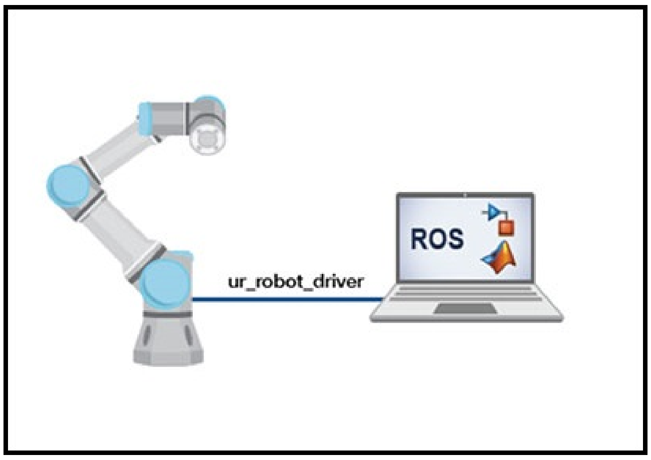


Figure 13: Connection between MATLAB and the physical UR robot.[28]

2.3 Optitrack Motive

OptiTrack produced Motive, a motion capture software that uses a system of high-precision infrared cameras (Fig.(14)) to collect, monitor, and analyze motion data, detecting the position and orientation of their dedicated markers (Fig.(14)) placed on the object or person to detect. allowing users to capture the movement of objects or people with high accuracy.

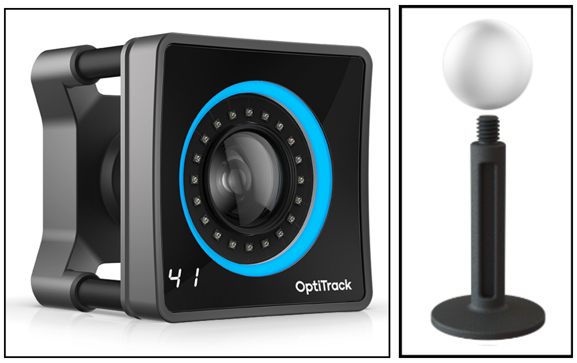


Figure 14: Sx: Optitrack's infrared camera Dx: Optitrack's marker.

In the following, features provided by the official website [2] about Motive are shown below:

- Marker Tracking: Motive is capable of tracking over 1,500 markers at one time.
- Rigid Body Tracking: Track over 300 rigid bodies in real-time, with the following parameters:
  - Positional accuracies typically ± 0.2mm or less.
  - Rotational accuracies typically ± 0.1deg or less.
  - Latency < 10ms.
- Calibration : This parameter implies the actual accuracy of the system. The calibration is done calculating the 3D position and the lens distortion of cameras with an intuitive interface and easy to understand visuals.
- Data Export: Motive supports exporting motion capture data to various file formats, which can be integrated into other software platforms like animation programs or, in this case, analysis tools (e.g. MATLAB).

[2]Optitrack official website: https://optitrack.com/software/motive/specs.html

2.3.1 Connection between Optitrack and MATLAB

The connection between Optitrack and MATLAB is needed to capture and process the data acquired by Motive during the operations, and is possible through the NatNet SDK, which is a Software Development Kit that allows to stream data from Motive to another application in real-time. This also allows you to control some aspects of Motive remotely.

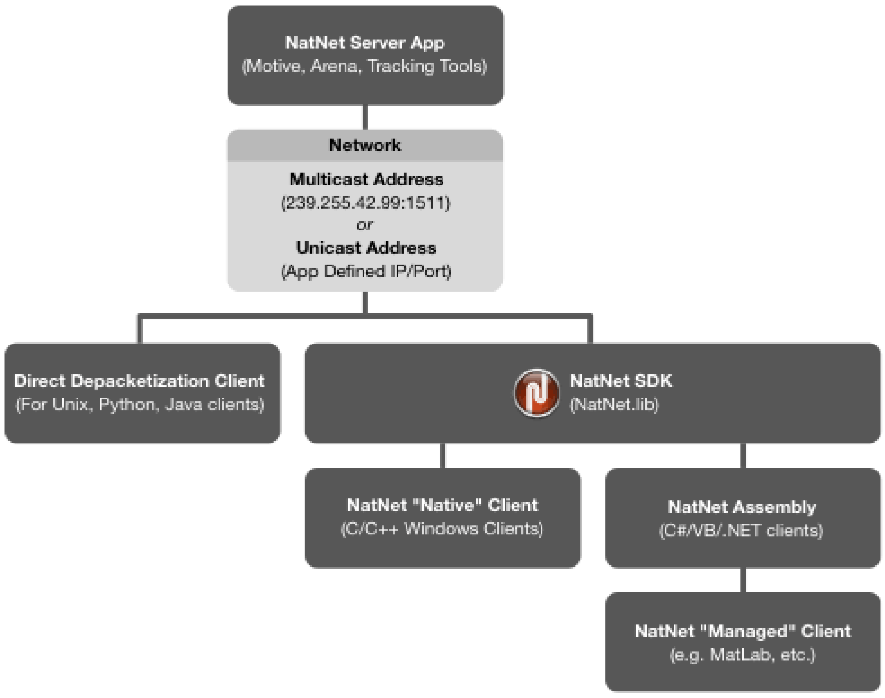


Figure 15: A client/server networking SDK for streaming motion tracking data across networks[20].

To connect Motive and MATLAB trough the NatNet SDK and to capture the Motive coordinates during the work and process them in MATLAB, pre-made scripts were used[9][3] and adapted to this project(Appendix1.5-Appendix1.6).

[3]Giuseppe La Spisa, Design and evaluation of different digital control technologies of a mobile manipulator, Politecnico di Torino, 2023, p.81

3 Case study

The project has been simulated in a production line, consisting in the assembly of a skateboard, in a working environment where human and robot are collaborating. So, the MOMA will take from a warehouse the pieces needed, not paying too much attention on the positioning of the objects on the MIR platform because the aim is to decrease the accuracy of the action but increasing the success rate of them, because the most important thing is to deliver the objects to the worker, that will assemble and store them after some quality and safe checks.

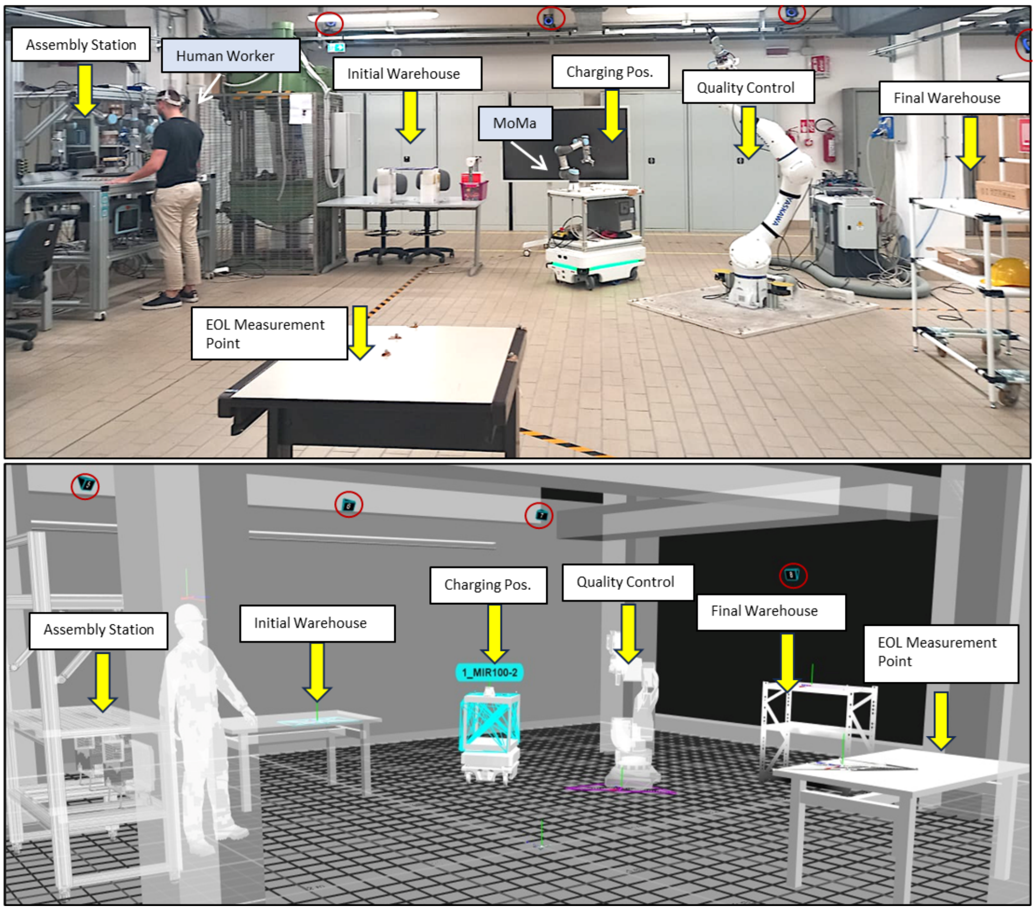


Figure 16: Mind4Lab Laboratory and Mind4Lab laboratory in Optitrack Motive.

3.1 Layout of the working environment

The layout for this project is composed by six stations:

1. **Charging Station**
   Is the starting and ending point of the line of production, where the MOMA waits for the task.
   If the robot is not in the Charging Station position, it goes there for different cases, when the operator gives to the robot the command to execute the tasks of the line of production, if it is not already in the charging position, automatically moves there and then will start the first task.
   It may also happen that, during the execution, the battery percentage goes under a certain limit (that can be set), in that case, the robot will interrupt the current job and will move in the Charging Station, waiting until the battery is over an appropriate percentage.

2. **Initial Warehouse**
   In this station, the MOMA has to pick the parts of the skateboard and place them on his shelf.
   This action is divided in two tasks, the first one consists in picking the skateboard in a certain position of the warehouse, in the second task the robot has to move to the location where the wheels of the skateboard are stored and pick them.
   Once all the components has been picked and places, can be transported to the next station.

3. **Assembly Station**
   In this station there is the first interaction between the human and the robot.

The worker will wait the components from the robot, before starting to assemble them he will give some tools to the MOMA that will transport them to the assigned station.
by the time the human has completed the assembly of the skateboard, the robot already went back, waiting for the assembled piece that will be transported to the next station.

4. **End Of Line Measurement Point (EOL) Station**
The mobile manipulator has two different tasks here, the first one consist of dropping the tools that the operator previously provided, then the robot will be back in this station with the assembled skateboard, at this point there is the first collaboration between the human and the robot.
With the help of the tools transported previously by the robot, the human gives a quick check of the skateboard, looking for coarse errors.
This process is done with the help of the MOMA, which will hold the piece for the operator for the time of the inspection.
Once finished, if the check is successful, both human and robot will keep their duty in the assigned stations, otherwise the worker will report it to the robot, that will go back to the Assembly Station for the adjustment.

5. **Quality Control Station**
In the following station there is the second and last collaboration between human and robot.
The MOMA has to execute the same task as before, so it has the task of holding the assembled skateboard for the operator that in this case will do more accurate quality checks.
As before, if the quality control does not find any problems, the operator will go back to the Assembly Station and the robot will head to the next and final station.
If some problems arise, the line of production will be stopped by the operator in order to solve the problem or discard the product.

6. **Final Warehouse station**
This is the final station of the line of production, where the mobile manipulator will deposit the finished product before going back to the charging station in order to restart the cycle.
While the robot goes back to the charging point, the operator will reach the Final Warehouse to pack the product.

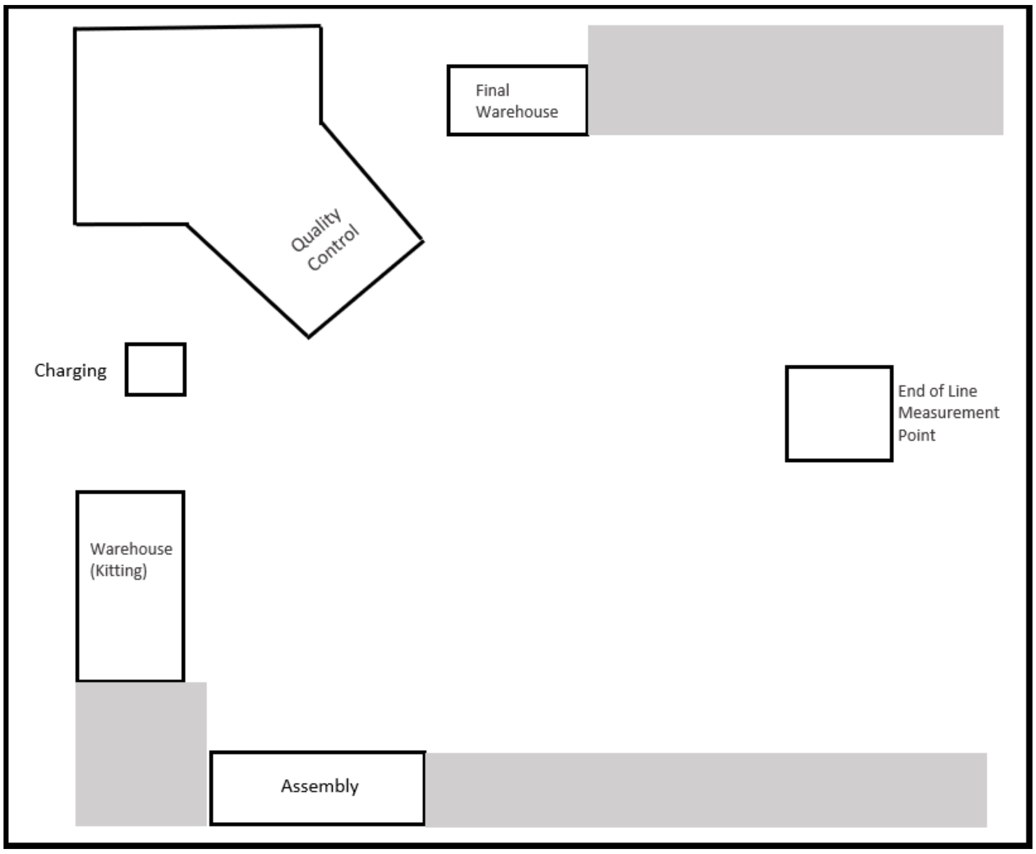


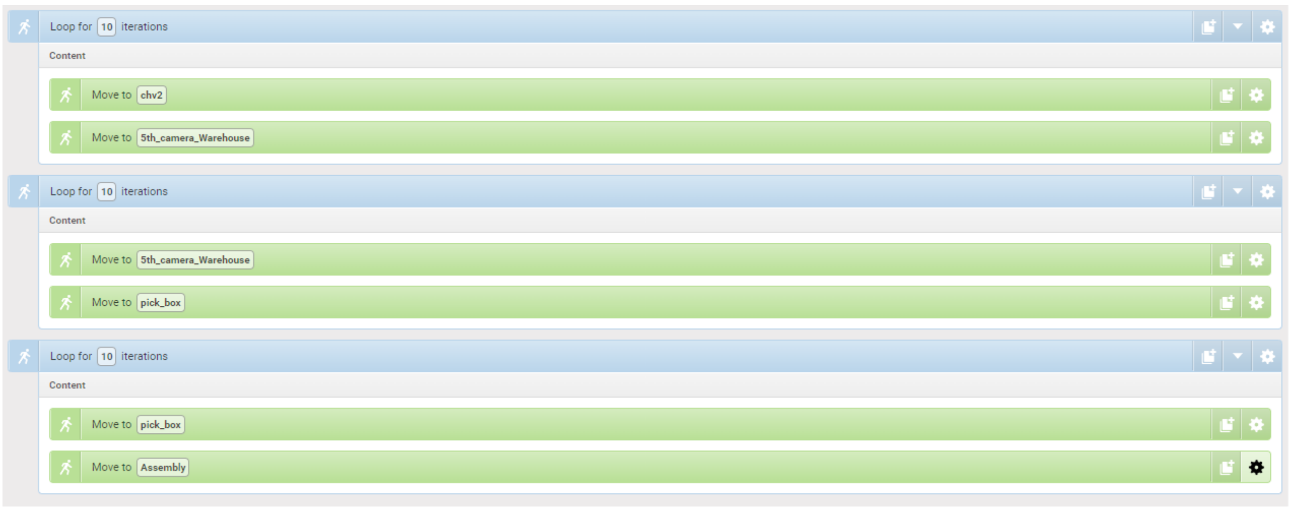
Figure 17: Working environment top view.

3.2 Component selection

The Mobile Manipulator, as explained before, is composed by different robots, with different accuracies.
For the UR3 the accuracy is negligible for the purposes of this project, but for the MIR100 has been discovered that the accuracy is crucial for the selection of the components of the line of production.

3.2.1 MIR100 accuracy

For every station except the charging one, has been monitored the center point of the Mobile robot once reached the station, this process has been repeated ten times.



Loop for 10 iterations
Content
Move to chv2
Move to 5th_camera_Warehouse
Loop for 10 iterations
Content
Move to 5th_camera_Warehouse
Move to pick_box
Loop for 10 iterations
Content
Move to pick_box
Move to Assembly

Figure 18: Script for three stations to study the accuracy of the MIR100.

The coordinates captured by Motive has been processed with MATLAB and presented in Excel, giving the following results:

- Initial Warehouse:
  - Pick Skate position:

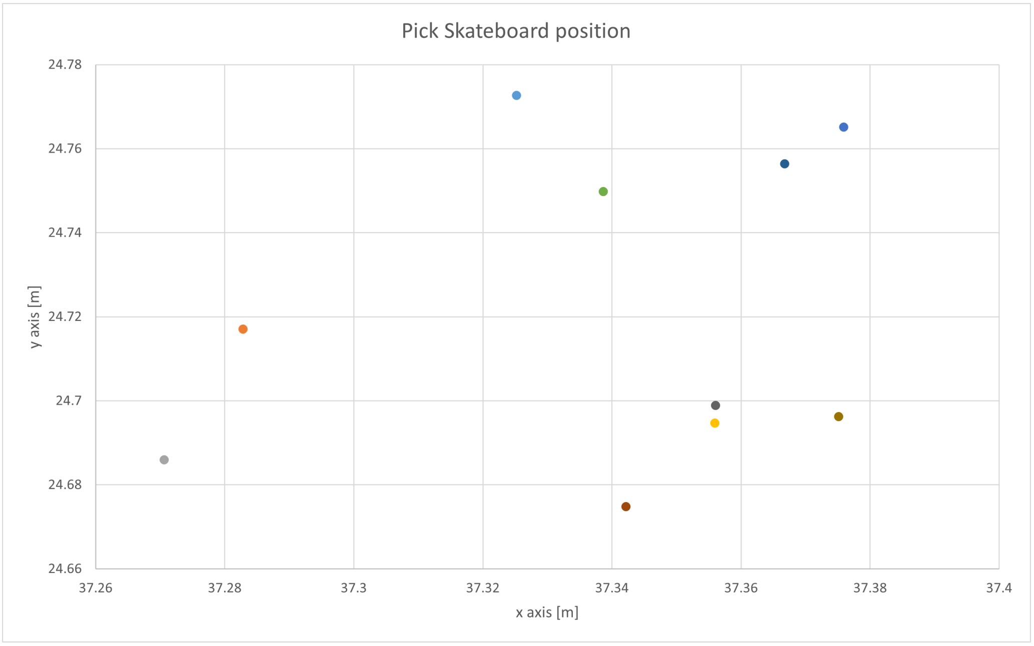


Figure 19: Ten arrival points of the MIR100 in the Skateboard picking point in the Initial Warehouse.

– Pick Box position:

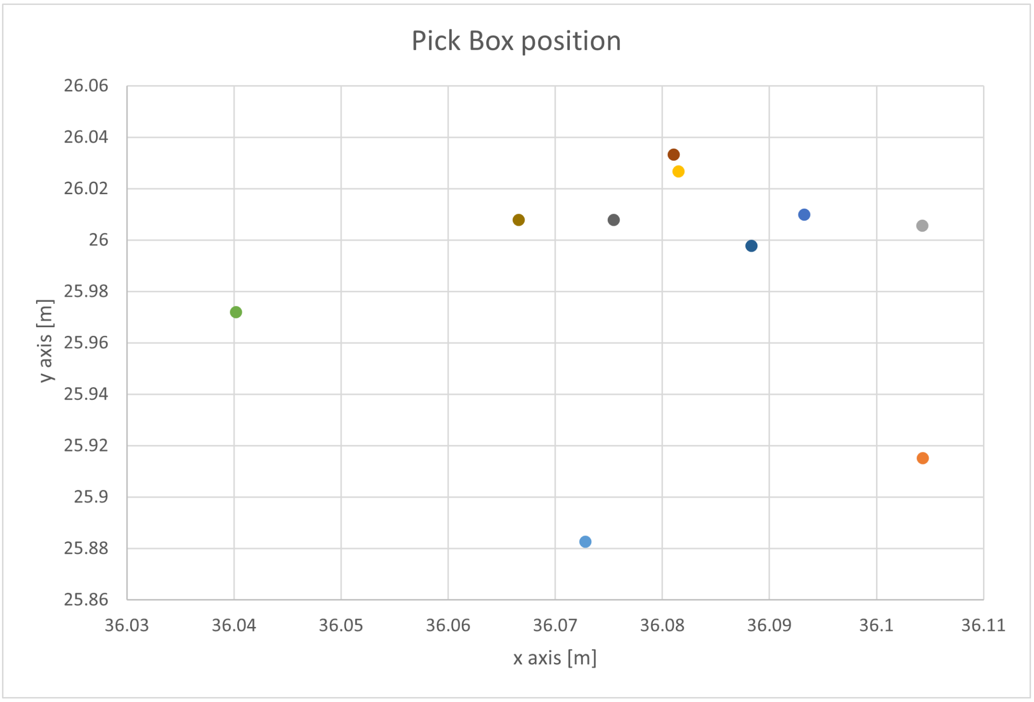


Figure 20: Ten arrival points of the MIR100 in the Box picking point in the Initial Warehouse.

- Assembly Station:

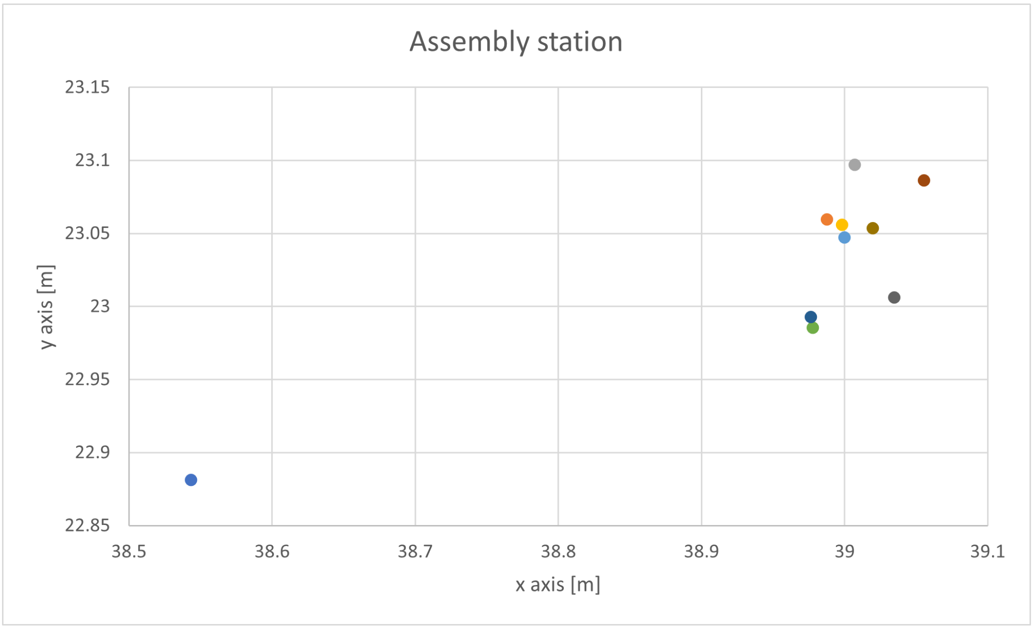


Figure 21: Ten arrival points of the MIR100 in the Assembly station.

- End Of Line Measurement Point (EOL) Station:

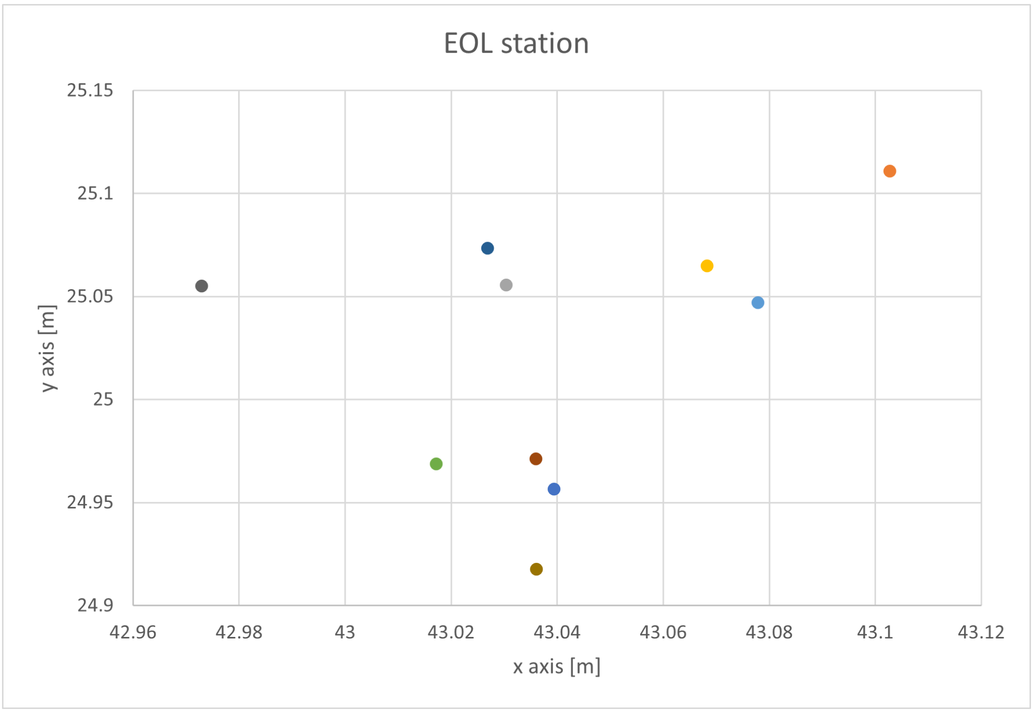


Figure 22: Ten arrival points of the MIR100 in the EOL station.

- Quality Control Station:

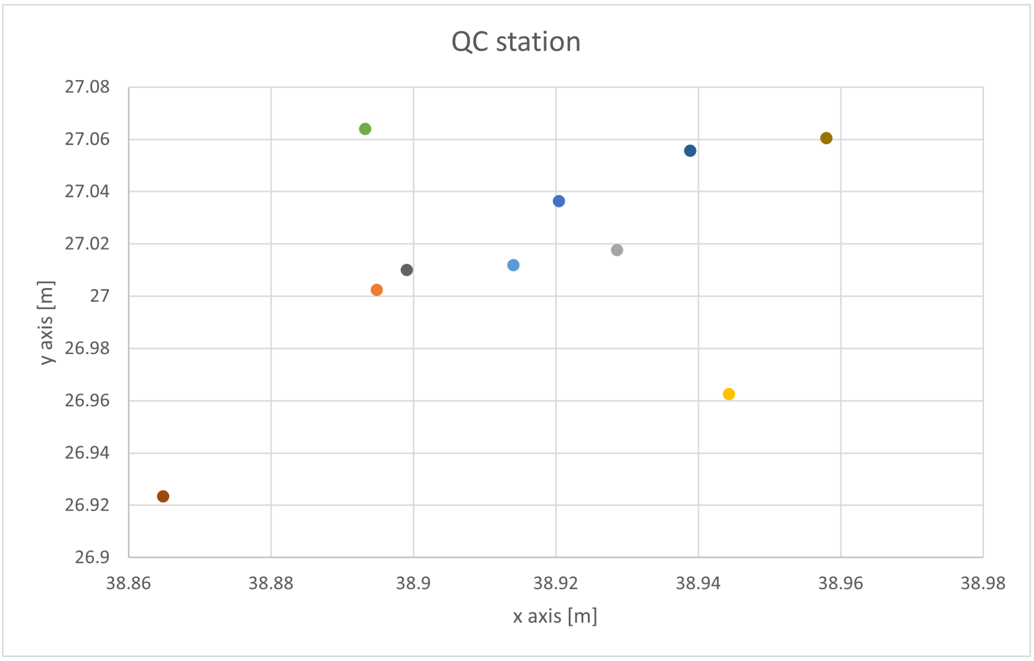


Figure 23: Ten arrival points of the MIR100 in the QC station.

- Final Warehouse:

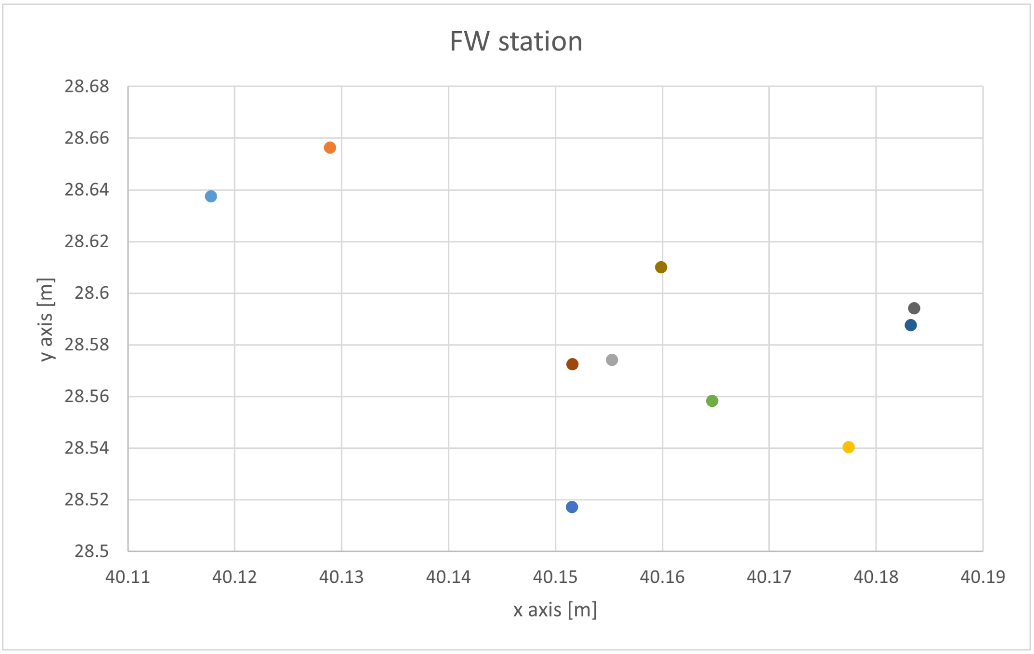


Figure 24: Ten arrival points of the MIR100 in the FW .

As can be noticed in the previous charts, the accuracy in positioning of the MIR100 is $\pm 10cm$, which is a significant value, because in a pick and place application, if the working piece is small (e.g. 2cm) the efficiency of the operation is compromised, because the probability of failing is high.
this led to the decision to simulate the assembly of a skateboard (Fig.(25)), as its dimensions were within the accuracy range of the MIR.

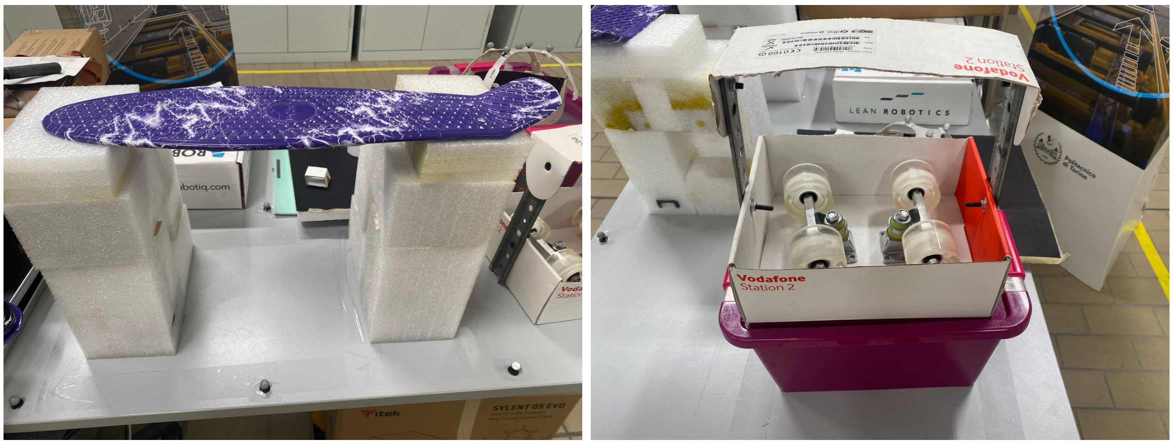
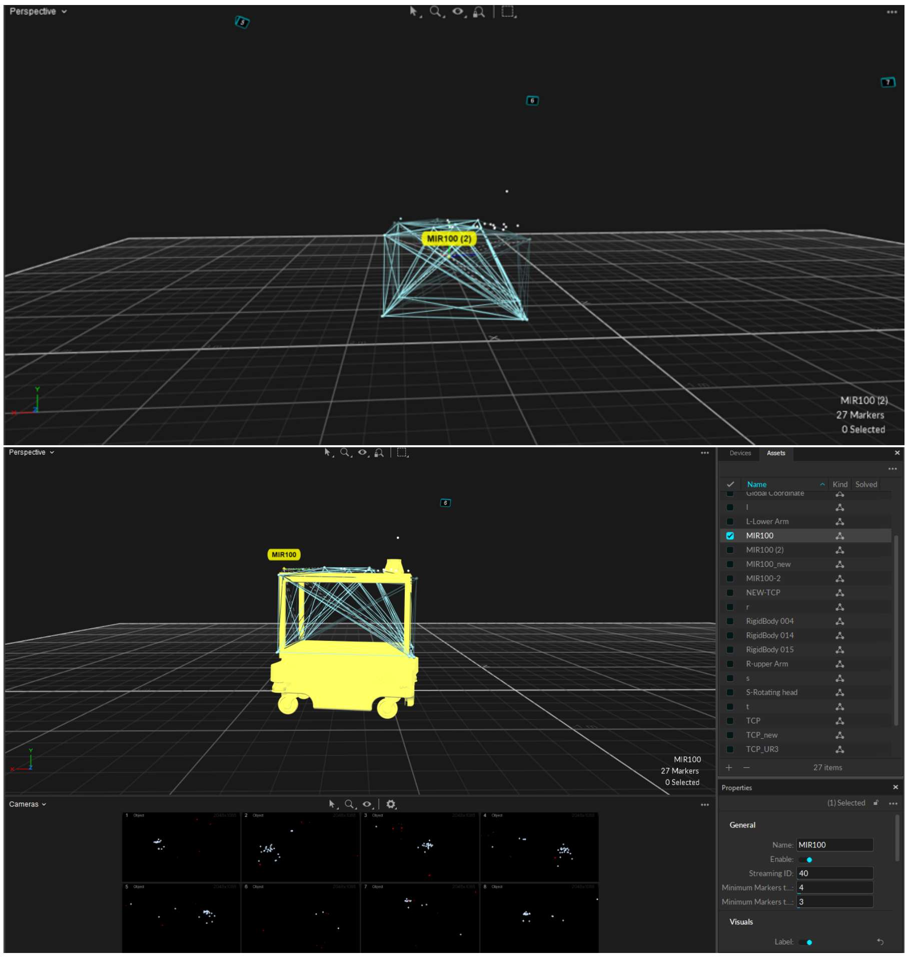


Figure 25: Skateboard and Wheels used for the assembly simulation .

3.3 Motive rigid bodies

To track in real time the MOMA and the Human has been used the markers cited before(Chapter 2.3) placed in a specific position:

- **MOMA**
  On the Mobile Manipulator the markers has been placed on top and on its sides, to have the center point as close as possible to the physical one.



✓	Name	Kind	Solved
	Global Coordinate		
	l		
	L-Lower Arm		
✓	MIR100		
	MIR100 (2)		
	MIR100_new		
	MIR100-2		
	NEW-TCP		
	r		
	RigidBody 004		
	RigidBody 014		
	RigidBody 015		
	R-upper Arm		
	s		
	S-Rotating head		
	t		
	TCP		
	TCP_new		
	TCP_UR3		

27 items

Properties

(1) Selected

General

Name: MIR100
Enable: ●
Streaming ID: 40
Minimum Markers t...: 4
Minimum Markers t...: 3

Visuals

Label: ●

Figure 26: MIR100 rigid body representation in Motive[8].

- **Human**
  To track the human has been used a rigid hat with five markers placed on top.

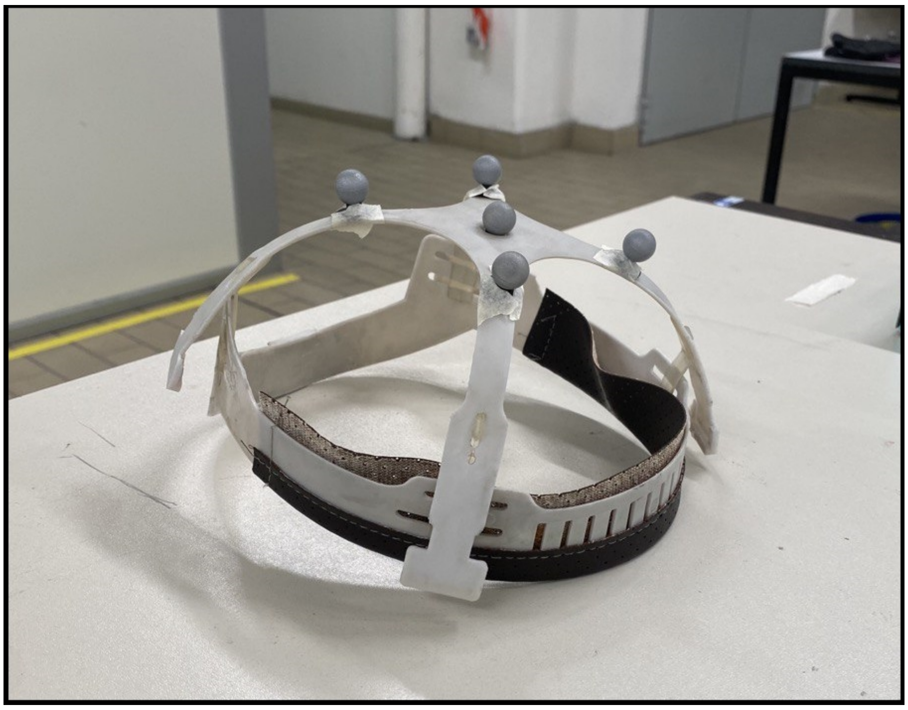
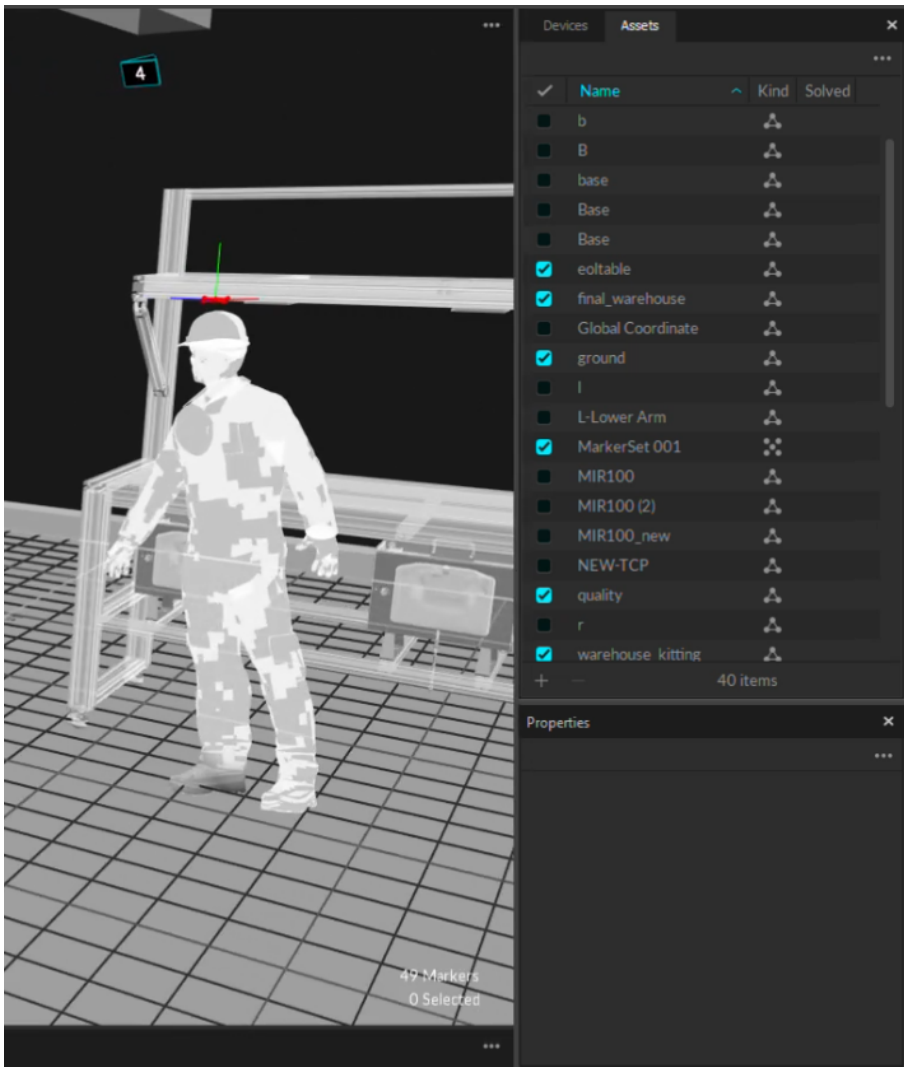


Figure 27: Setup for the Human tracking in real-time with Motive.

This setup is not able to represent physically all parts of a human, but for this project was necessary to just track his movements in the work space, paying no attention to what he does specifically.



✓	Name	Kind	Solved
	b		
	B		
	base		
	Base		
	Base		
✓	eoltable		
✓	final_warehouse		
	Global Coordinate		
✓	ground		
	l		
	L-Lower Arm		
✓	MarkerSet 001		
	MIR100		
	MIR100 (2)		
	MIR100_new		
	NEW-TCP		
✓	quality		
	r		
✓	warehouse_kitting		

40 items

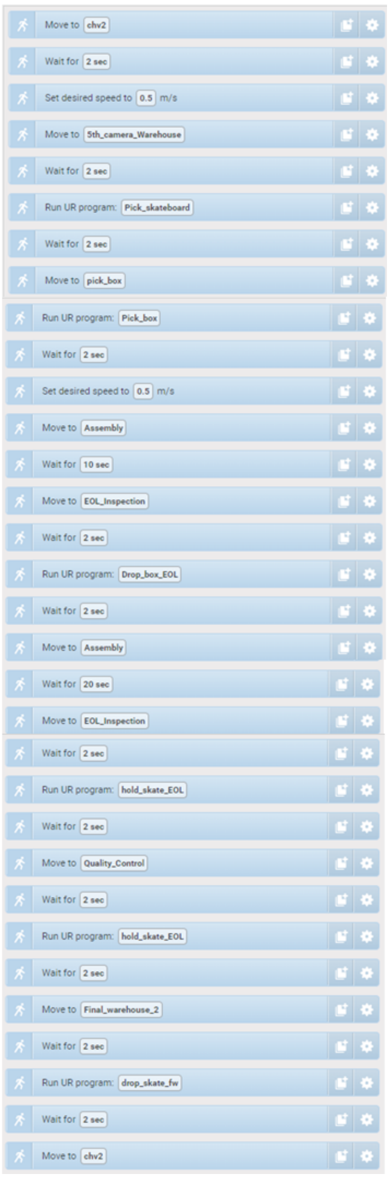
Figure 28: Human rigid body in Motive.

3.4 Tasks

The tasks for this simulation has been developed initially in the MIR software and on the UR3 Teach Pendant, then were replicated on ROS to create the FSM and achieve a more flexible system(Appendix2).

3.4.1 MIR100 tasks

The MIR100 as explained before, has the duty of moving from a station to another, picking components and collaborating with the Operator.
All the tasks are included in a single script, that is also able to run the UR3 scripts done in the Teach Pendant:



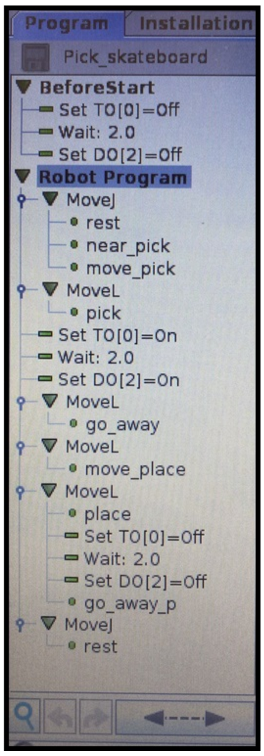
```
Move to chv2
Wait for 2 sec
Set desired speed to 0.5 m/s
Move to 5th_camera_Warehouse
Wait for 2 sec
Run UR program: Pick_skateboard
Wait for 2 sec
Move to pick_box
Run UR program: Pick_box
Wait for 2 sec
Set desired speed to 0.5 m/s
Move to Assembly
Wait for 10 sec
Move to EOL_Inspection
Wait for 2 sec
Run UR program: Drop_box_EOL
Wait for 2 sec
Move to Assembly
Wait for 20 sec
Move to EOL_Inspection
Wait for 2 sec
Run UR program: hold_skate_EOL
Wait for 2 sec
Move to Quality_Control
Wait for 2 sec
Run UR program: hold_skate_EOL
Wait for 2 sec
Move to Final_warehouse_2
Wait for 2 sec
Run UR program: drop_skate_fw
Wait for 2 sec
Move to chv2
```

Figure 29: Script in the MIR100 Software to execute the tasks in the working environment.

3.4.2 UR3 tasks

The UR3 tasks are several:

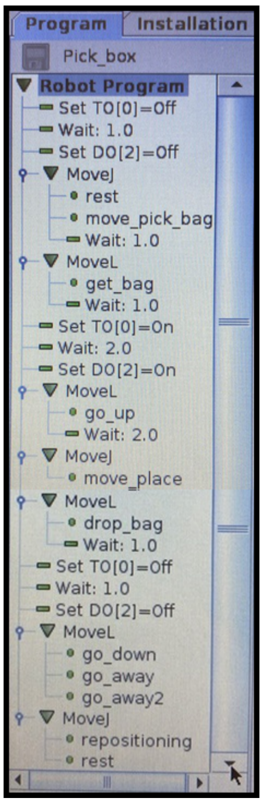
- Pick Skate: The cobot has to pick the skateboard from the Initial Warehouse and place it on his platform.



```
Pick_skateboard
▼ BeforeStart
    Set TO[0]=Off
    Wait: 2.0
    Set DO[2]=Off
▼ Robot Program
    ▼ MoveJ
        rest
        near_pick
        move_pick
    ▼ MoveL
        pick
    Set TO[0]=On
    Wait: 2.0
    Set DO[2]=On
    ▼ MoveL
        go_away
    ▼ MoveL
        move_place
    ▼ MoveL
        place
        Set TO[0]=Off
        Wait: 2.0
        Set DO[2]=Off
        go_away_p
    ▼ MoveJ
        rest
```

Figure 30: UR script in the Teach Pendant to pick the Skateboard.

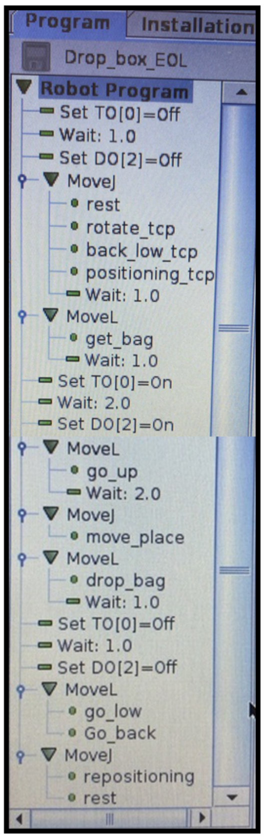
- Pick Box: The cobot has to pick the box with the wheels inside of it from the Initial Warehouse and place it on top of the skateboard.



```
Pick_box
▼ Robot Program
    Set TO[0]=Off
    Wait: 1.0
    Set DO[2]=Off
    ▼ MoveJ
        rest
        move_pick_bag
        Wait: 1.0
    ▼ MoveL
        get_bag
        Wait: 1.0
    Set TO[0]=On
    Wait: 2.0
    Set DO[2]=On
    ▼ MoveL
        go_up
        Wait: 2.0
    ▼ MoveJ
        move_place
    ▼ MoveL
        drop_bag
        Wait: 1.0
    Set TO[0]=Off
    Wait: 1.0
    Set DO[2]=Off
    ▼ MoveL
        go_down
        go_away
        go_away2
    ▼ MoveJ
        repositioning
        rest
```

Figure 31: UR script in the Teach Pendant to pick the Box.

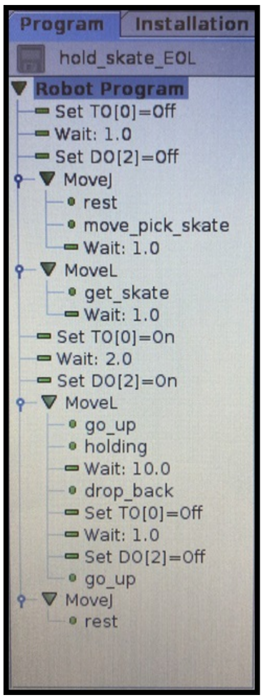
- Drop Box: The cobot has to drop the Box containing some tools in the EOL station.



```
Program | Installation
Drop_box_EOL
▼ Robot Program
   Set TO[0]=Off
   Wait: 1.0
   Set DO[2]=Off
  ▼ MoveJ
      rest
      rotate_tcp
      back_low_tcp
      positioning_tcp
      Wait: 1.0
  ▼ MoveL
      get_bag
      Wait: 1.0
   Set TO[0]=On
   Wait: 2.0
   Set DO[2]=On
  ▼ MoveL
      go_up
      Wait: 2.0
  ▼ MoveJ
      move_place
  ▼ MoveL
      drop_bag
      Wait: 1.0
   Set TO[0]=Off
   Wait: 1.0
   Set DO[2]=Off
  ▼ MoveL
      go_low
      Go_back
  ▼ MoveJ
      repositioning
      rest
```

Figure 32: UR script in the Teach Pendant to drop the Box on the EOL station.

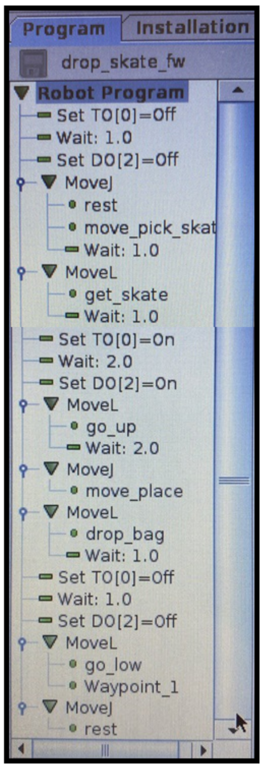
- Hold Skate: This task is repeated two times, in the Quality Control and in the EOL station, and it has the duty to hold the assembled skateboard while the worker is operating on it.



```
Program | Installation
hold_skate_EOL
▼ Robot Program
   Set TO[0]=Off
   Wait: 1.0
   Set DO[2]=Off
  ▼ MoveJ
      rest
      move_pick_skate
      Wait: 1.0
  ▼ MoveL
      get_skate
      Wait: 1.0
   Set TO[0]=On
   Wait: 2.0
   Set DO[2]=On
  ▼ MoveL
      go_up
      holding
      Wait: 10.0
      drop_back
      Set TO[0]=Off
      Wait: 1.0
      Set DO[2]=Off
      go_up
  ▼ MoveJ
      rest
```

Figure 33: UR script in the Teach Pendant to hold the assembled Skateboard.

- Drop Skate: The cobot has to drop the assembled skateboard in the Final Warehouse



```
Program   Installation
drop_skate_fw
▼ Robot Program
   Set TO[0]=Off
   Wait: 1.0
   Set DO[2]=Off
 ▼ MoveJ
     rest
     move_pick_skat
     Wait: 1.0
 ▼ MoveL
     get_skate
     Wait: 1.0
   Set TO[0]=On
   Wait: 2.0
   Set DO[2]=On
 ▼ MoveL
     go_up
     Wait: 2.0
 ▼ MoveJ
     move_place
 ▼ MoveL
     drop_bag
     Wait: 1.0
   Set TO[0]=Off
   Wait: 1.0
   Set DO[2]=Off
 ▼ MoveL
     go_low
     Waypoint_1
 ▼ MoveJ
     rest
```

Figure 34: UR script in the Teach Pendant to drop the assembled skateboard on the Final Warehouse.

3.5 Errors in the line of production

In addition to the normal course of the production line, non-standard situations were taken into account:

- Skateboard miss: The robot may miss the skateboard, in this case it has to retry.
- Box miss: The robot may miss the box, in this case it has to retry.
- Non-compliance during control at the EOL station: While the operator is checking the assembled piece, may notice some minor problems, for example if a piece is not firm, so the MOMA has to go back to the Assembly station to let the operator adjust the error.
- Quality errors: In this section there are more accurate controls, checking if the piece complies with the standards. If this is not the case, the MOMA can not continue his task because the piece is not suitable for sale.
- Obstacles on the way: Objects or people may obstruct the road where the robot operates, so the MOMA has to find always another way or waiting for the passage to be free without getting stuck.
- Emergency stop: If the operator needs to press the emergency stop button several times during the work the MOMA must stop and start again every time without getting stuck.

The introduction of different types of errors shows the limitation of the basic software provided by the robot manufacturers, because they are not able to handle it.
To this end, the system will be transferred to ROS, which will give it greater adaptability and elasticity.

4 Computer-MIR100-UR3 Network

The network created in this project is composed by three elements, The computer (with ROS installed inside it in a Virtual Machine), The MIR100 and the UR3, all connected via Ethernet by a Tp-link modem, that permits to create a LAN between them.

4.1 ROS configuration in Computer

First of all, in order to configure ROS, a Virtual Machine has been installed into the computer, since the OS installed in it was Windows 10, and ROS is still ”experimental” on Windows and MacOS, instead Linux is the only Operating system listed as ”supported”, so the project has been created on it [30]. The installed Virtual Machine is ’Oracle VM VirtualBox’ v.7.0.10.
The settings of the Virtual Machine that has been created are:

- Hard Disk memory: 10 Gb;
- Processor: 5 Cores;
- Network card: Board with bridge (Realtek PCIe GbE Family Controller);
- ISO image: ’ubuntu-20.04.6-desktop-amd64’.

Once the Virtual Machine has been created, ROS can be installed in it, specifically, ROS Noetic 20.04 [4][26].

4.2 MIR100-Computer

Once ROS is correctly installed in the computer, the MIR100 packages can be installed in order to connect the Mobile Robot to all the necessary apps, in particular, the MIR100 has to be able to comunicate with:

- ROS;
- MATLAB;
- Simulink.

4.2.1 MIR100-ROS

The connection between the MIR100 and ROS is the most important one, because permits, thanks to the packages that this link creates, the connection with all the other apps.
First of all, has to be installed the repository that contains a ROS driver and ROS configuration files for the MIR robots[5], and it can be used for different MIR models (MIR100, MIR200, MIR250, MIR500).
This repository hasn’t been created by the company ”Mobile Industrial Robots”, so it contains different type of packages that are not the totality of the packages available in the phisical MIR robots.
The packages contained in this repository are[7]:

- mir_actions: Action definitions for the MiR robot;
- mir_description: URDF description of the MiR robot;
- mir_dbw_critics: Plugins for the dwb_local_planner used in Gazebo;
- mir_driver: A reverse ROS bridge for the MiR robot
- mir_gazebo: Simulation specific launch and configuration files for the MiR robot
- mir_msgs: Message definitions for the MiR robot
- mir_navigation: move_base launch and configuration files

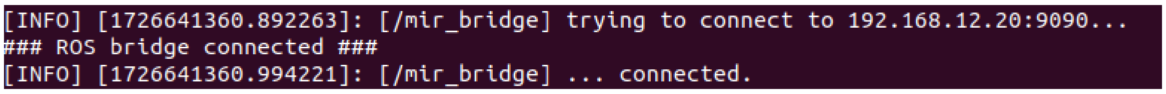
In this project the installation type was the recommended one, so the ’binary install’. With this installation the source is not modified, it is recommended for this reason.
Once the installation of the repository is completed, the next step is connecting to the MIR Wi-Fi, then, is possible to link all the obtained files containing the MIR’s information to ROS.

[4]ROS Noetic installation site: https://wiki.ros.org/noetic/Installation/Ubuntu
[5]MIR100 driver installation site: https://github.com/DFKI-NI/mir_robot

To do so, in ROS terminal has to be launched the 'mir.launch' script as follows:

$$\text{"}roslaunch \quad mir\_driver \quad mir.launch\text{"} \tag{1}$$

If everything has been installed correctly and the computer is connected properly, the bridge between MIR100 and ROS will be established, so the connection will be set as shown in Fig.(35).

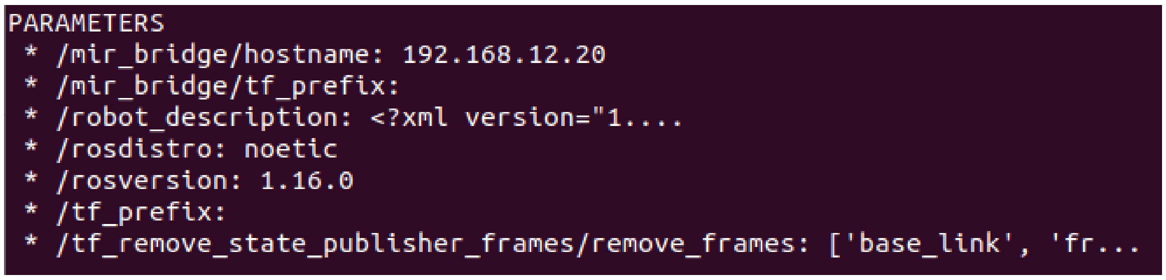


```
[INFO] [1726641360.892263]: [/mir_bridge] trying to connect to 192.168.12.20:9090...
### ROS bridge connected ###
[INFO] [1726641360.994221]: [/mir_bridge] ... connected.
```

Figure 35: Connection set between the MIR100 and ROS established.

In the terminal window, it is possible to see several information about the MIR100:

- PARAMETERS: The parameters shows some basic information about the robot and the operating system as shown below (Fig.36):

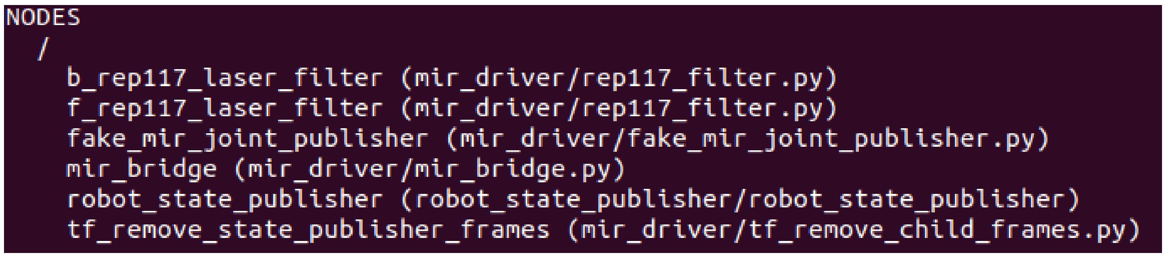


```
PARAMETERS
 * /mir_bridge/hostname: 192.168.12.20
 * /mir_bridge/tf_prefix:
 * /robot_description: <?xml version="1....
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /tf_prefix:
 * /tf_remove_state_publisher_frames/remove_frames: ['base_link', 'fr...
```

Figure 36: Parameters generated by the connection between the MIR100 and ROS.

The most useful information for this project is the 'hostname', because it will be a vital data to establish the network between all the devices.

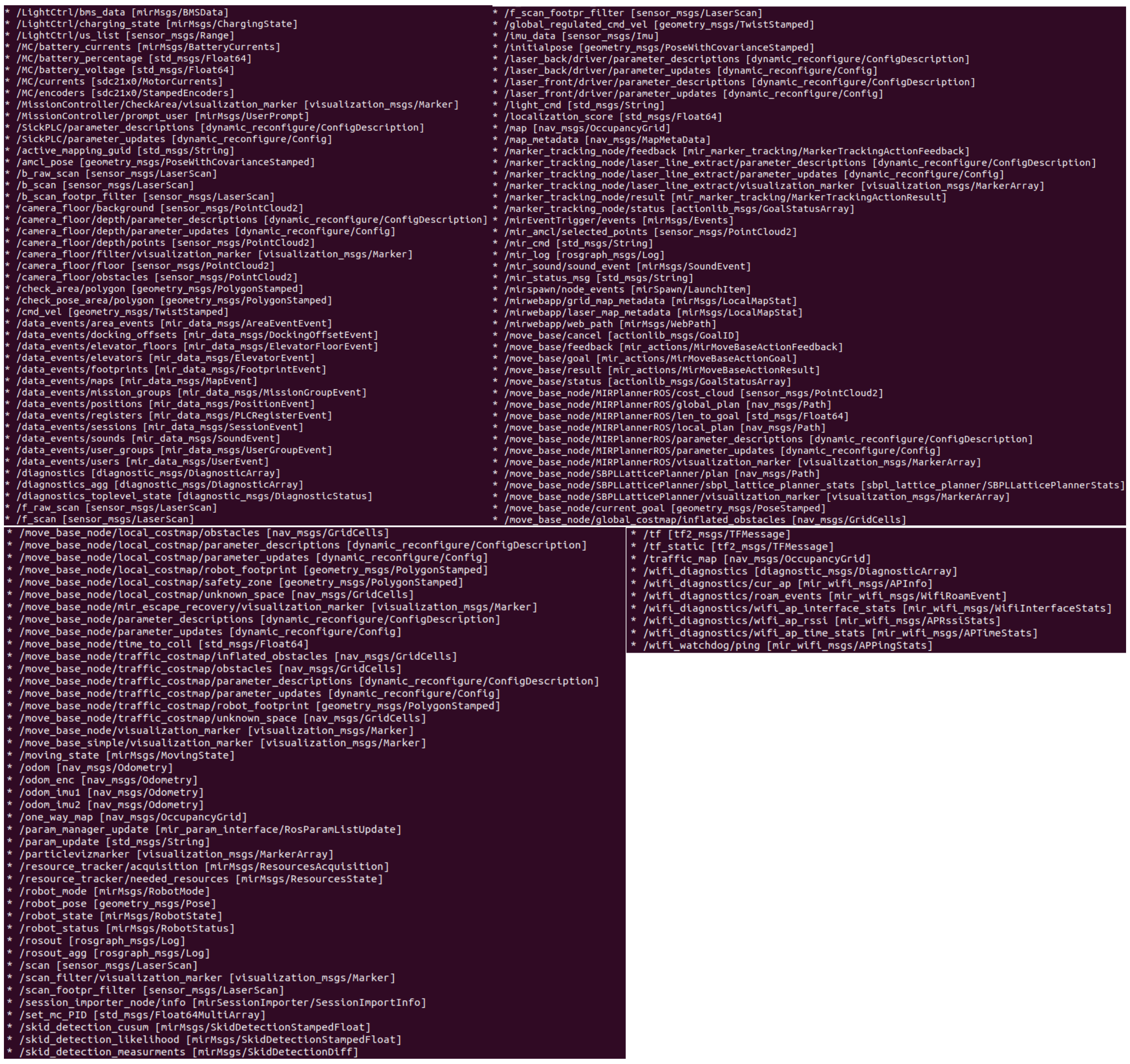
- NODES: The nodes contain all the processes necessary for the MIR100 to work properly (Fig.(37)):



```
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)
```

Figure 37: Nodes generated by the connection between the MIR100 and ROS.

There is a node that, for this project, need more attention, that is the 'mir_bridge' node, which contains several useful topics as the position of the robot or the map where it operates, that are crucial for the execution of the tasks.
The topics contained in the nodes are not all available to the user, because the repository containing the ROS driver and the ROS configuration files for the MIR robots comes from a private as mentioned before, so some packages have not been integrated.
In the 'mir.launch' terminal window, are shown all the MIR100 topics and the messages types related to them (Fig.(38)):



```
* /LightCtrl/bms_data [mirMsgs/BMSData]
* /LightCtrl/charging_state [mirMsgs/ChargingState]
* /LightCtrl/us_list [sensor_msgs/Range]
* /MC/battery_currents [mirMsgs/BatteryCurrents]
* /MC/battery_percentage [std_msgs/Float64]
* /MC/battery_voltage [std_msgs/Float64]
* /MC/currents [sdc21x0/MotorCurrents]
* /MC/encoders [sdc21x0/StampedEncoders]
* /MissionController/CheckArea/visualization_marker [visualization_msgs/Marker]
* /MissionController/prompt_user [mirMsgs/UserPrompt]
* /SickPLC/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /SickPLC/parameter_updates [dynamic_reconfigure/Config]
* /active_mapping_guid [std_msgs/String]
* /amcl_pose [geometry_msgs/PoseWithCovarianceStamped]
* /b_raw_scan [sensor_msgs/LaserScan]
* /b_scan [sensor_msgs/LaserScan]
* /b_scan_footpr_filter [sensor_msgs/LaserScan]
* /camera_floor/background [sensor_msgs/PointCloud2]
* /camera_floor/depth/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /camera_floor/depth/parameter_updates [dynamic_reconfigure/Config]
* /camera_floor/depth/points [sensor_msgs/PointCloud2]
* /camera_floor/filter/visualization_marker [visualization_msgs/Marker]
* /camera_floor/floor [sensor_msgs/PointCloud2]
* /camera_floor/obstacles [sensor_msgs/PointCloud2]
* /check_area/polygon [geometry_msgs/PolygonStamped]
* /check_pose_area/polygon [geometry_msgs/PolygonStamped]
* /cmd_vel [geometry_msgs/TwistStamped]
* /data_events/area_events [mir_data_msgs/AreaEventEvent]
* /data_events/docking_offsets [mir_data_msgs/DockingOffsetEvent]
* /data_events/elevator_floors [mir_data_msgs/ElevatorFloorEvent]
* /data_events/elevators [mir_data_msgs/ElevatorEvent]
* /data_events/footprints [mir_data_msgs/FootprintEvent]
* /data_events/maps [mir_data_msgs/MapEvent]
* /data_events/mission_groups [mir_data_msgs/MissionGroupEvent]
* /data_events/positions [mir_data_msgs/PositionEvent]
* /data_events/registers [mir_data_msgs/PLCRegisterEvent]
* /data_events/sessions [mir_data_msgs/SessionEvent]
* /data_events/sounds [mir_data_msgs/SoundEvent]
* /data_events/user_groups [mir_data_msgs/UserGroupEvent]
* /data_events/users [mir_data_msgs/UserEvent]
* /diagnostics [diagnostic_msgs/DiagnosticArray]
* /diagnostics_agg [diagnostic_msgs/DiagnosticArray]
* /diagnostics_toplevel_state [diagnostic_msgs/DiagnosticStatus]
* /f_raw_scan [sensor_msgs/LaserScan]
* /f_scan [sensor_msgs/LaserScan]
```

```
* /f_scan_footpr_filter [sensor_msgs/LaserScan]
* /global_regulated_cmd_vel [geometry_msgs/TwistStamped]
* /imu_data [sensor_msgs/Imu]
* /initialpose [geometry_msgs/PoseWithCovarianceStamped]
* /laser_back/driver/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /laser_back/driver/parameter_updates [dynamic_reconfigure/Config]
* /laser_front/driver/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /laser_front/driver/parameter_updates [dynamic_reconfigure/Config]
* /light_cmd [std_msgs/String]
* /localization_score [std_msgs/Float64]
* /map [nav_msgs/OccupancyGrid]
* /map_metadata [nav_msgs/MapMetaData]
* /marker_tracking_node/feedback [mir_marker_tracking/MarkerTrackingActionFeedback]
* /marker_tracking_node/laser_line_extract/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /marker_tracking_node/laser_line_extract/parameter_updates [dynamic_reconfigure/Config]
* /marker_tracking_node/laser_line_extract/visualization_marker [visualization_msgs/MarkerArray]
* /marker_tracking_node/result [mir_marker_tracking/MarkerTrackingActionResult]
* /marker_tracking_node/status [actionlib_msgs/GoalStatusArray]
* /mirEventTrigger/events [mirMsgs/Events]
* /mir_amcl/selected_points [sensor_msgs/PointCloud2]
* /mir_cmd [std_msgs/String]
* /mir_log [rosgraph_msgs/Log]
* /mir_sound/sound_event [mirMsgs/SoundEvent]
* /mir_status_msg [std_msgs/String]
* /mirspawn/node_events [mirSpawn/LaunchItem]
* /mirwebapp/grid_map_metadata [mirMsgs/LocalMapStat]
* /mirwebapp/laser_map_metadata [mirMsgs/LocalMapStat]
* /mirwebapp/web_path [mirMsgs/WebPath]
* /move_base/cancel [actionlib_msgs/GoalID]
* /move_base/feedback [mir_actions/MirMoveBaseActionFeedback]
* /move_base/goal [mir_actions/MirMoveBaseActionGoal]
* /move_base/result [mir_actions/MirMoveBaseActionResult]
* /move_base/status [actionlib_msgs/GoalStatusArray]
* /move_base_node/MIRPlannerROS/cost_cloud [sensor_msgs/PointCloud2]
* /move_base_node/MIRPlannerROS/global_plan [nav_msgs/Path]
* /move_base_node/MIRPlannerROS/len_to_goal [std_msgs/Float64]
* /move_base_node/MIRPlannerROS/local_plan [nav_msgs/Path]
* /move_base_node/MIRPlannerROS/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /move_base_node/MIRPlannerROS/parameter_updates [dynamic_reconfigure/Config]
* /move_base_node/MIRPlannerROS/visualization_marker [visualization_msgs/MarkerArray]
* /move_base_node/SBPLLatticePlanner/plan [nav_msgs/Path]
* /move_base_node/SBPLLatticePlanner/sbpl_lattice_planner_stats [sbpl_lattice_planner/SBPLLatticePlannerStats]
* /move_base_node/SBPLLatticePlanner/visualization_marker [visualization_msgs/MarkerArray]
* /move_base_node/current_goal [geometry_msgs/PoseStamped]
* /move_base_node/global_costmap/inflated_obstacles [nav_msgs/GridCells]
```

```
* /move_base_node/local_costmap/obstacles [nav_msgs/GridCells]
* /move_base_node/local_costmap/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /move_base_node/local_costmap/parameter_updates [dynamic_reconfigure/Config]
* /move_base_node/local_costmap/robot_footprint [geometry_msgs/PolygonStamped]
* /move_base_node/local_costmap/safety_zone [geometry_msgs/PolygonStamped]
* /move_base_node/local_costmap/unknown_space [nav_msgs/GridCells]
* /move_base_node/mir_escape_recovery/visualization_marker [visualization_msgs/Marker]
* /move_base_node/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /move_base_node/parameter_updates [dynamic_reconfigure/Config]
* /move_base_node/time_to_coll [std_msgs/Float64]
* /move_base_node/traffic_costmap/inflated_obstacles [nav_msgs/GridCells]
* /move_base_node/traffic_costmap/obstacles [nav_msgs/GridCells]
* /move_base_node/traffic_costmap/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
* /move_base_node/traffic_costmap/parameter_updates [dynamic_reconfigure/Config]
* /move_base_node/traffic_costmap/robot_footprint [geometry_msgs/PolygonStamped]
* /move_base_node/traffic_costmap/unknown_space [nav_msgs/GridCells]
* /move_base_node/visualization_marker [visualization_msgs/Marker]
* /move_base_simple/visualization_marker [visualization_msgs/Marker]
* /moving_state [mirMsgs/MovingState]
* /odom [nav_msgs/Odometry]
* /odom_enc [nav_msgs/Odometry]
* /odom_imu1 [nav_msgs/Odometry]
* /odom_imu2 [nav_msgs/Odometry]
* /one_way_map [nav_msgs/OccupancyGrid]
* /param_manager_update [mir_param_interface/RosParamListUpdate]
* /param_update [std_msgs/String]
* /particlevizmarker [visualization_msgs/MarkerArray]
* /resource_tracker/acquisition [mirMsgs/ResourcesAcquisition]
* /resource_tracker/needed_resources [mirMsgs/ResourcesState]
* /robot_mode [mirMsgs/RobotMode]
* /robot_pose [geometry_msgs/Pose]
* /robot_state [mirMsgs/RobotState]
* /robot_status [mirMsgs/RobotStatus]
* /rosout [rosgraph_msgs/Log]
* /rosout_agg [rosgraph_msgs/Log]
* /scan [sensor_msgs/LaserScan]
* /scan_filter/visualization_marker [visualization_msgs/Marker]
* /scan_footpr_filter [sensor_msgs/LaserScan]
* /session_importer_node/info [mirSessionImporter/SessionImportInfo]
* /set_mc_PID [std_msgs/Float64MultiArray]
* /skid_detection_cusum [mirMsgs/SkidDetectionStampedFloat]
* /skid_detection_likelihood [mirMsgs/SkidDetectionStampedFloat]
* /skid_detection_measurments [mirMsgs/SkidDetectionDiff]
```

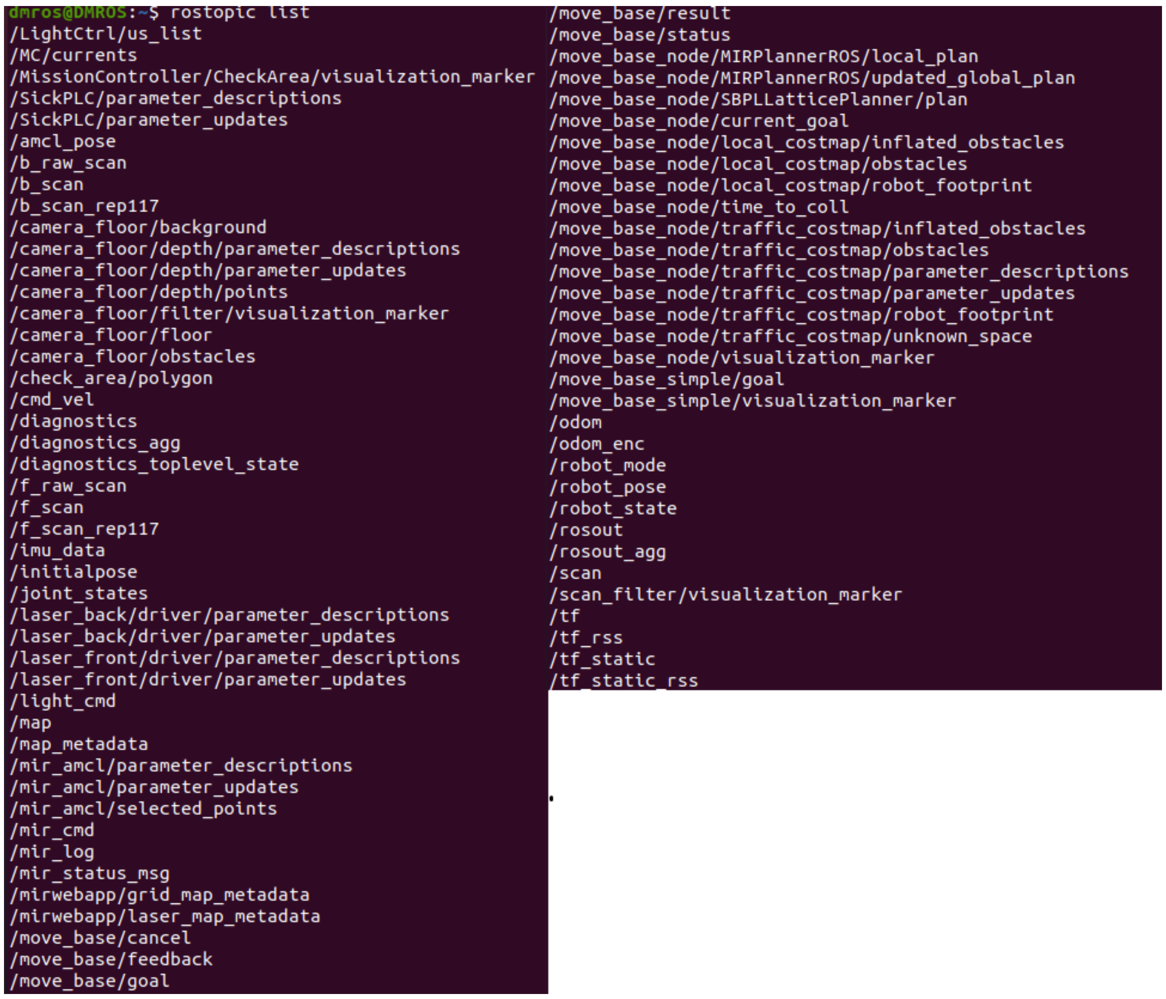
```
* /tf [tf2_msgs/TFMessage]
* /tf_static [tf2_msgs/TFMessage]
* /traffic_map [nav_msgs/OccupancyGrid]
* /wifi_diagnostics [diagnostic_msgs/DiagnosticArray]
* /wifi_diagnostics/cur_ap [mir_wifi_msgs/APInfo]
* /wifi_diagnostics/roam_events [mir_wifi_msgs/WifiRoamEvent]
* /wifi_diagnostics/wifi_ap_interface_stats [mir_wifi_msgs/WifiInterfaceStats]
* /wifi_diagnostics/wifi_ap_rssi [mir_wifi_msgs/APRssiStats]
* /wifi_diagnostics/wifi_ap_time_stats [mir_wifi_msgs/APTimeStats]
* /wifi_watchdog/ping [mir_wifi_msgs/APPingStats]
```

Figure 38: Totality of the topics available in the MIR100

but to see the real available topics is necessary to open a new terminal window and typing the command :

$$rostopic\,list \tag{2}$$

The result, in Fig.(39), will manifest on screen a different amount of topics than what is shown in the previous figure.

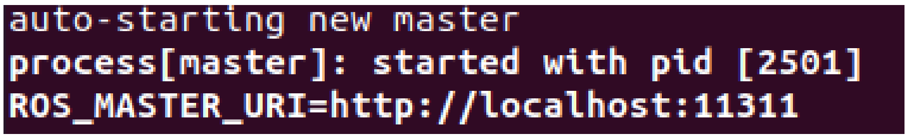


```
dmros@DMROS:~$ rostopic list
/LightCtrl/us_list
/MC/currents
/MissionController/CheckArea/visualization_marker
/SickPLC/parameter_descriptions
/SickPLC/parameter_updates
/amcl_pose
/b_raw_scan
/b_scan
/b_scan_rep117
/camera_floor/background
/camera_floor/depth/parameter_descriptions
/camera_floor/depth/parameter_updates
/camera_floor/depth/points
/camera_floor/filter/visualization_marker
/camera_floor/floor
/camera_floor/obstacles
/check_area/polygon
/cmd_vel
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/f_raw_scan
/f_scan
/f_scan_rep117
/imu_data
/initialpose
/joint_states
/laser_back/driver/parameter_descriptions
/laser_back/driver/parameter_updates
/laser_front/driver/parameter_descriptions
/laser_front/driver/parameter_updates
/light_cmd
/map
/map_metadata
/mir_amcl/parameter_descriptions
/mir_amcl/parameter_updates
/mir_amcl/selected_points
/mir_cmd
/mir_log
/mir_status_msg
/mirwebapp/grid_map_metadata
/mirwebapp/laser_map_metadata
/move_base/cancel
/move_base/feedback
/move_base/goal
/move_base/result
/move_base/status
/move_base_node/MIRPlannerROS/local_plan
/move_base_node/MIRPlannerROS/updated_global_plan
/move_base_node/SBPLLatticePlanner/plan
/move_base_node/current_goal
/move_base_node/local_costmap/inflated_obstacles
/move_base_node/local_costmap/obstacles
/move_base_node/local_costmap/robot_footprint
/move_base_node/time_to_coll
/move_base_node/traffic_costmap/inflated_obstacles
/move_base_node/traffic_costmap/obstacles
/move_base_node/traffic_costmap/parameter_descriptions
/move_base_node/traffic_costmap/parameter_updates
/move_base_node/traffic_costmap/robot_footprint
/move_base_node/traffic_costmap/unknown_space
/move_base_node/visualization_marker
/move_base_simple/goal
/move_base_simple/visualization_marker
/odom
/odom_enc
/robot_mode
/robot_pose
/robot_state
/rosout
/rosout_agg
/scan
/scan_filter/visualization_marker
/tf
/tf_rss
/tf_static
/tf_static_rss
```

Figure 39: Effective topics available for the user

It is clear that the available topics are far fewer than those shown.

- ROS Master: Running the 'mir.launch' script, all the MIR nodes will connect, if not otherwise specified, to the ROS Master, whose address will be shown in the terminal window as shown below in Fig.(40):



```
auto-starting new master
process[master]: started with pid [2501]
ROS_MASTER_URI=http://localhost:11311
```

Figure 40: ROS Master URI IP address

Typically, as in this case, if all the ROS nodes are executed on a single machine, the IP of the ROS Master will be set to:

$$http://localhost:11311 \tag{3}$$

Where

$$localhost = 127.0.0.1 \tag{4}$$

The last number shown in the IP, '11311', is the communication port where the ROS Master communicates with the nodes.
The choice of this specific port is a convention of the ROS Team.
The choice of ROS as the master has been taken because, with this settings, is possible to monitor and control from the ROS terminal the available topics smoothly. Another possible choice would have been the use of the MIR100 as the master, so we would have seen as 'ROS_MASTER_URI:

$$http://192.168.12.20:11311 \tag{5}$$

This IP comes from the MIR100 hostname shown in the Parameters.
The problem with this choice is that it is not possible to use and monitor several topics from the ROS terminal, because the ROS nodes aren't communicating with the proper Master. Has been noticed localization problems while trying to command the Mobile Robot setting the MIR100 as master, so this choice has been aborted.

In Fig.(41) is possible to visualize, thanks to the graphical tool 'rqt_graph', the communication structure of the Master and Nodes. It can be seen the exchange of messages through active topics and services.

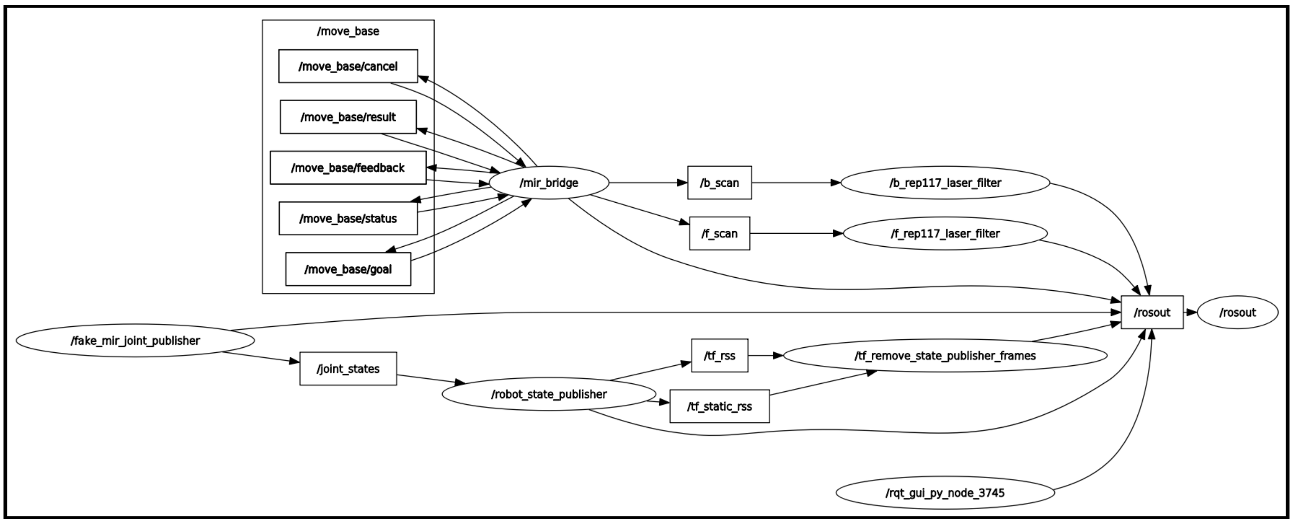


Figure 41: RQT Graph of the connection between the MIR100 and the ROS Master.

Thanks to this configuration is possible to control the MIR100 directly from the terminal using the available topics.

4.2.2 MIR100-MATLAB

It is possible to manipulate the topics not only from the ROS terminal, but also from MATLAB.
The first thing that has been done is downloading MATLAB directly inside the Virtual Machine, in order to avoid an extra layer of communication that there would have been if the application was downloaded on Windows.
To establish this connection, there is a dedicated command in MATLAB: 'rosinit'.
Running the 'rosinit' command in MATLAB, permits to initialize the connection between MATLAB and a ROS system. This command connects MATLAB to the ROS master so that it can function as a ROS node and communicate with the other components of the robotic system when utilizing MATLAB to interface with ROS[11].
This command can be defined in different ways:

- rosinit: starts the global ROS node with a default name and tries to connect to a ROS master running on localhost and port 11311. If the global ROS node cannot connect to the ROS master, rosinit also starts a ROS core in MATLAB, which consists of a ROS master, a ROS parameter server, and a rosout logging node;
- rosinit(hostname): tries to connect to the ROS master at the host name or IP address specified by hostname. The communication port used is the default one;
- rosinit(hostname,port): tries to connect to the ROS master at the host name or IP address specified by hostname. The communication port used is no longer the default one but is specified by the user;
- rosinit(URI): the ROS_MASTER_URI to which MATLAB attempts to connect is specified by the user.

In this case the used command is the following:

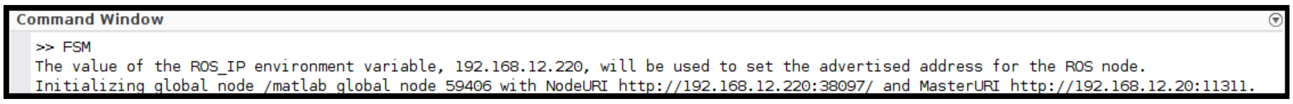
$$rosinit("http://192.168.12.20:11311") \tag{6}$$

The ROS_MASTER_URI specified corresponds to the MIR100 master.
This decision was taken because using the MIR100 as master, gives to the user a wider range usable topics,

some of them are needed for the project, but as mentioned before, it wasn't possible to establish the connection between MIR100 and ROS using the MIR100 as the master, therefore has been adopted this solution to be able to subscribe the needed topics. The idea is to get the data in MATLAB from those topics and manipulate them in ROS.
Another command has been used to complete the communication system:

$$setenv('ROS\_IP', '192.168.12.220') \tag{7}$$

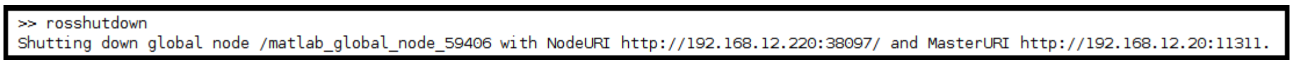
This command permits to set environment variables. Environment variables are values that are used to configure different aspects of an operating system's and running program's operation[12].
Setting this variable is important for the communication between MATLAB and the MIR100, because the Master must be able to recognise the IP address of the machine on which MATLAB is running, otherwise ROS Master may not be able to correctly return messages to MATLAB nodes.
If everything is set up correctly, after running those command, the following messages should appear on the command window as in fig.(42):



```
Command Window
  >> FSM
  The value of the ROS_IP environment variable, 192.168.12.220, will be used to set the advertised address for the ROS node.
  Initializing global node /matlab_global_node_59406 with NodeURI http://192.168.12.220:38097/ and MasterURI http://192.168.12.20:11311.
```

Figure 42: Environment variable set and connection to the MIR Master.

After establishing the connection between the MIR100 and MATLAB, can be noticed that, running the 'rqt_graph' command, this connection is not present in the system, because the MATLAB node isn't connected to the ROS Master that creates the graph but to another one. This is another reason why it is better to avoid the connection to an external ROS Master, because if some problems occur, not being able to see from the Computer the whole communication system, the debugging is more difficult.
The MATLAB node becomes present only after running the Simulink model, because they start communicating (it can be seen denoted as '/matlab_introspect' in Fig.(46)).
To terminate the connection, simply enter the command: 'rosshutdown' (fig(43)).



```
>> rosshutdown
Shutting down global node /matlab_global_node_59406 with NodeURI http://192.168.12.220:38097/ and MasterURI http://192.168.12.20:11311.
```

Figure 43: ROS Master shutted down.

4.2.3 MIR100-Simulink

Simulink is the software where the Finite State Machine is built, so is essential to establish a connection between it and the MIR100.
Simulink has to read data and send commands to ROS through the topics, so ROS will transfer the information to the physical robot that will execute them if needed.
This software is integrated in MATLAB, so it is installed in the ROS environment. It can communicate with ROS thanks to the 'ROS Toolbox'.
The most important blocks provided by this toolbox are:

- Subscribe block: Used to receive a message from the ROS Network;

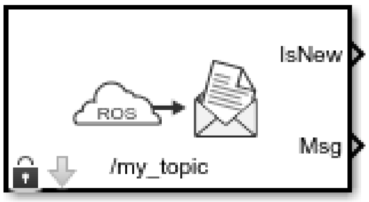


Figure 44: Subscribe block provided by ROS Toolbox.

- Publish block: Used to send a message to the ROS Network;

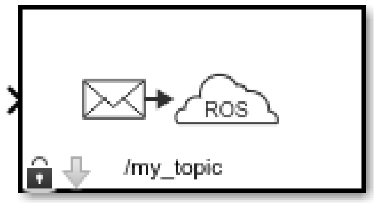


Figure 45: Publish block provided by ROS Toolbox.

The topics available to publish and subscribe are the ones provided by the connection between the MIR100 and ROS, having the Robot Operating System as Master.
This happens because, even if MATLAB node is connected to the MIR Master, we still have the ROS Master created setting the mir bridge, because the MIR100 can act as node and as Master, and the ROS Toolbox establish a connection between the ROS Master and Simulink, so MATLAB and Simulink are talking to different Masters, that means that have different available topics.
This fact can be noticed looking at the rqt graph:

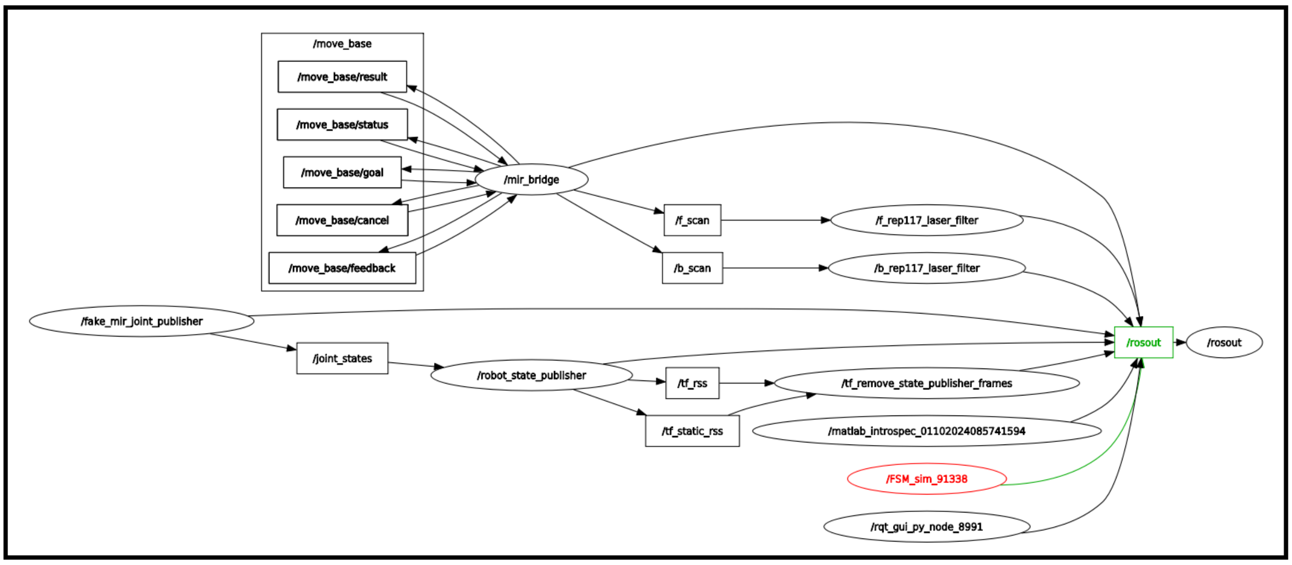


Figure 46: Rqt graph of the active Simulink node.

As mentioned before, with the Simulink nodes it creates also the MATLAB one.
The advantage of having MATLAB and Simulink talking with different Masters is that, even if we don't have all the available packages and topics in ROS, we can get more of them from MATLAB, that can easily pass all the obtained data to Simulink.
Due to the lack of packages, the movements of the MIR100 won't be controlled by the publish block in Simulink, instead, from the 'MATLAB function' block, will be called a MATLAB function that will call a bash script defined in ROS that will execute the movements of the MIR100 from a station to another.

4.3 UR3-Computer

Once ROS is correctly installed in the computer, the UR3 packages can be installed[6] in order to connect the Universal Robot UR3 to all the necessary apps, in particular, it has to be able to communicate with:

- ROS;
- MATLAB;
- Simulink.

[6]UR3 driver installation site: https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

4.3.1 UR3-ROS

The connection between the UR3 and the computer is the backbone of the system, because contains all the packages available to control and monitor the Cobot, and because the most of the work has been done in the ROS platform instead of the other apps.
To start with, the repository to download contains all the packages in order to have a stable and sustainable interface between UR robots and ROS[7].
The main packages contained in the repository are:

- ur_calibration: this package contains the robot's factory calibration information. Those information are extracted and converted by the package so that in ROS the joints coordinates provided corresponds to the real ones;
- ur_dashboard_msgs: Message definitions used for interacting with the dashboard node;
- ur_robot_driver: This is the most important repository, because contains all the commands usefull to actually move the UR, so it contains the actual driver of the robot.
  in Fig.(47)is shown an approximate scheme of the driver's architecture:

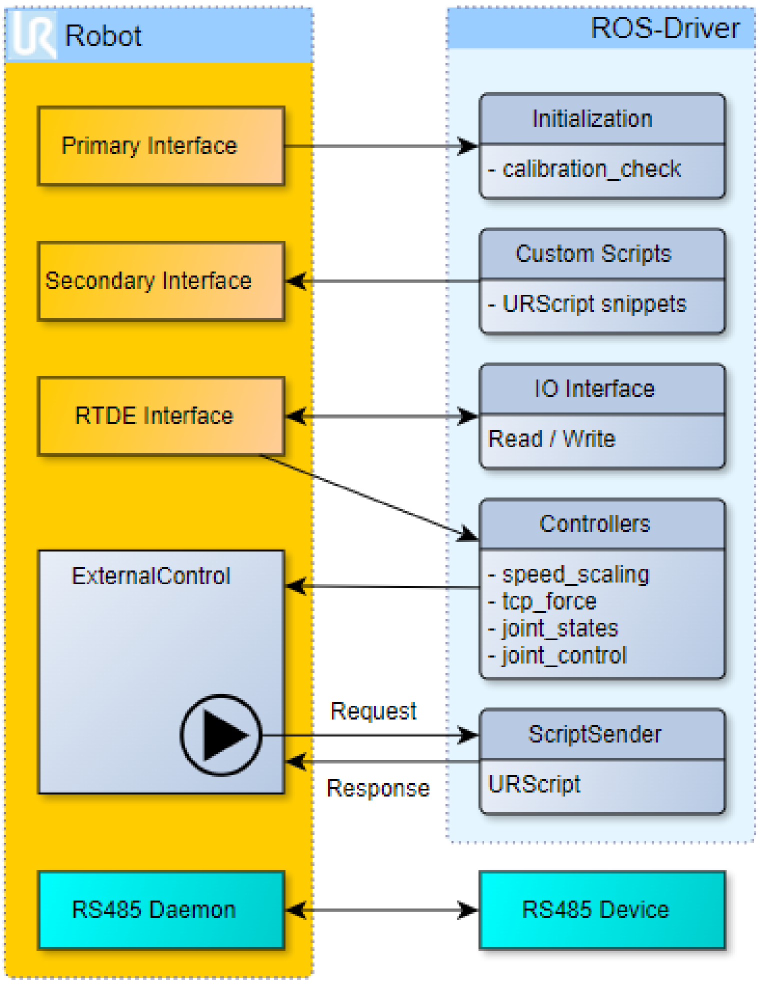
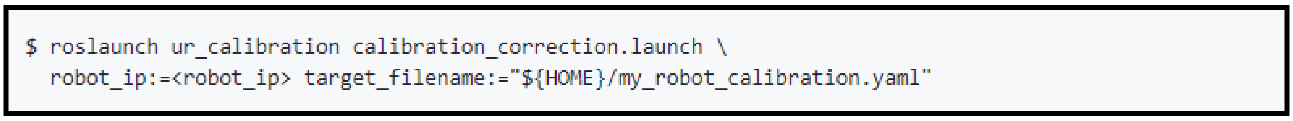


Figure 47: Superficial overview of the driver's architecture[5] .

The UR driver has been installed from source.
Once the installation is completed, the first thing needed is the calibration of the robot. The command to execute in the terminal to do is shown in fig.(48):

[7]UR3 driver installation site: https://github.com/UniversalRobots/Universal Robots ROS Driver



```
$ roslaunch ur_calibration calibration_correction.launch \
  robot_ip:=<robot_ip> target_filename:="${HOME}/my_robot_calibration.yaml"
```

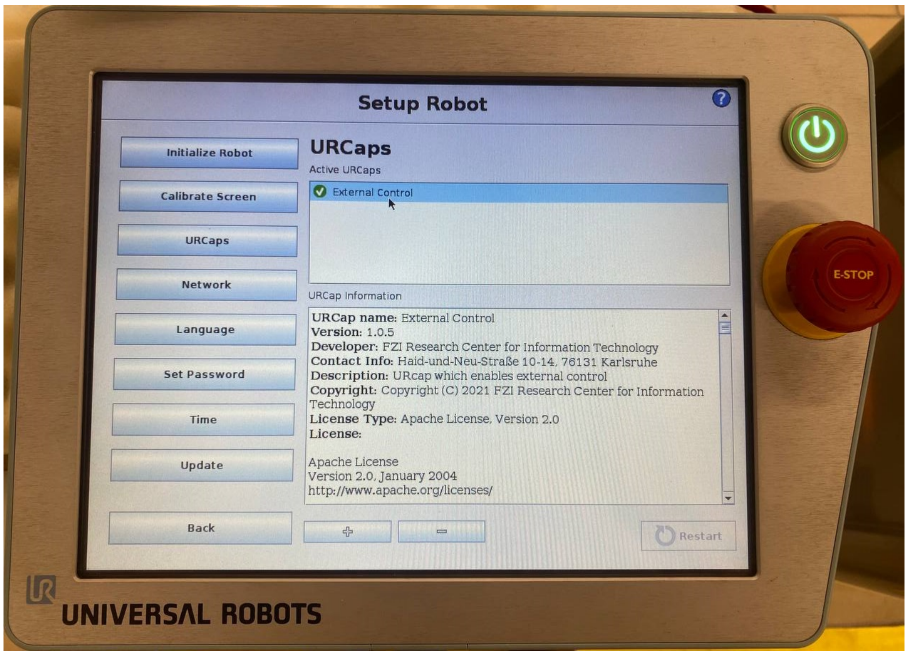
Figure 48: ROS command to make the calibration of the UR robot.

For this project, it will be selected the 'ur3_calibration' file, and the robot IP it will be varying depending on if the UR3 is used physically or remotely (offline simulation).
Moving on, has to be installed in the UR3 Teach Pendant the URcap. A URCap is a Java-based plugin, that integrates in to PolyScope [22], usefull to extend its functionality, by adding new programming screens and new specific functionalities.
Below are listed the key features of URcaps [19]:

- New Installation Nodes: Set up application contexts and general settings for end effectors, conveyors, and coordinate systems;
- Program Nodes: Carry out predetermined actions in a predetermined order, like repositioning the robot, shutting a gripper, and shifting the object it has grabbed;
- Daemon Processes: executables in the background that increase PolyScope's functionality;
- Program Templates: Workflows can be efficiently created by combining URCap nodes with integrated program nodes;
- New Variables and Scripting Functions: more personalization and control.

The specific URcap used for this project is provided by MathWorks, and is called ' matlab_externalcontrol-1.0.0.urcap'[8].
First thing to do is downloading the MATLAB Toolbox ' Robotics System Toolbox™ Support Package for Universal Robots UR Series Manipulators' which serves to establish the communication between ROS and a UR manipulator.
Once the comunication between the two is set, the MathWorks's URcap is needed to have an external control of the manipulator on ROS. The URcap has been set as follows:

- The URcap has been downloaded from GitHub[15] into a USB flash drive;
- Plug the USB flash drive in the UR3 Teach Pendant;
- Setup Robot → URcaps → + → add the 'matlab_externalcontrol-1.0.0.urcap' file.
  If everything has been done correctly, the loaded URcap should appear in the 'Active URcaps' section:



URCap name: External Control
Version: 1.0.5
Developer: FZI Research Center for Information Technology
Contact Info: Haid-und-Neu-Straße 10-14, 76131 Karlsruhe
Description: URcap which enables external control
Copyright: Copyright (C) 2021 FZI Research Center for Information Technology
License Type: Apache License, Version 2.0
License:

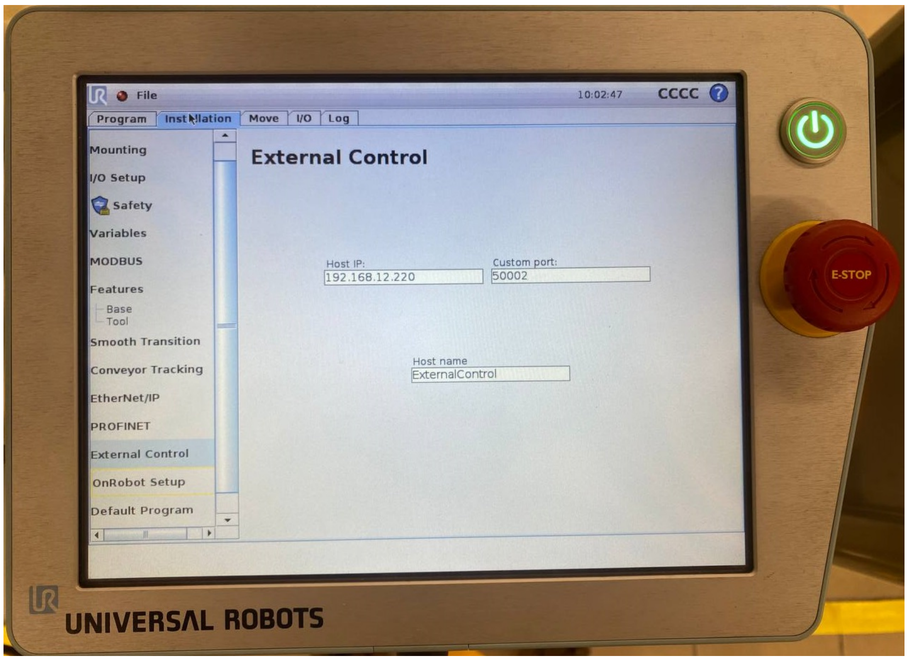
Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

Figure 49: 'matlab_externalcontrol-1.0.0.urcap' loaded in the UR3 teach pendant.

[8]URcap installation site : https://it.mathworks.com/help/robotics/urseries/ug/install-urcap.html

Press the restart button once the notification on the bottom right appears.

- Program Robot → Installation → External Control, where has to be set the Host IP. in the case of this project the Host and the UR3 are working with the MIR100, which provides his own connection with IP:192.168.12.20. The Host (computer) is assigned a default IP once connected to the MIR connection, so it only remains to assign the IP to the robot.



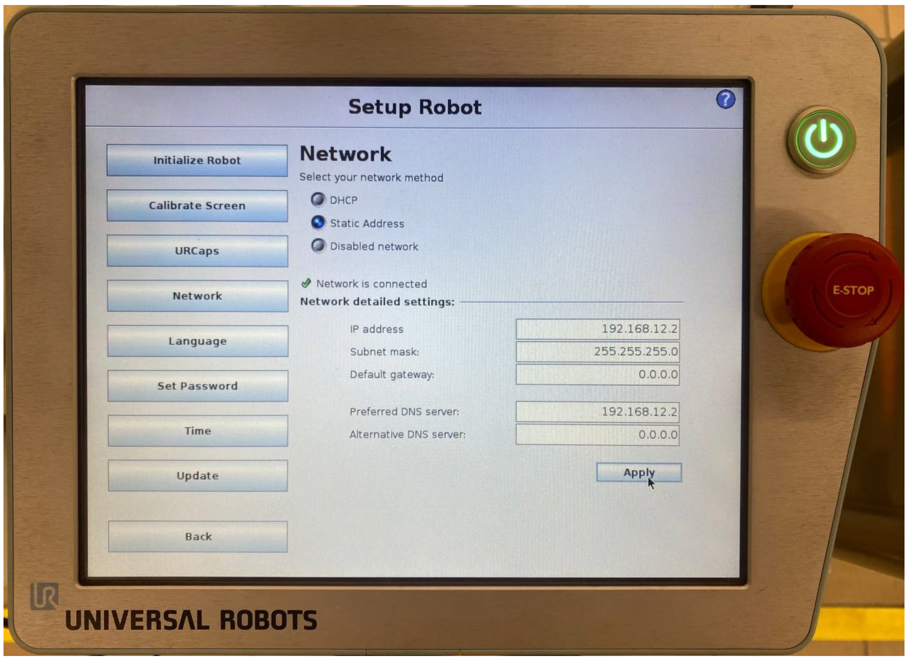
External Control

Host IP: 192.168.12.220
Custom port: 50002
Host name: ExternalControl

Figure 50: Host IP Setting for the External control.

Save the changes in the 'Load\Save' section in the Installation tab.

- Setup Robot → Network, where has to be set the Robot IP. The Host, the UR Robot and the MIR100 IPs have to be in the same Network, since the Host and the MIR IPs are already set, the UR3 IP can be selected by the user, making sure that it is in the same network as the other devices.



Setup Robot

Network

Select your network method

DHCP [✓ Static Address] Disabled network

✓ Network is connected

Network detailed settings:

IP address: 192.168.12.2
Subnet mask: 255.255.255.0
Default gateway: 0.0.0.0
Preferred DNS server: 192.168.12.2
Alternative DNS server: 0.0.0.0

Figure 51: Robot IP Setting for the External control.

- The last step is to create a program with the URcap inside, so that, after the launch of the UR robot driver in ROS, the Cobot can be controlled by the Host, directly from the ROS terminal or using the apps installed in it.
  To do so, from the main screen, Program Robot → Empty Program → Structure → URcaps, select the External control downloaded.

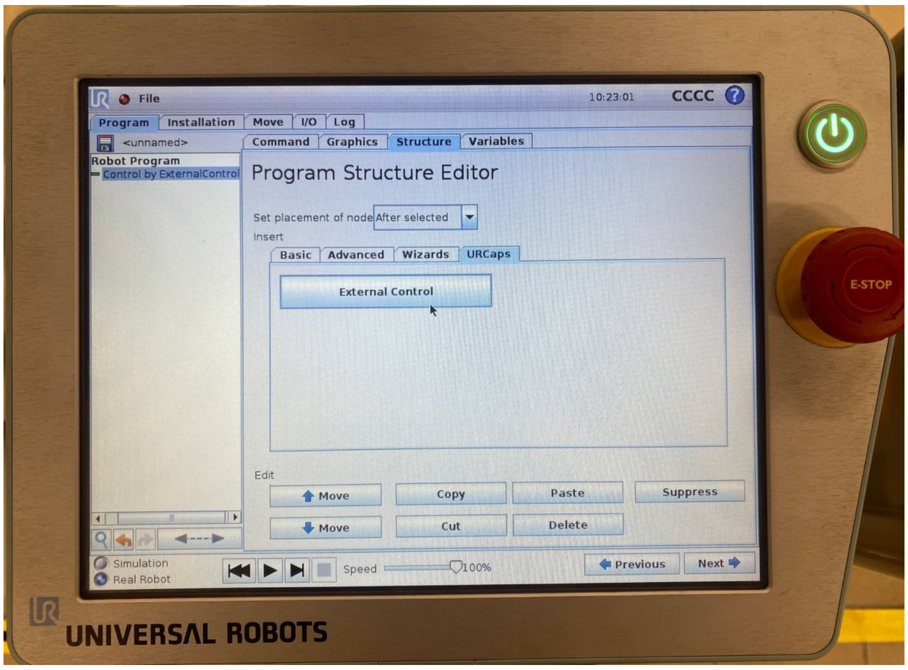
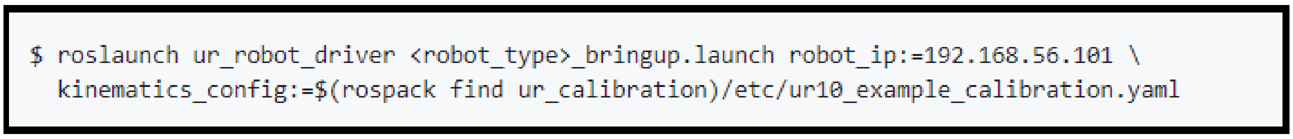


Figure 52: UR3 program for the External control.

Once the URcap and the calibration has been done, the UR robot driver can be launched in ROS:

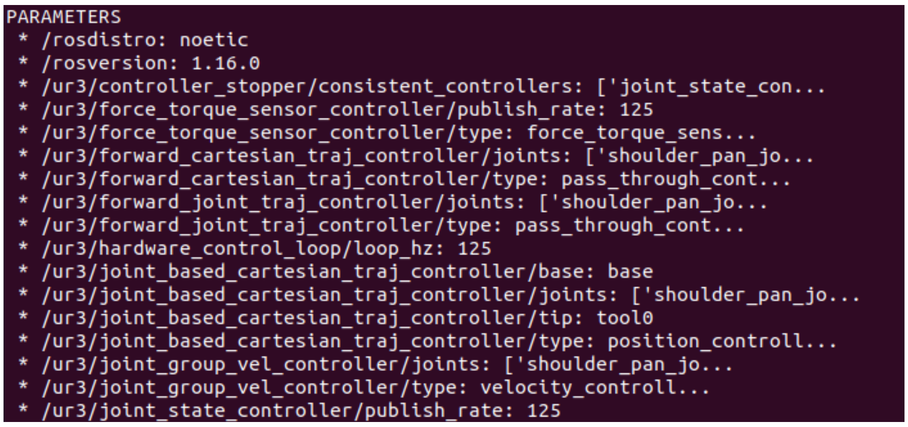


```
$ roslaunch ur_robot_driver <robot_type>_bringup.launch robot_ip:=192.168.56.101 \
  kinematics_config:=$(rospack find ur_calibration)/etc/ur10_example_calibration.yaml
```

Figure 53: Command to launch the UR driver in ROS.

To start controlling the UR3 from ROS, simply run the code in fig.(53), adding the right IP, calibration file and selecting the right UR. If the network is set properly, the launch is successful.
On the UR robot driver launch terminal window is possible to see various useful information:

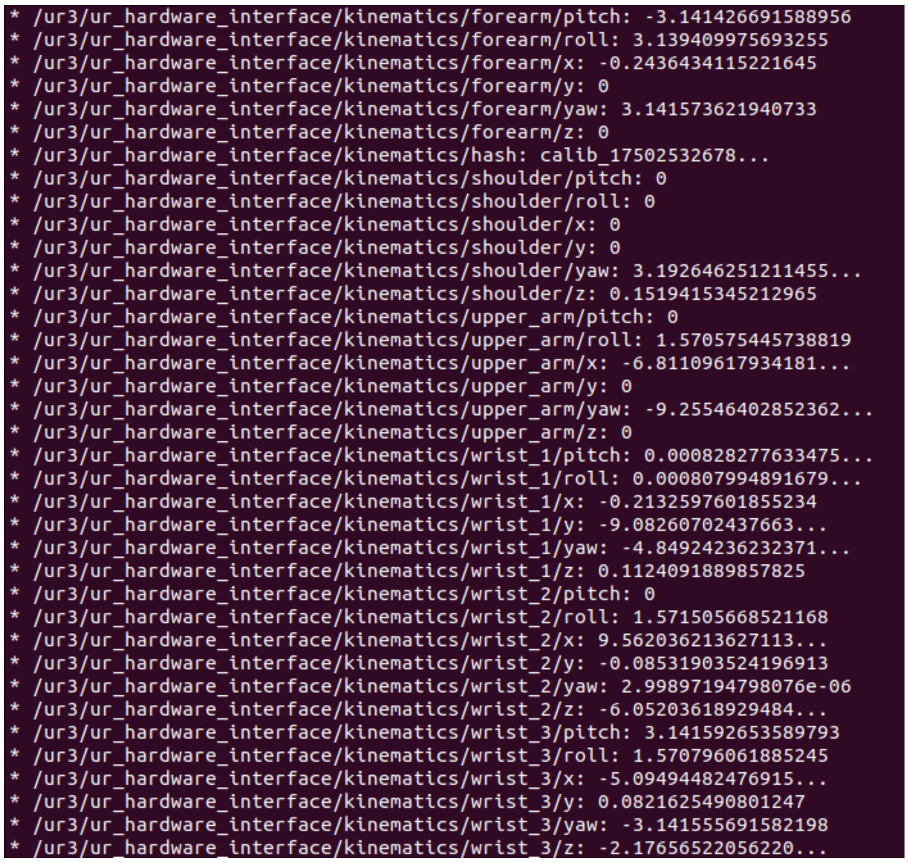
- PARAMETERS: The parameters shows some basic information about the UR3 robot and the Operating System. For example, in Fig.(54) are shown some configuration parameters for the ROS-driver external control:



```
PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /ur3/controller_stopper/consistent_controllers: ['joint_state_con...
 * /ur3/force_torque_sensor_controller/publish_rate: 125
 * /ur3/force_torque_sensor_controller/type: force_torque_sens...
 * /ur3/forward_cartesian_traj_controller/joints: ['shoulder_pan_jo...
 * /ur3/forward_cartesian_traj_controller/type: pass_through_cont...
 * /ur3/forward_joint_traj_controller/joints: ['shoulder_pan_jo...
 * /ur3/forward_joint_traj_controller/type: pass_through_cont...
 * /ur3/hardware_control_loop/loop_hz: 125
 * /ur3/joint_based_cartesian_traj_controller/base: base
 * /ur3/joint_based_cartesian_traj_controller/joints: ['shoulder_pan_jo...
 * /ur3/joint_based_cartesian_traj_controller/tip: tool0
 * /ur3/joint_based_cartesian_traj_controller/type: position_controll...
 * /ur3/joint_group_vel_controller/joints: ['shoulder_pan_jo...
 * /ur3/joint_group_vel_controller/type: velocity_controll...
 * /ur3/joint_state_controller/publish_rate: 125
```

Figure 54: UR3's configuration parameters.

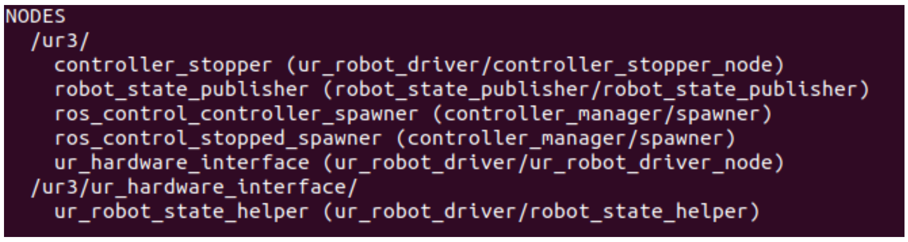
In Fig.(55) instead, are shown the Roll-Pitch-Yaw (RPY) parameters, which represent the orientation of every joint of the UR3:



```
* /ur3/ur_hardware_interface/kinematics/forearm/pitch: -3.141426691588956
* /ur3/ur_hardware_interface/kinematics/forearm/roll: 3.139409975693255
* /ur3/ur_hardware_interface/kinematics/forearm/x: -0.2436434115221645
* /ur3/ur_hardware_interface/kinematics/forearm/y: 0
* /ur3/ur_hardware_interface/kinematics/forearm/yaw: 3.141573621940733
* /ur3/ur_hardware_interface/kinematics/forearm/z: 0
* /ur3/ur_hardware_interface/kinematics/hash: calib_17502532678...
* /ur3/ur_hardware_interface/kinematics/shoulder/pitch: 0
* /ur3/ur_hardware_interface/kinematics/shoulder/roll: 0
* /ur3/ur_hardware_interface/kinematics/shoulder/x: 0
* /ur3/ur_hardware_interface/kinematics/shoulder/y: 0
* /ur3/ur_hardware_interface/kinematics/shoulder/yaw: 3.192646251211455...
* /ur3/ur_hardware_interface/kinematics/shoulder/z: 0.1519415345212965
* /ur3/ur_hardware_interface/kinematics/upper_arm/pitch: 0
* /ur3/ur_hardware_interface/kinematics/upper_arm/roll: 1.570575445738819
* /ur3/ur_hardware_interface/kinematics/upper_arm/x: -6.81109617934181...
* /ur3/ur_hardware_interface/kinematics/upper_arm/y: 0
* /ur3/ur_hardware_interface/kinematics/upper_arm/yaw: -9.25546402852362...
* /ur3/ur_hardware_interface/kinematics/upper_arm/z: 0
* /ur3/ur_hardware_interface/kinematics/wrist_1/pitch: 0.000828277633475...
* /ur3/ur_hardware_interface/kinematics/wrist_1/roll: 0.000807994891679...
* /ur3/ur_hardware_interface/kinematics/wrist_1/x: -0.2132597601855234
* /ur3/ur_hardware_interface/kinematics/wrist_1/y: -9.08260702437663...
* /ur3/ur_hardware_interface/kinematics/wrist_1/yaw: -4.84924236232371...
* /ur3/ur_hardware_interface/kinematics/wrist_1/z: 0.1124091889857825
* /ur3/ur_hardware_interface/kinematics/wrist_2/pitch: 0
* /ur3/ur_hardware_interface/kinematics/wrist_2/roll: 1.571505668521168
* /ur3/ur_hardware_interface/kinematics/wrist_2/x: 9.562036213627113...
* /ur3/ur_hardware_interface/kinematics/wrist_2/y: -0.08531903524196913
* /ur3/ur_hardware_interface/kinematics/wrist_2/yaw: 2.99897194798076e-06
* /ur3/ur_hardware_interface/kinematics/wrist_2/z: -6.05203618929484...
* /ur3/ur_hardware_interface/kinematics/wrist_3/pitch: 3.141592653589793
* /ur3/ur_hardware_interface/kinematics/wrist_3/roll: 1.570796061885245
* /ur3/ur_hardware_interface/kinematics/wrist_3/x: -5.09494482476915...
* /ur3/ur_hardware_interface/kinematics/wrist_3/y: 0.0821625490801247
* /ur3/ur_hardware_interface/kinematics/wrist_3/yaw: -3.141555691582198
* /ur3/ur_hardware_interface/kinematics/wrist_3/z: -2.17656522056220...
```

Figure 55: RPY parameters of the UR3.

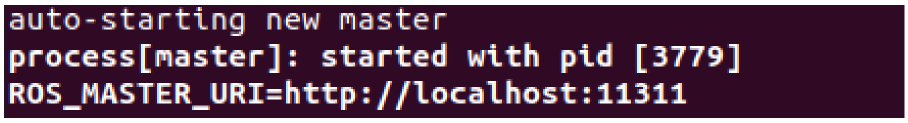
- NODES: The nodes contain all the processes necessary for the UR3 to work properly (fig(56)):



```
NODES
  /ur3/
    controller_stopper (ur_robot_driver/controller_stopper_node)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    ros_control_controller_spawner (controller_manager/spawner)
    ros_control_stopped_spawner (controller_manager/spawner)
    ur_hardware_interface (ur_robot_driver/ur_robot_driver_node)
  /ur3/ur_hardware_interface/
    ur_robot_state_helper (ur_robot_driver/robot_state_helper)
```

Figure 56: Nodes generated by the UR3 driver.

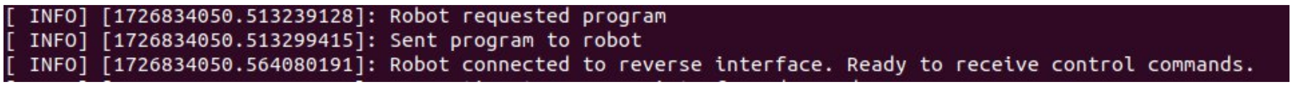
- ROS Master: When the UR3 driver is launched, all the generated nodes will talk to their ROS Master. It can be seen in the terminal as in Fig.(57):



```
auto-starting new master
process[master]: started with pid [3779]
ROS_MASTER_URI=http://localhost:11311
```

Figure 57: ROS Master URI IP address.

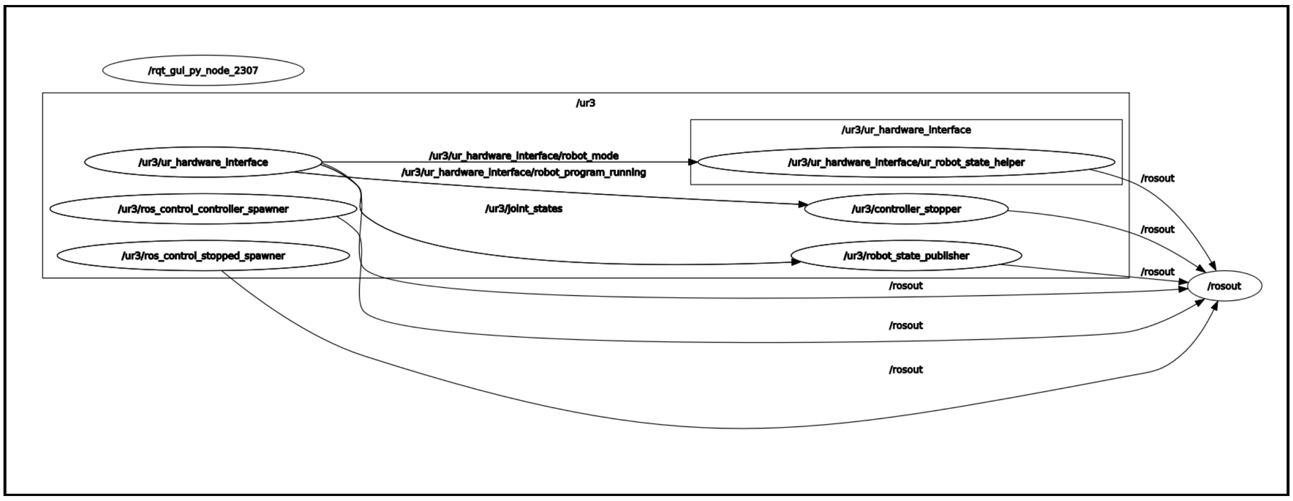
Finally, to start the remote control, is sufficient to run the program on the UR3 teach pendant with the URcap program inside. If everything works correctly the sentence in Fig.(58) should appear on the UR3 driver launch script terminal window.



```
[ INFO] [1726834050.513239128]: Robot requested program
[ INFO] [1726834050.513299415]: Sent program to robot
[ INFO] [1726834050.564080191]: Robot connected to reverse interface. Ready to receive control commands.
```

Figure 58: UR3 is ready to receive commands from the ROS interface.

As for the MIR100, is possible to visualize the communication structure that has been set between ROS and the UR3, thanks to the rqt graph tool (Fig.(59)):



/rqt_gui_py_node_2307
/ur3
/ur3/ur_hardware_interface
/ur3/ur_hardware_interface
/ur3/ur_hardware_interface/robot_mode
/ur3/ur_hardware_interface/robot_program_running
/ur3/ur_hardware_interface/ur_robot_state_helper
/rosout
/ur3/ros_control_controller_spawner
/ur3/joint_states
/ur3/controller_stopper
/rosout
/ur3/ros_control_stopped_spawner
/ur3/robot_state_publisher
/rosout
/rosout
/rosout
/rosout
/rosout

Figure 59: RQT graph of the active UR driver connected to the ROS Master.

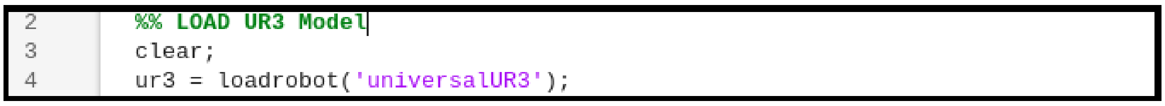
4.3.2 UR3-MATLAB

the construction of the FSM did not require the use of the direct connection between MATLAB and the UR3, because for the control movements was sufficient the use of the URcap provided by the MathWorks and ROS, but it was of vital importance for the initial simulations.
In order to be able to do simulations on the UR3 the following Toolboxes have been added:

- Robotics System Toolbox: Used to load the desired robot model;
- Robotics System Toolbox Supported Hardware: Used to physically move the robot.

First step for the simulation is to launch the UR3 driver and run the URcap program as explained previously. Done that, all the next steps will be in MATLAB:

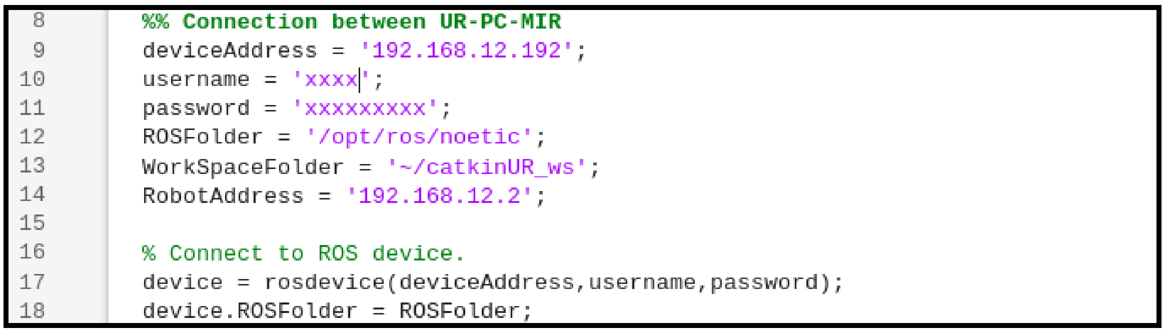
- Load the UR3 rigid model: Thanks to the Robot System Toolbox is possible to load in MATLAB the predefined robot model and his specifications;



```
%% LOAD UR3 Model
clear;
ur3 = loadrobot('universalUR3');
```

Figure 60: MATLAB command to load the UR3 model.

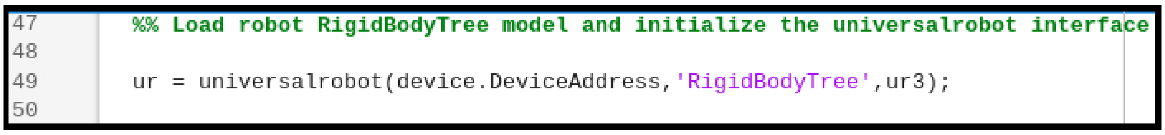
- Connection to ROS repository: With the script in Fig.(61) are specified the information regarding the OS and the paths to the UR repository, from which MATLAB will take the data;



```
%% Connection between UR-PC-MIR
deviceAddress = '192.168.12.192';
username = 'xxxx';
password = 'xxxxxxxxx';
ROSFolder = '/opt/ros/noetic';
WorkSpaceFolder = '~/catkinUR_ws';
RobotAddress = '192.168.12.2';

% Connect to ROS device.
device = rosdevice(deviceAddress,username,password);
device.ROSFolder = ROSFolder;
```

Figure 61: MATLAB commands to connect to the UR driver folders in ROS.

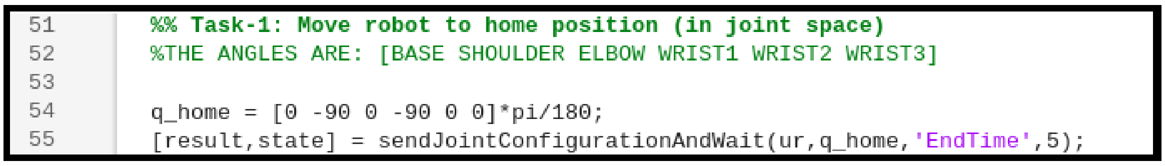
- Connection to the physical UR3: The following command (Fig.(62)) creates a connection with the physical cobot. In the function, additionally, is specified the RigidBodyTree, which contains a complete description of the robot geometry and of his cinematic model;



```
%% Load robot RigidBodyTree model and initialize the universalrobot interface

ur = universalrobot(device.DeviceAddress,'RigidBodyTree',ur3);

```

Figure 62: Connection to the physical UR3.

- Move the cobot: Now that the connection is established it is possible to move the robot with the commands provided by the Toolbox. An example is shown in Fig.(63).



```
%% Task-1: Move robot to home position (in joint space)
%THE ANGLES ARE: [BASE SHOULDER ELBOW WRIST1 WRIST2 WRIST3]

q_home = [0 -90 0 -90 0 0]*pi/180;
[result,state] = sendJointConfigurationAndWait(ur,q_home,'EndTime',5);
```

Figure 63: Command to move the UR3 to the location specified by the joint angles.

This simulation was very important to test the interaction between the UR3 robot and an external control.

4.3.3 UR3-Simulink

There isn't a direct connection between the UR3 and Simulink, because for this project has been found that the easiest way to control the UR3 movements was using ROS shell scripts, but its function is to call those scripts from the Finite State Machine.
In order to find that, some tests have been done, where the FSM in Simulink called an UR3 script defined in MATLAB, so configured as is shown in the Chapter 4.3.2. This simulation has been done to check the stability of the communication of the system.
An example of a simulation controlling UR3 MATLAB scripts with Simulink is shown below in Fig.(64):

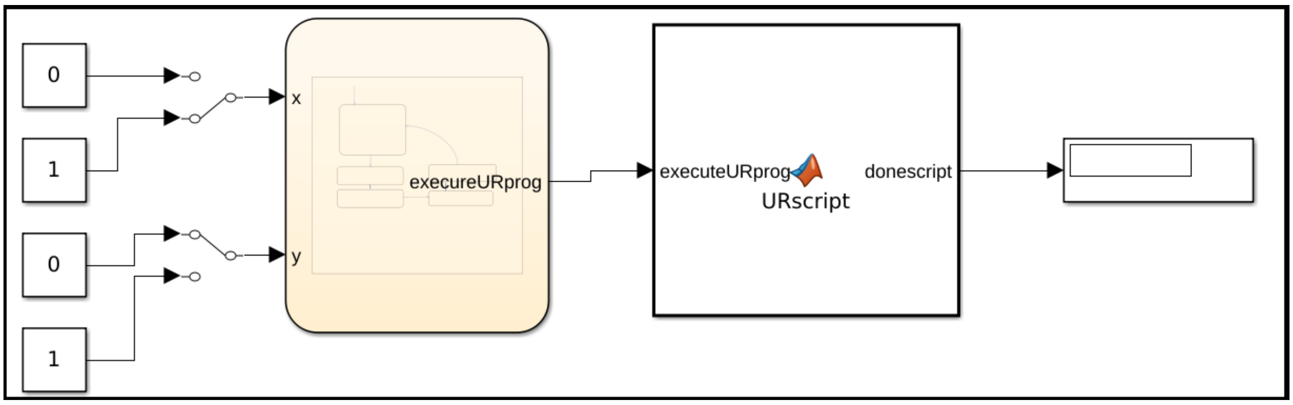
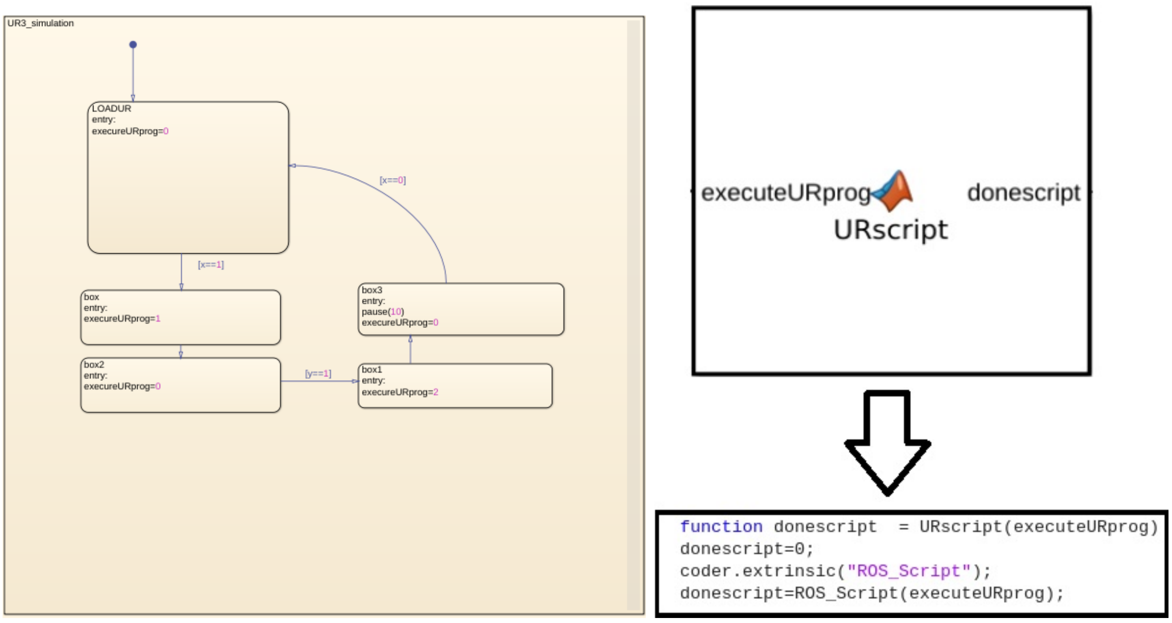


Figure 64: Finite State Machine for the simulation of the UR3 external control with MATLAB and Simulink.

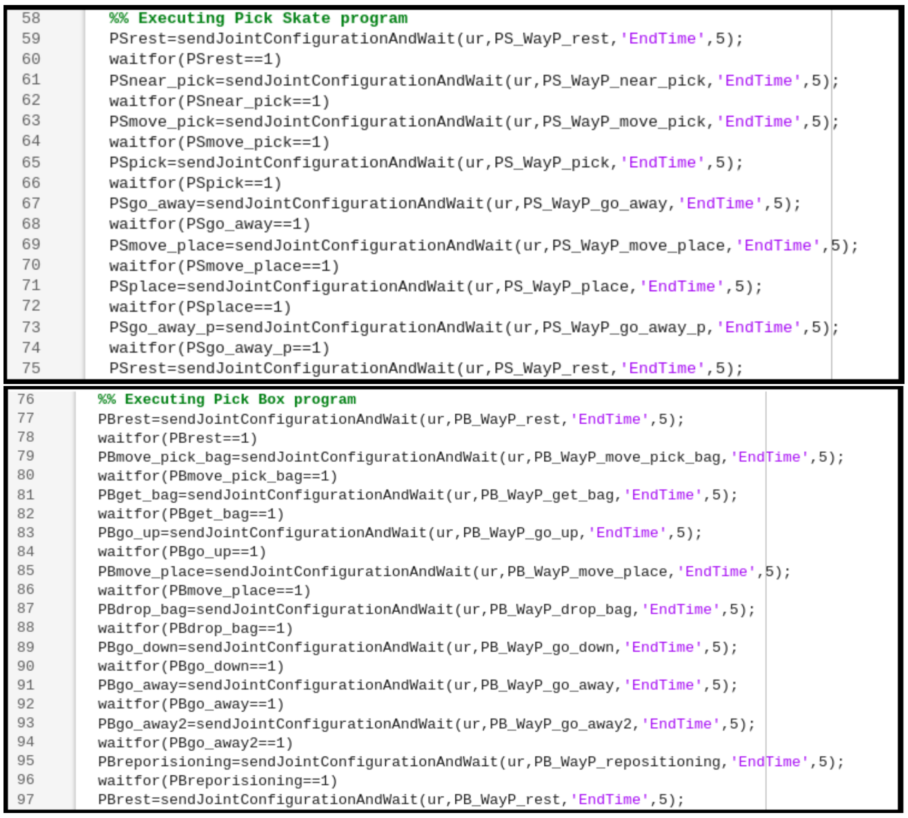
In this FSM, x and y are used to pass manually from a state to another, to decide which UR3 program has to be executed.
In the following picture (Fig.(65)) is represented the FSM and the 'Matlab function' block that recalls the actual MATLAB function, defined in MATLAB.



```
function donescript  = URscript(executeURprog)
donescript=0;
coder.extrinsic("ROS_Script");
donescript=ROS_Script(executeURprog);
```

Figure 65: Sx. FSM testing the UR3 scripts. Dx. Simulink block to call the MATLAB function to execute the UR3 sripts.

Inside the 'ROS_Script' MATLAB function, besides the configuration steps in the Chapter 4.3.2, are defined two UR3 scripts, in the first one (Pick Skate) the UR3 simulates to pick the skateboard from the Warehouse, the second one (Pick box) simulates to pick the box of components always from the Warehouse (in Fig.(66) are just shown the commands to execute the tasks). The waypoints has been defined previously, taking the joint values from the corresponding scripts in the teach pendant.



```
%% Executing Pick Skate program
PSrest=sendJointConfigurationAndWait(ur,PS_WayP_rest,'EndTime',5);
waitfor(PSrest==1)
PSnear_pick=sendJointConfigurationAndWait(ur,PS_WayP_near_pick,'EndTime',5);
waitfor(PSnear_pick==1)
PSmove_pick=sendJointConfigurationAndWait(ur,PS_WayP_move_pick,'EndTime',5);
waitfor(PSmove_pick==1)
PSpick=sendJointConfigurationAndWait(ur,PS_WayP_pick,'EndTime',5);
waitfor(PSpick==1)
PSgo_away=sendJointConfigurationAndWait(ur,PS_WayP_go_away,'EndTime',5);
waitfor(PSgo_away==1)
PSmove_place=sendJointConfigurationAndWait(ur,PS_WayP_move_place,'EndTime',5);
waitfor(PSmove_place==1)
PSplace=sendJointConfigurationAndWait(ur,PS_WayP_place,'EndTime',5);
waitfor(PSplace==1)
PSgo_away_p=sendJointConfigurationAndWait(ur,PS_WayP_go_away_p,'EndTime',5);
waitfor(PSgo_away_p==1)
PSrest=sendJointConfigurationAndWait(ur,PS_WayP_rest,'EndTime',5);
```

```
%% Executing Pick Box program
PBrest=sendJointConfigurationAndWait(ur,PB_WayP_rest,'EndTime',5);
waitfor(PBrest==1)
PBmove_pick_bag=sendJointConfigurationAndWait(ur,PB_WayP_move_pick_bag,'EndTime',5);
waitfor(PBmove_pick_bag==1)
PBget_bag=sendJointConfigurationAndWait(ur,PB_WayP_get_bag,'EndTime',5);
waitfor(PBget_bag==1)
PBgo_up=sendJointConfigurationAndWait(ur,PB_WayP_go_up,'EndTime',5);
waitfor(PBgo_up==1)
PBmove_place=sendJointConfigurationAndWait(ur,PB_WayP_move_place,'EndTime',5);
waitfor(PBmove_place==1)
PBdrop_bag=sendJointConfigurationAndWait(ur,PB_WayP_drop_bag,'EndTime',5);
waitfor(PBdrop_bag==1)
PBgo_down=sendJointConfigurationAndWait(ur,PB_WayP_go_down,'EndTime',5);
waitfor(PBgo_down==1)
PBgo_away=sendJointConfigurationAndWait(ur,PB_WayP_go_away,'EndTime',5);
waitfor(PBgo_away==1)
PBgo_away2=sendJointConfigurationAndWait(ur,PB_WayP_go_away2,'EndTime',5);
waitfor(PBgo_away2==1)
PBreporisioning=sendJointConfigurationAndWait(ur,PB_WayP_repositioning,'EndTime',5);
waitfor(PBreporisioning==1)
PBrest=sendJointConfigurationAndWait(ur,PB_WayP_rest,'EndTime',5);
```

Figure 66: MATLAB commands to execute the Pick Skate and Pick Box tasks.

After this simulation it turned out that, the MATLAB script wasn't stable enough to work in a Finite State Machine, because it made the process slow, less flexible and made the system crash often, so for the project the UR3 scripts are developed in shell scripts in ROS, where Simulink, will call a MATLAB function, which will in turn call those scripts.

4.3.4 UR3-OnRobot GR2 Gripper

The OnRobot GR2 Gripper is already installed in the UR3 and is easily controlled by the teach pendant, piloting the Tool output pin related to it (Fig.(67-68)).

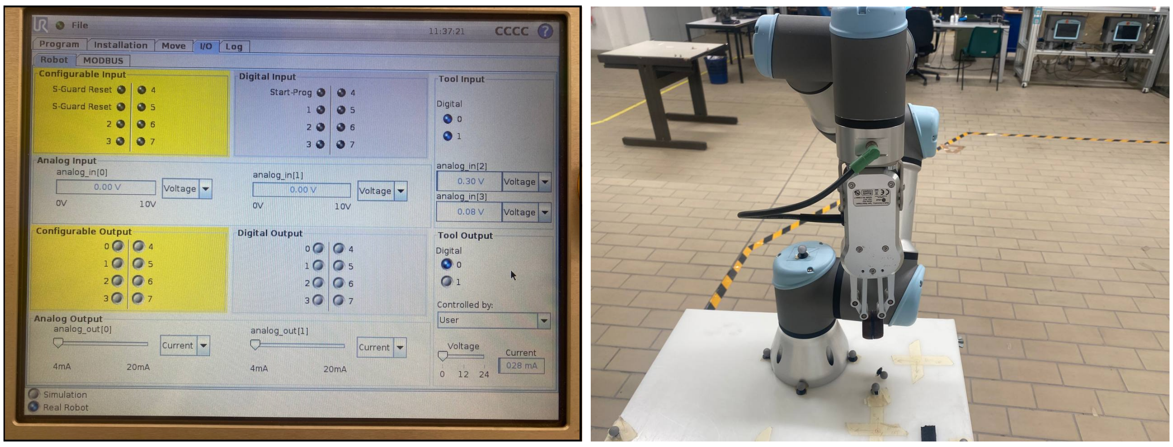


Figure 67: Tool Output pin [0] closes the pins when is '1'.

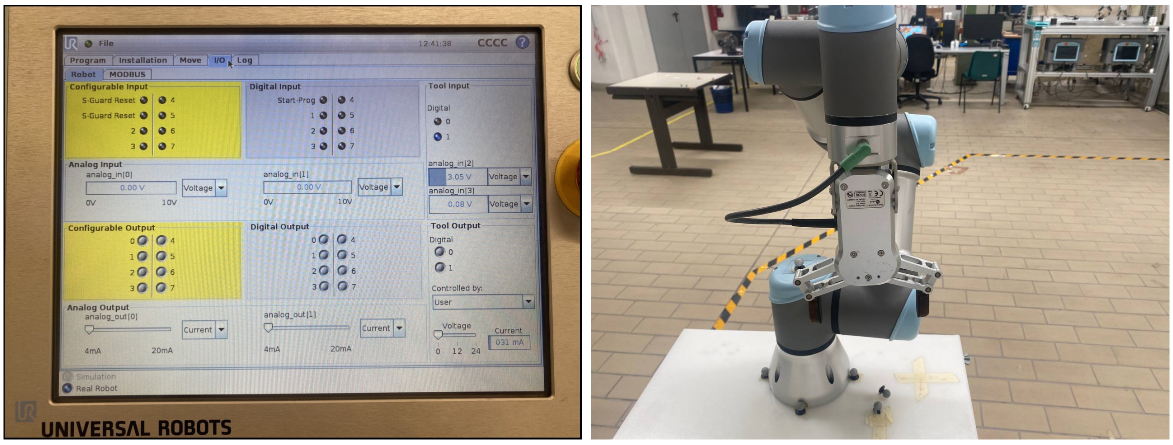
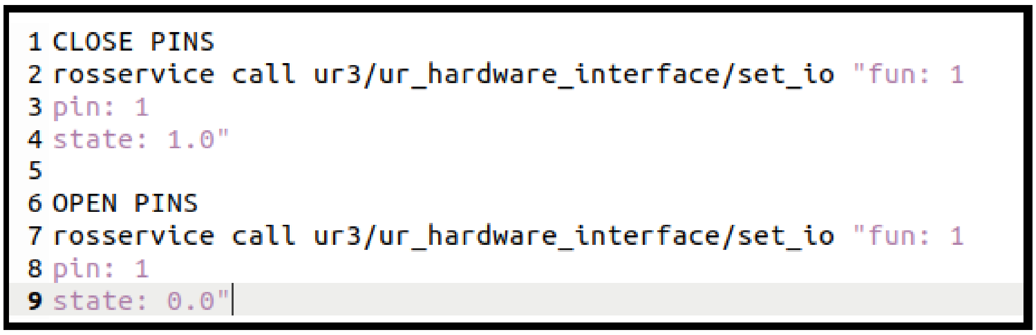


Figure 68: Tool Output pin [0] opens the pins when is '0'.

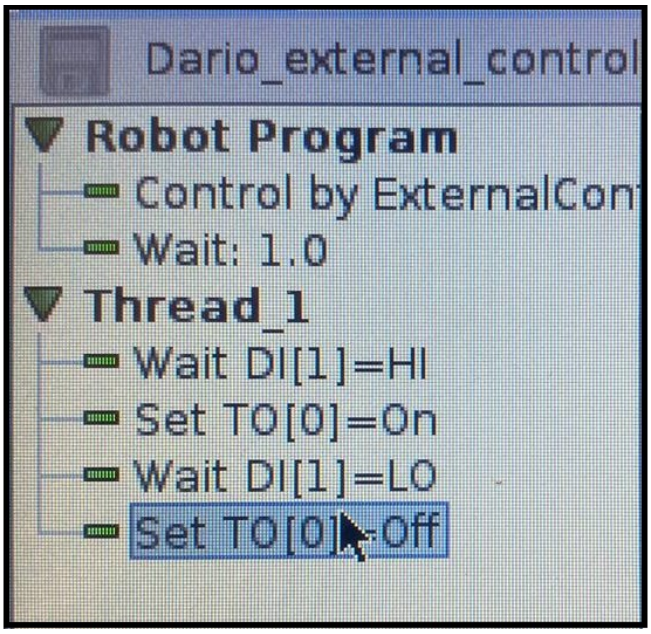
To use this tool in ROS without having to download dedicated packages, it is possible to control the digital outputs of the UR3 thanks to the packages downloaded previously in the Chapter 4.3.1.
The ROS command that permits to control the UR3 I/O pins is: rosservice.
With this command is possible to toggle the Digital Outputs of the UR3:



```
CLOSE PINS
rosservice call ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

OPEN PINS
rosservice call ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"
```

Figure 69: Rosservice script to toggle the Digital Output pin [0].

This script allows to control the Digital Output [0]. In order to link the Digital Output to the Tool output has been added a script into the URcap program in the Teach Pendant:



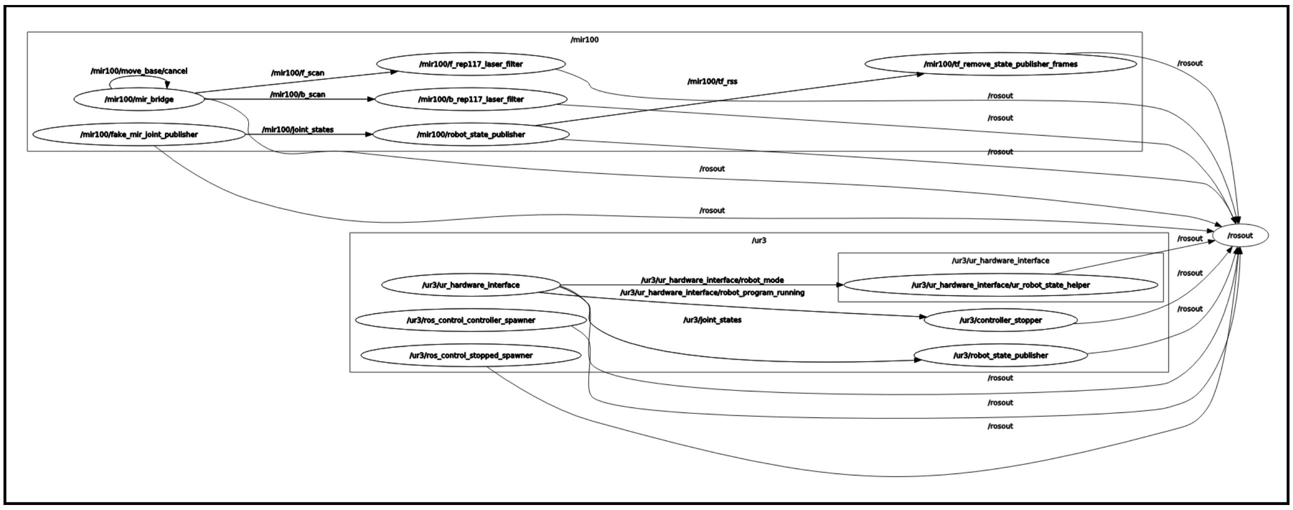
```
Dario_external_control
▼ Robot Program
   Control by ExternalCon
   Wait: 1.0
▼ Thread_1
   Wait DI[1]=HI
   Set TO[0]=On
   Wait DI[1]=LO
   Set TO[0]=Off
```

Figure 70: Teach Pendant script to link the Digital Output pin [0] to the Tool Output pin [0].

With this model, during the execution of a task, every time that the Digital Output pin is toggled by the rosservice command it will toggle consequently the Tool Output pin, which will activate the end effector.

4.4 Complete Network

Connecting both MIR100 and the UR3 to the computer the final network has been created, where the Host can control the two robots simultaneously.
With the rqt graph is possible to see all the active nodes of both robots communicating with the ROS Master:

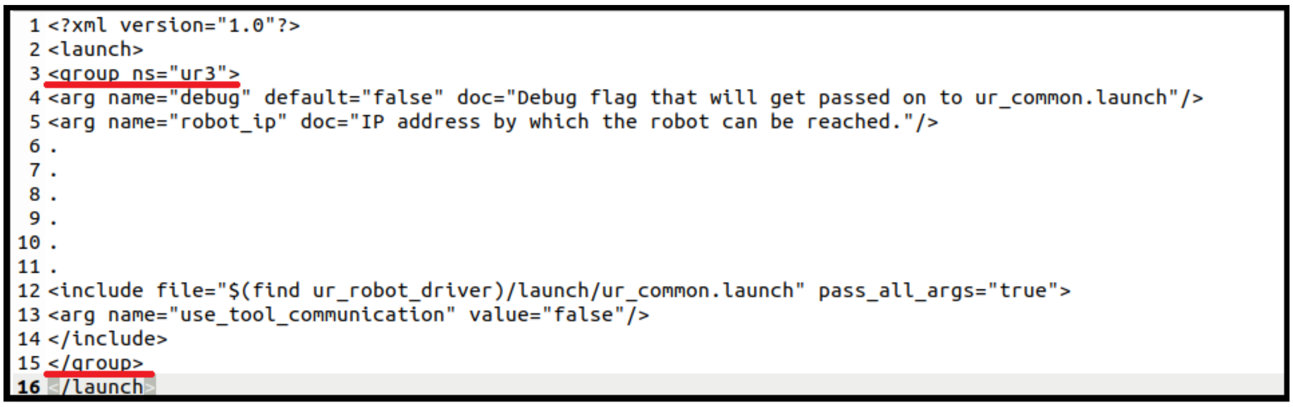


/mir100
/mir100/move_base/cancel
/mir100/mir_bridge
/mir100/f_scan
/mir100/b_scan
/mir100/f_rep117_laser_filter
/mir100/b_rep117_laser_filter
/mir100/fake_mir_joint_publisher
/mir100/joint_states
/mir100/robot_state_publisher
/mir100/tf_rss
/mir100/tf_remove_state_publisher_frames
/rosout
/ur3
/ur3/ur_hardware_interface
/ur3/ur_hardware_interface/robot_mode
/ur3/ur_hardware_interface/robot_program_running
/ur3/ur_hardware_interface/ur_robot_state_helper
/ur3/ros_control_controller_spawner
/ur3/joint_states
/ur3/controller_stopper
/ur3/ros_control_stopped_spawner
/ur3/robot_state_publisher

Figure 71: RQT graph of the complete MIR100-UR3-Pc Network.

An important consideration about this graph is that, as is shown in the picture, the nodes of the robots are encapsulated in two different groups, because the two drivers have common nodes, which created conflicts in the communication of the network.
In order to avoid this problem the two launch files has been modified as follows:

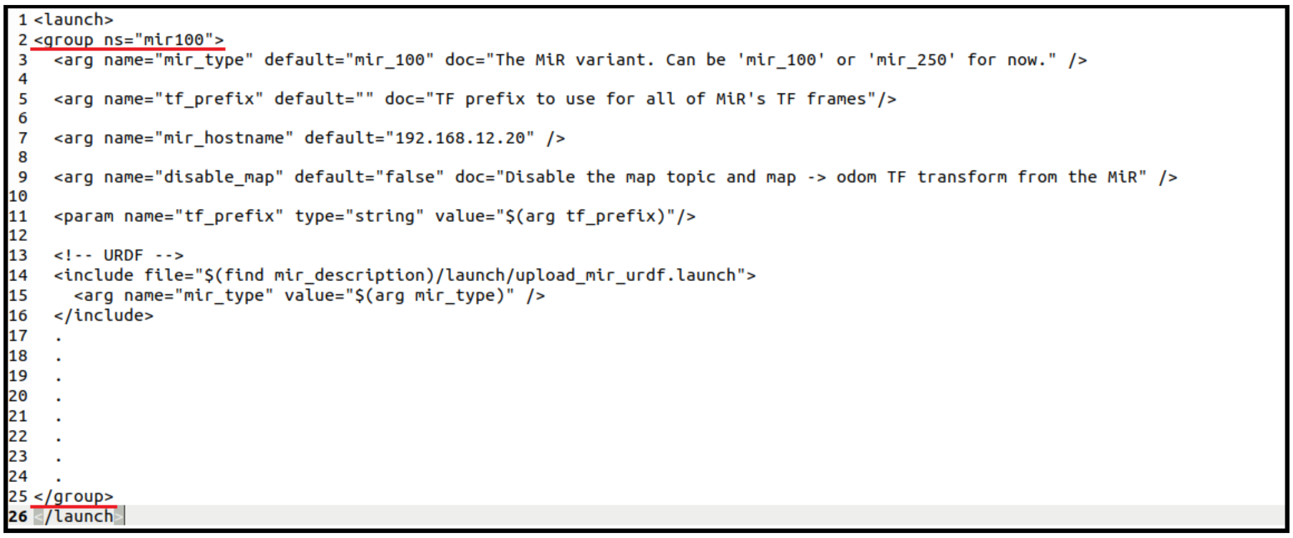
- *ur3_bringup.launch* file: A group called 'ur3' has been added to the script, enclosing all the nodes of the driver:



```
<?xml version="1.0"?>
<launch>
<group ns="ur3">
<arg name="debug" default="false" doc="Debug flag that will get passed on to ur_common.launch"/>
<arg name="robot_ip" doc="IP address by which the robot can be reached."/>
.
.
.
.
.
.
<include file="$(find ur_robot_driver)/launch/ur_common.launch" pass_all_args="true">
<arg name="use_tool_communication" value="false"/>
</include>
</group>
</launch>
```

Figure 72: Modified ur3_bringup.launch script.

- *mir.launch* file: A group called 'mir100' has been added to the script, enclosing all the nodes of the driver:



```
<launch>
<group ns="mir100">
  <arg name="mir_type" default="mir_100" doc="The MiR variant. Can be 'mir_100' or 'mir_250' for now." />

  <arg name="tf_prefix" default="" doc="TF prefix to use for all of MiR's TF frames"/>

  <arg name="mir_hostname" default="192.168.12.20" />

  <arg name="disable_map" default="false" doc="Disable the map topic and map -> odom TF transform from the MiR" />

  <param name="tf_prefix" type="string" value="$(arg tf_prefix)"/>

  <!-- URDF -->
  <include file="$(find mir_description)/launch/upload_mir_urdf.launch">
    <arg name="mir_type" value="$(arg mir_type)" />
  </include>
  .
  .
  .
  .
  .
  .
  .
  .
</group>
</launch>
```

Figure 73: Modified mir.launch script.

The previous figures are not counting another communication in the system, which is the one between MATLAB and the MIR100 Master. The figure shown below (Fig.(74)) shows the communication between all the Nodes and Masters of the system:

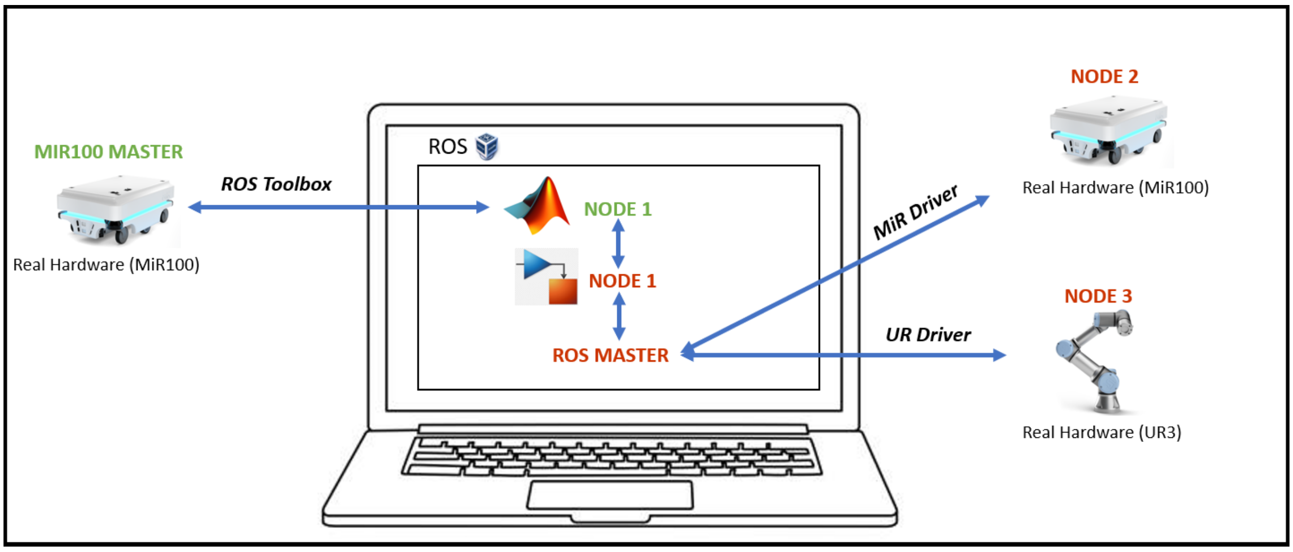


Figure 74: Communication between all the Nodes and Masters present in the system.

The correspondent physical network is composed as follows:

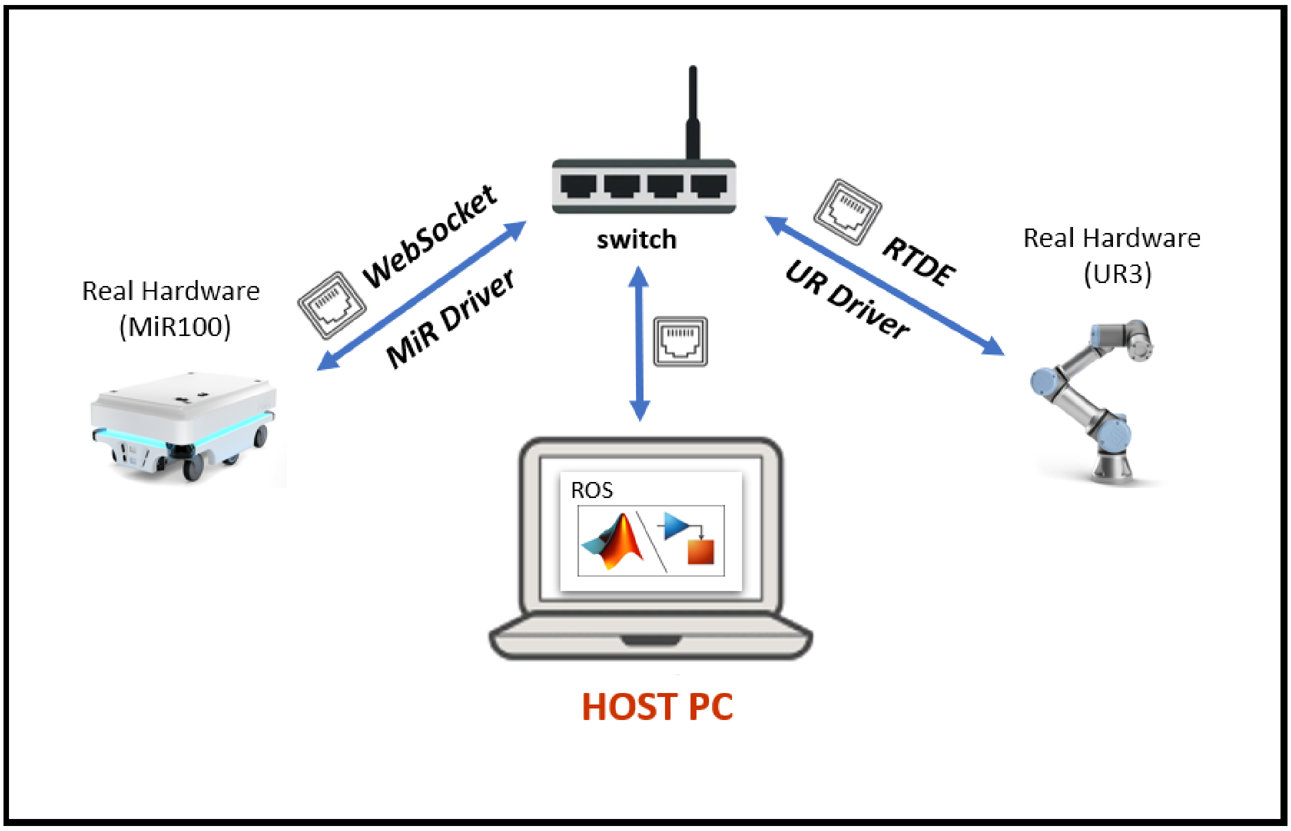


Figure 75: Final Network configuration.

In this system there are two different communication protocols:

- Real Time Data Exchange (RTDE)[9]: Is a communication protocol created by the Universal Robots, allows to receive and transmit data in real time between the UR controller and an external device over a standard TCP/IP connection.
- WebSocket: It is a communication protocol which permits a bidirectional connection between Client and Server through a TCP/IP socket, so as the RTDE permits the exhange of data in real time .

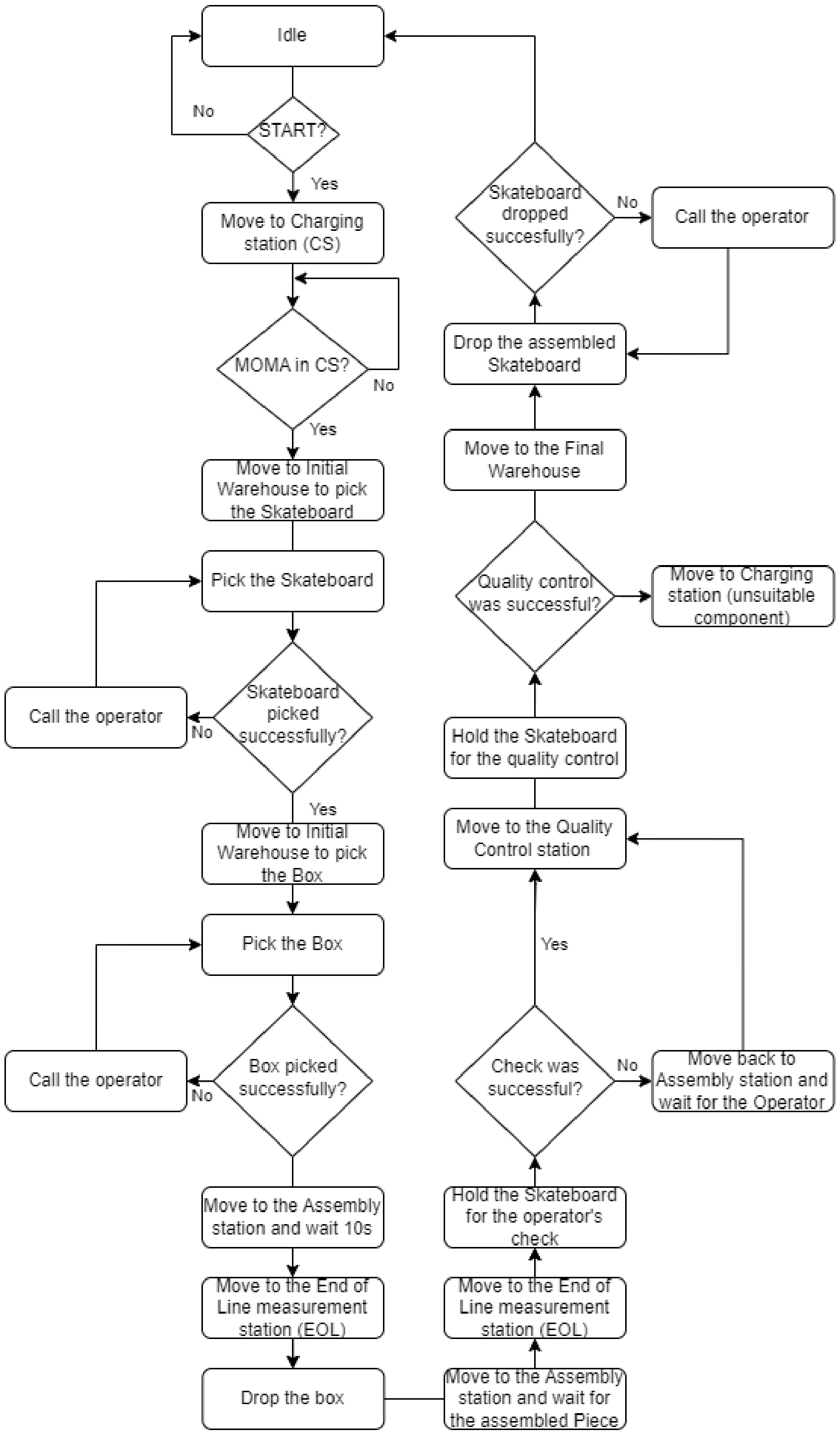
[9]RTDE official guide: https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/

5 Finite State Machine

The finite state machine has been developed in Simulink. Is made to simulate a production line where a skateboard is assembled, handling in the meantime non-standard situations. The advantage of this machine is that is very versatile, additionally, external devices can be added to the system to improve it.

5.1 Flow Chart

Before creating the Finite State Machine, the Flow chart of a cycle of production of the MOMA has been designed:



Idle
START?
No
Yes
Move to Charging station (CS)
MOMA in CS?
No
Yes
Move to Initial Warehouse to pick the Skateboard
Pick the Skateboard
Call the operator
Skateboard picked successfully?
No
Yes
Move to Initial Warehouse to pick the Box
Pick the Box
Call the operator
Box picked successfully?
No
Move to the Assembly station and wait 10s
Move to the End of Line measurement station (EOL)
Drop the box
Move to the Assembly station and wait for the assembled Piece
Move to the End of Line measurement station (EOL)
Hold the Skateboard for the operator's check
Check was successful?
No
Move back to Assembly station and wait for the Operator
Yes
Move to the Quality Control station
Hold the Skateboard for the quality control
Quality control was successful?
Move to Charging station (unsuitable component)
Move to the Final Warehouse
Drop the assembled Skateboard
Skateboard dropped succesfully?
No
Call the operator

Figure 76: Flow chart of a MOMA's cycle of production.

5.2 Complete System

The system is divided in different sections, there are all the blocks provided by the MATLAB Toolboxes that capture data in real-time to compute the next state of the FSM, in the meantime, the FSM will provide output signals that will be used by the MATLAB functions to call the UR3 and MIR100 scripts to execute the tasks.

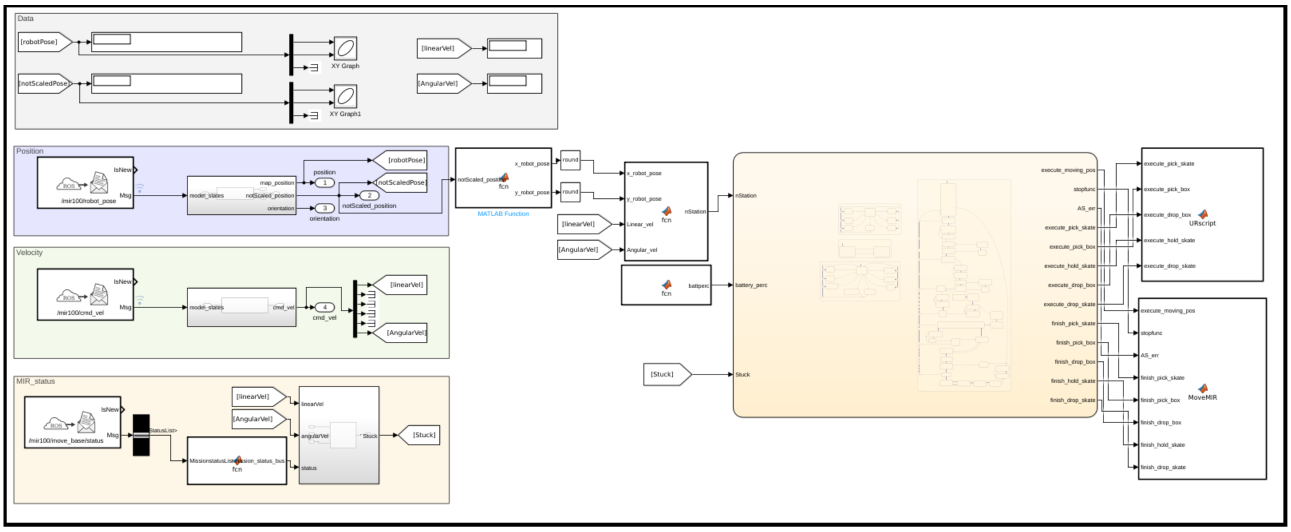


Figure 77: Complete system of the FSM of the MOMA .

5.2.1 MOMA's FSM

In Fig.(78) is represented the Finite State Machine for the MOMA's line of production, which sends the signals to run the MOMA's scripts and handles different types of situations that may occur during the work.
It receives three input signals:

- nStation: This signal is used to compute the actual position of the robot, to make it aware of where it is at any time. This is also useful for handling non standard situation, because based on where the MOMA is, it can understand easily what type of problem could be. The generation of this signal will be explained in Chapter 5.2.2.
- battery_perc: This signal calculates in real time the percentage of the MIR100 battery, so when it is below a certain value the MOMA will interrupt his work and will move to the charging station. The generation of this signal will be explained in Chapter 5.2.3.
- Stuck: During the movement of the MIR100 around the working environment, there could be a lot of unexpected situations, due to people moving around, interacting with the robot in a different way than one would imagine, or due to obstacles in the line of production. All these situations lead the robot to send different status signals, that are elaborated so that it is able to handle all of them. The generation of this signal will be explained in Chapter 5.2.4.

It provides 13 output signals:

- execute_moving_pos: This signal enters in the 'MoveMIR' MATLAB function block. Every value of this parameter corresponds to a working station, so it is used to select on which the MIR100 has to move to.
- stopfunc: This signal enters in the 'MoveMIR' MATLAB function block. It is used avoid reiterating the MIR100 script several times.
- AS_error: This signal enters in the 'MoveMIR' MATLAB function block. It is used in case the component check in the EOL station is not successful. This signal should be connected to a button to report the problem, but for the purposes of this project has just been designed to function theoretically.
- finish_pick_skate: This signal enters in the 'MoveMIR' MATLAB function block. This signal indicates when the UR3 finished getting the skateboard.
- finish_pick_box: This signal enters in the 'MoveMIR' MATLAB function block. This signal indicates when the UR3 finished getting the box.

- finish_drop_box: This signal enters in the 'MoveMIR' MATLAB function block. This signal indicates when the UR3 finished dropping the box in the EOL station.
- finish_hold_skate: This signal enters in the 'MoveMIR' MATLAB function block. This signal indicates when the UR3 finished the human-robot collaboration task.
- finish_drop_skate: This signal enters in the 'MoveMIR' MATLAB function block. This signal indicates when the UR3 finished dropping the assembled skateboard in the Final Warehouse.
- execute_pick_skate: This signal enters in the 'URscript' MATLAB function block. This signal permits to the UR3 to execute the skateboard pick-up command(Appendix2.1).
- execute_pick_box: This signal enters in the 'URscript' MATLAB function block. This signal permits to the UR3 to execute the box pick-up command(Appendix2.2).
- execute_drop_box: This signal enters in the 'URscript' MATLAB function block. This signal permits to the UR3 to execute the script to drop the box in the EOL station(Appendix2.3).
- execute_hold_skate: This signal enters in the 'URscript' MATLAB function block. This signal permits to the UR3 to execute the human-robot collaboration task(Appendix2.4).
- execute_drop_skate: This signal enters in the 'URscript' MATLAB function block. This signal permits to the UR3 to execute the drop box task in the Final Warehouse(Appendix2.5).

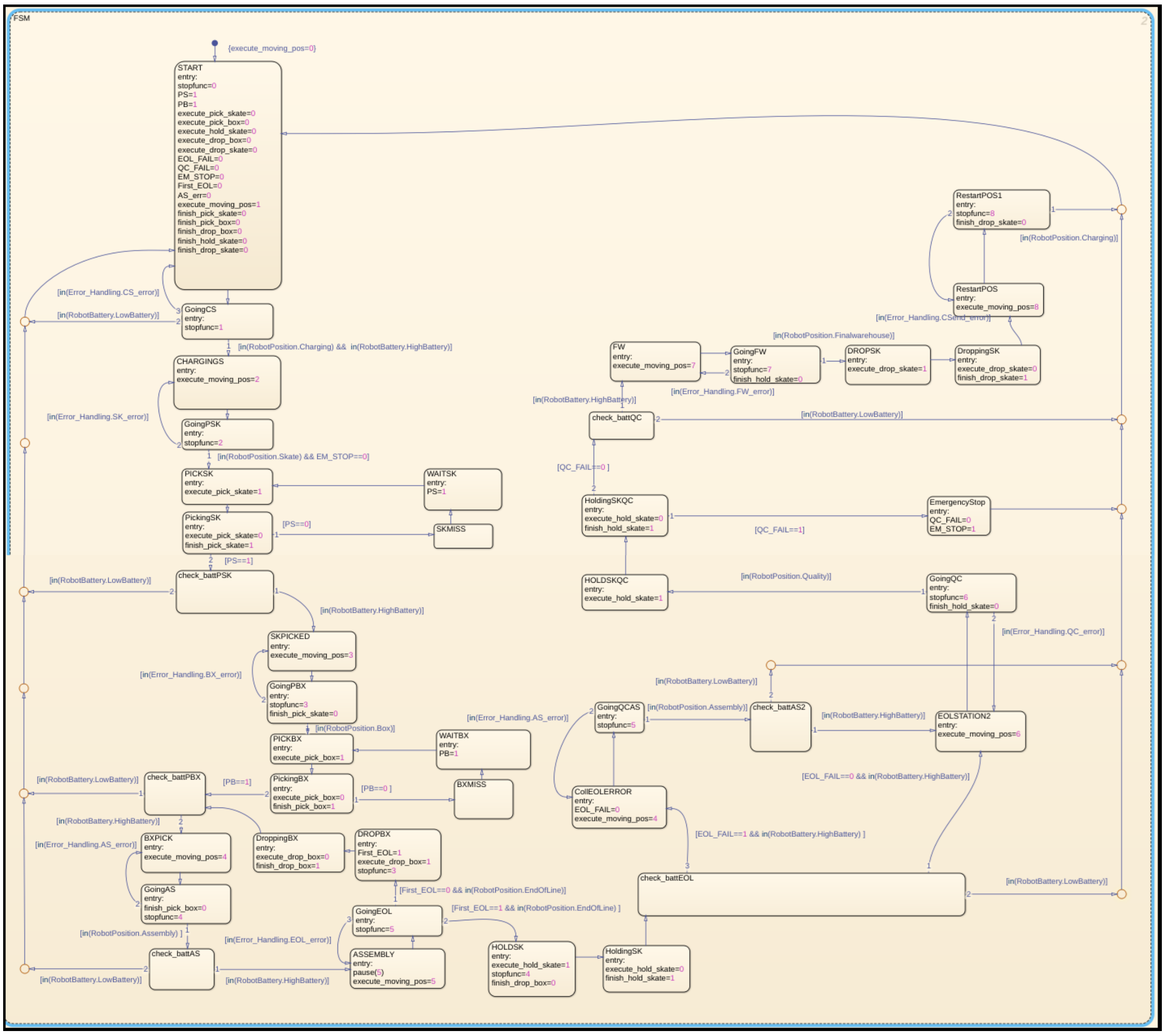


Figure 78: Finite State Machine of the MOMA's line of production .

5.2.2 Robot Position

The calculation of the nStatus input signal is composed by different blocks:

- Robot Pose calculation: In this section there is a block that provides, thanks to the ROS Toolbox, the real time position of the MIR100. Of the coordinates, only the value referring to the x and y axes is taken and then rounded up one digit after the decimal point.

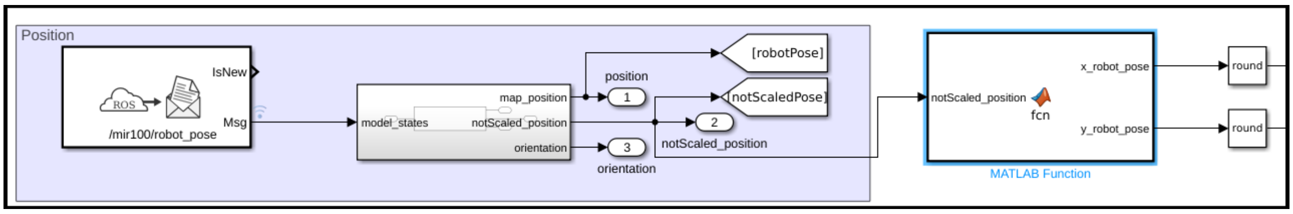


Figure 79: Robot_Pose topic subscribed from ROS.

- Robot Velocity: This block provides, thanks to the ROS Toolbox, the real time value of the linear and angular velocity of the Mobile Robot.

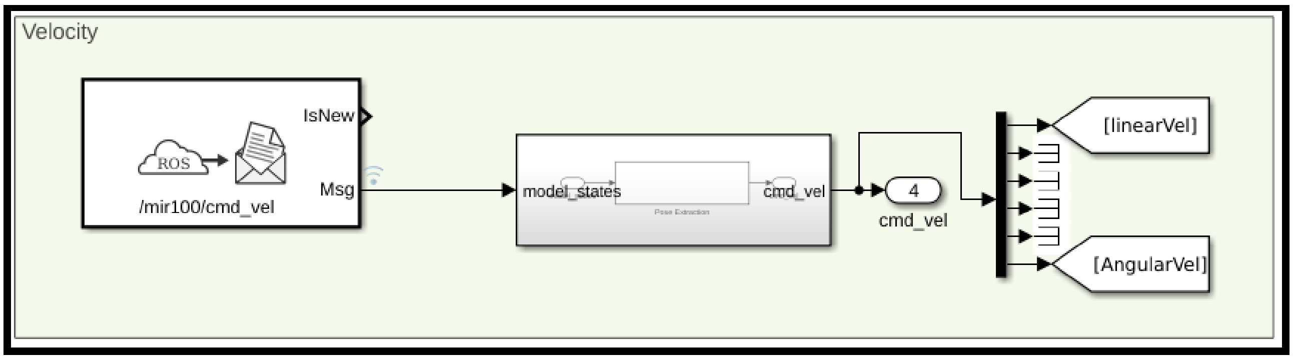


Figure 80: Cmd_vel topic subscribed from ROS.

Both position and velocity data are then sent to the 'Robopos' MATLAB function block (Appendix1.1) which will compute the value of the nStation pin based on the collected data.
The nStation input pin, based on its value, will decree in which station is the MOMA as shown below:

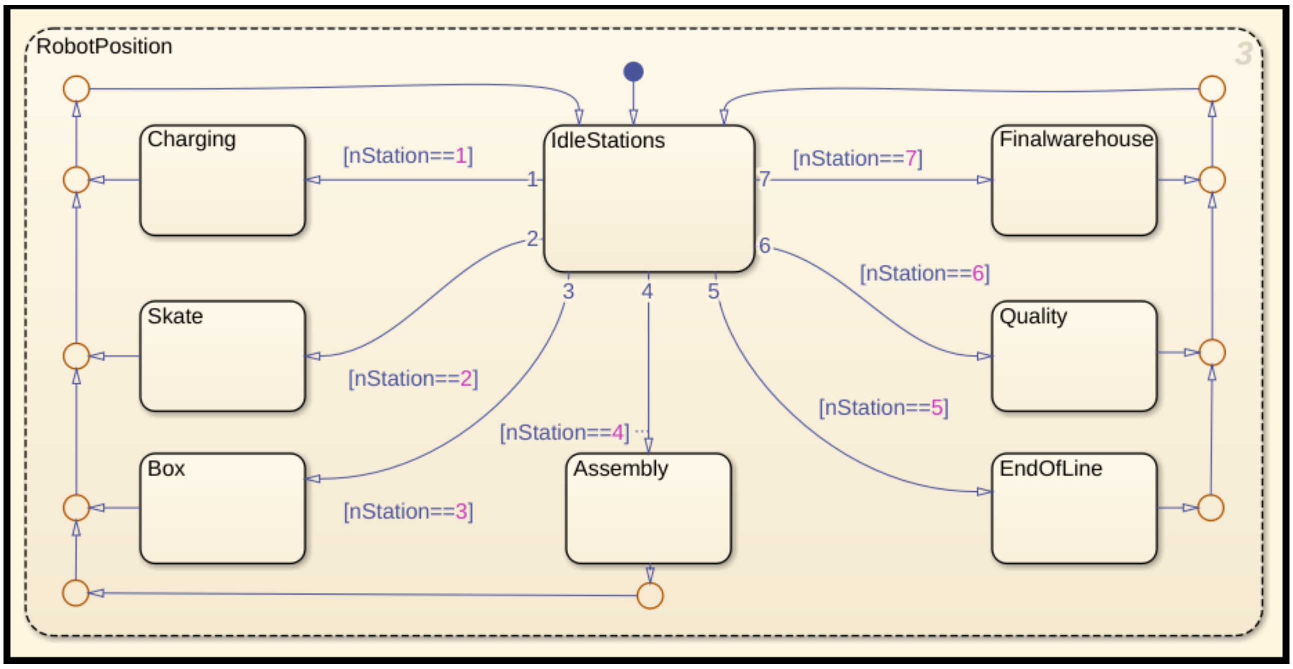
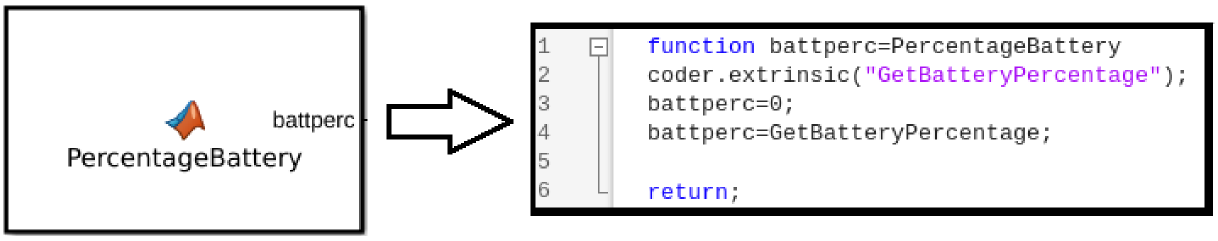


Figure 81: State flow of the Robot Position based on the 'nStation' variable.

These states will pilot the FSM during the execution, coordinating both MIR100 and UR3 and making the error handling more intelligent.

5.2.3 Robot Battery

The acquisition of the Battery percentage has not been done in Simulink, but MATLAB instead, because in Simulink wasn't available the Topic needed to get the value not being connected to the MIR100 Master; so a Simulink MATLAB function block calls a MATLAB function (Appendix1.2), which can get the battery percentage by subscribing the 'battery_percentage' topic and passing the value back to Simulink.



```
function battperc=PercentageBattery
coder.extrinsic("GetBatteryPercentage");
battperc=0;
battperc=GetBatteryPercentage;

return;
```

Figure 82: Matlab function block calling the function in MATLAB to get the battery percentage.

In Simulink, the value obtained from the function will be an input of the FSM and is used to decide if the MIR100 can continue the work:

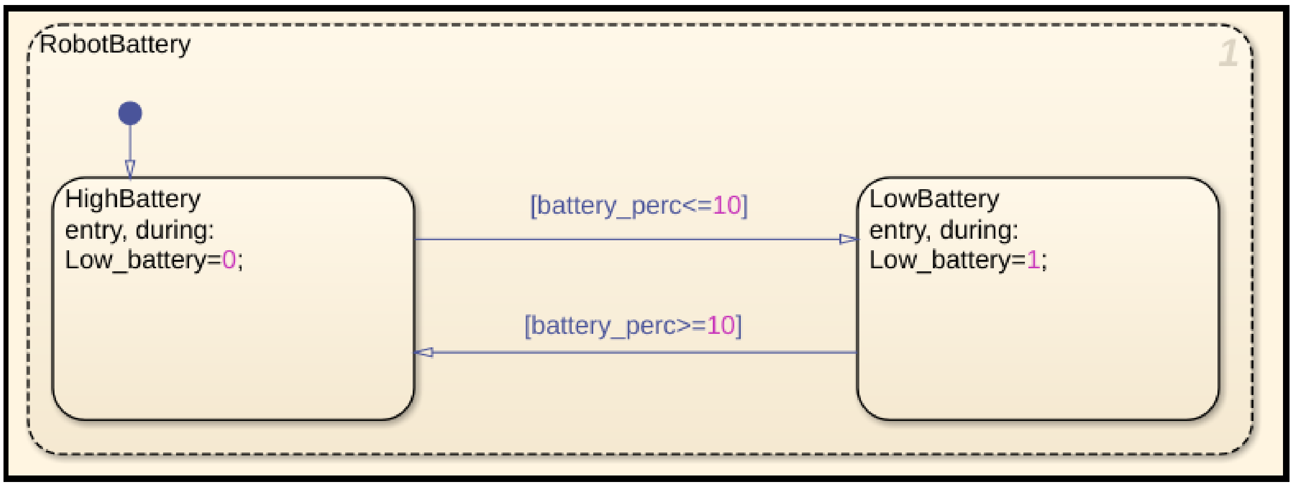


Figure 83: State Flow of battery monitoring.

Based on the value of the battery percentage (decided by the user) there are two different states. If the system is in the 'LowBattery' state, the MIR100 will go back to the charging station, otherwise it will keep working.

5.2.4 Error Handling

This section is made by getting, using again the subscribe block provided by the ROS Toolbox, the 'move_base/status' topic, which gives information on the status of the goals the mobile robot is trying to achieve.

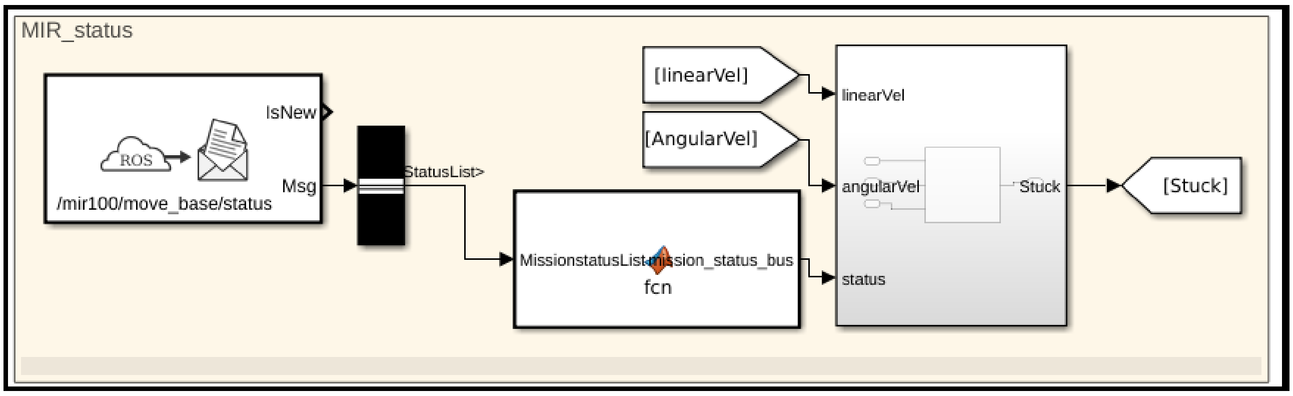


Figure 84: 'Move_base/status' topic subscribed from ROS.

The 'Stuck' signal will be equal to '1' if some problems occur during the mission of the MIR100. It depends on the values got by the topic. The most important are:

- $move\_base/status = 0 \longrightarrow PENDING$: The target has been received, but has not yet been processed.
- $move\_base/status = 1 \longrightarrow ACTIVE$: The target is currently running.
- $move\_base/status = 3 \longrightarrow SUCCEEDED$: The objective was successfully completed.
- $move\_base/status = 4 \longrightarrow ABORTED$: The objective was aborted.
- $move\_base/status = 5 \longrightarrow REJECTED$: The target was rejected and will not be executed.
- $move\_base/statusck = 9 \longrightarrow LOST$: The target has been lost and cannot be tracked any further.

When the execution of the goal has some problems, it will be discovered through the 'move_base/status' value, that will set the variable 'Stuck' to 1, so the system will now if the MIR100 is stuck and where, thanks to the 'stopfunc' variable (Fig.85).

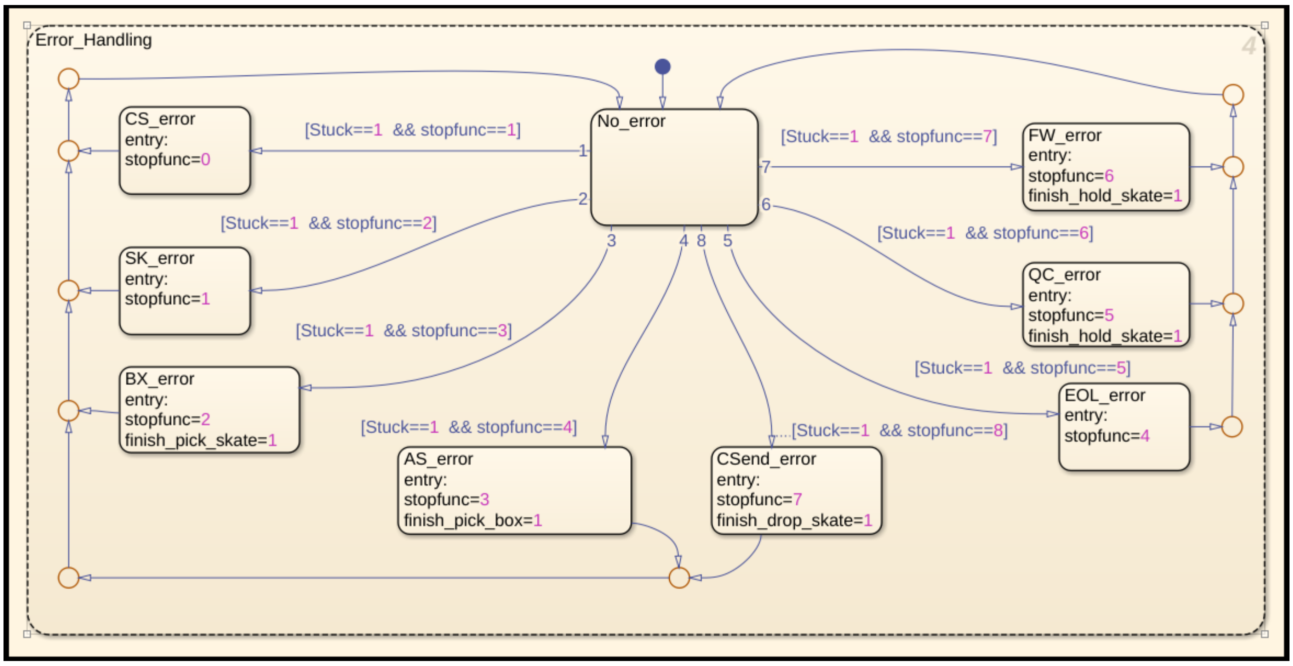
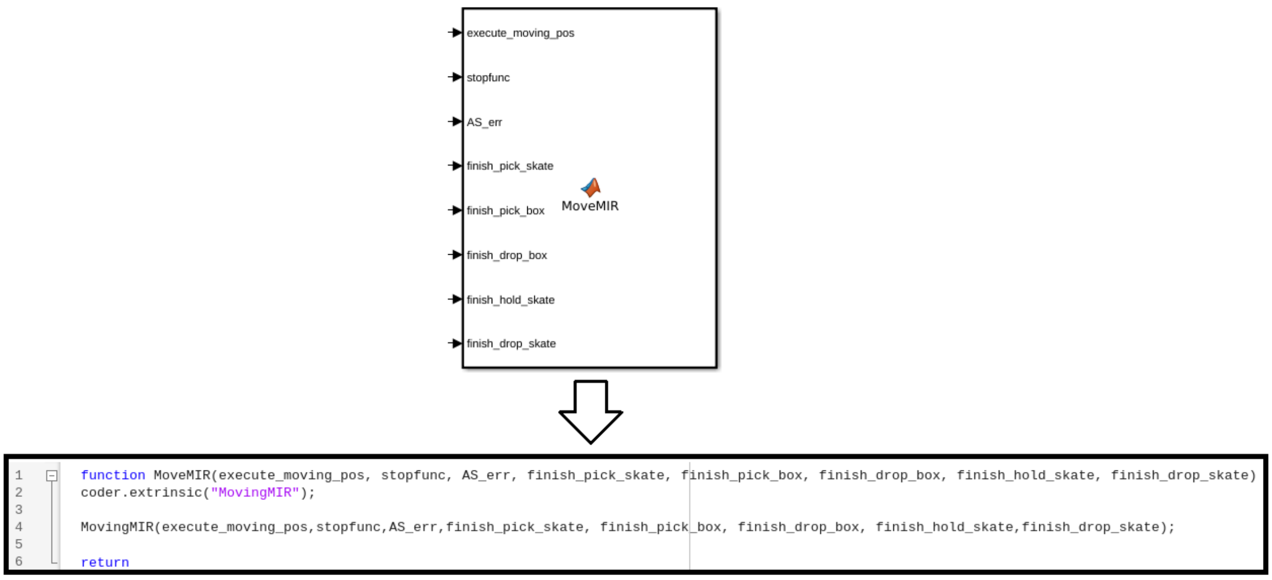


Figure 85: State flow of the Error Handling based on the 'Stuck' and 'stopfunc' variables.

5.2.5 Moving MIR-MATLAB function block

This 'MovingMIR' block, driven by the FSM outputs, calls a function defined in MATLAB(Appendix1.3), which will execute the shell scripts defined in ROS corresponding to the movements the MIR100 must make to perform the job(Appendix3).



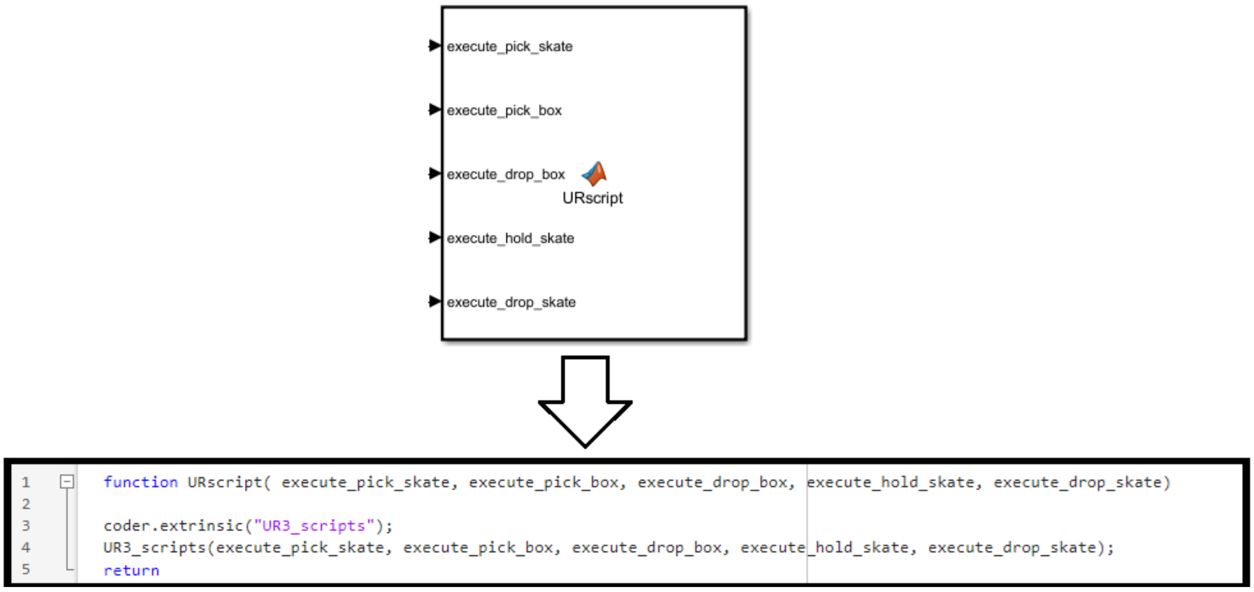
```
function MoveMIR(execute_moving_pos, stopfunc, AS_err, finish_pick_skate, finish_pick_box, finish_drop_box, finish_hold_skate, finish_drop_skate)
coder.extrinsic("MovingMIR");

MovingMIR(execute_moving_pos,stopfunc,AS_err,finish_pick_skate, finish_pick_box, finish_drop_box, finish_hold_skate,finish_drop_skate);

return
```

Figure 86: 'moveMIR' MATLAB function block calling the MATLAB function to move the MIR100.

This solution was opted due to the lack of essential packages to control the robot in Simulink.

5.2.6 UR Script-MATLAB function block

The 'URscript' block, driven by the FSM outputs, calls a function defined in MATLAB(Appendix1.4), which will execute the shell scripts defined in ROS corresponding to the tasks that the UR3 has to perform(Appendix2).



```
function URscript( execute_pick_skate, execute_pick_box, execute_drop_box, execute_hold_skate, execute_drop_skate)

coder.extrinsic("UR3_scripts");
UR3_scripts(execute_pick_skate, execute_pick_box, execute_drop_box, execute_hold_skate, execute_drop_skate);
return
```

Figure 87: 'URscript' MATLAB function block calling the MATLAB function to control the UR3.

the decision to control the UR3 on ROS instead of MATLAB/Simulink was taken because it was found to be more stable in the communication between the UR and the PC.

6 Tests

6.1 Preliminary tests

Preliminary tests were carried out by testing the two robots individually.
The first component of the MOMA that has been tested is the MIR100 and then the UR3.

- **MIR100**
  The first tests to move the MIR100 has been done in Simulink, by publishing the 'move_base/goal' topic in ROS:

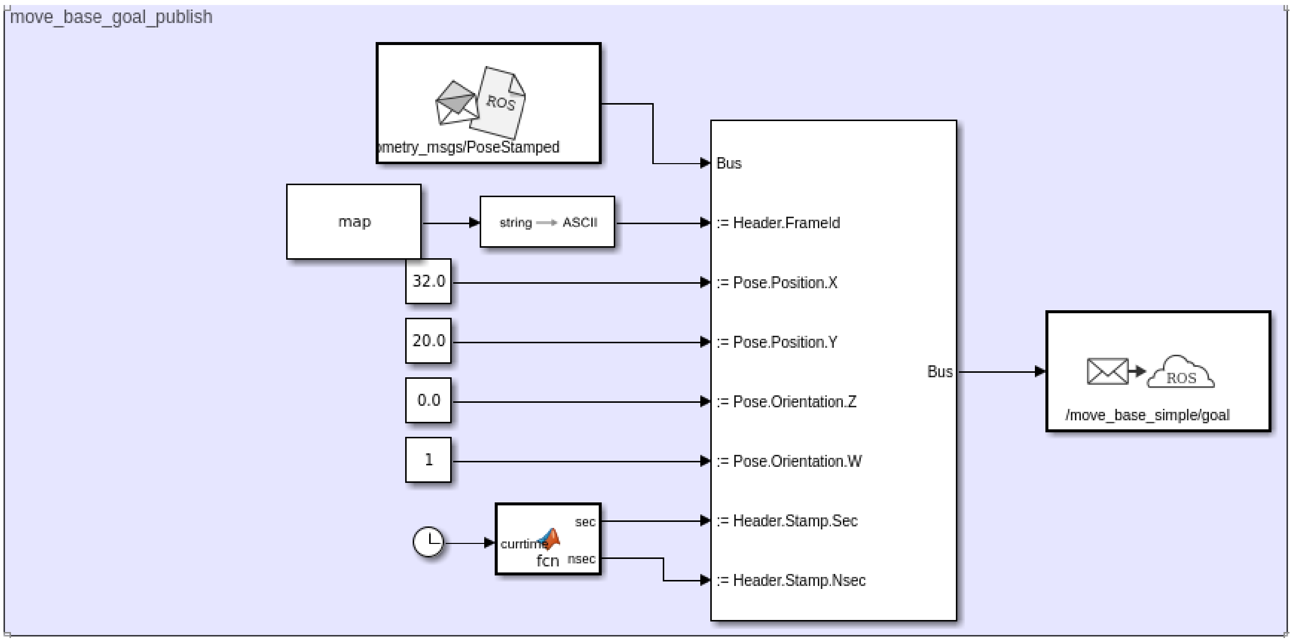


Figure 88: Publishing the 'move_base/goal' topic from Simulink to ROS.

  Even changing the way FrameId was written in the publish block, ROS wasn't able to read it because of a lack of packages. The same problem was encountered by publishing the same topic in MATLAB, so that led to the decision to move it with shell scripts, since MATLAB was able to run them.

- **UR3**
  The first UR3 tests has been executed in Simulink, using the structure seen previously (Cap.4.3.3) but instead of calling a ROS function, the UR3 configuration and the tasks were defined in a MATLAB scriptAppendix4.1.
  Since this method was too unstable, because led to frequent crashes, the ROS shell script has been used as for the MIR100.
  All the tests has been done keeping the UR3 and the Computer directly connected via Ethernet cable, because unlike the MIR100, is not possible to control the UR3 with ROS via Wi-Fi connection.

6.2 Final tests

Once both UR3 and MIR100 were tested successfully, they have been tested together, so the network was staged and the MOMA was tested in Simulink with the following prototype (Fig.(89)), where the tasks of the MOMA were executed with ROS shell scripts:

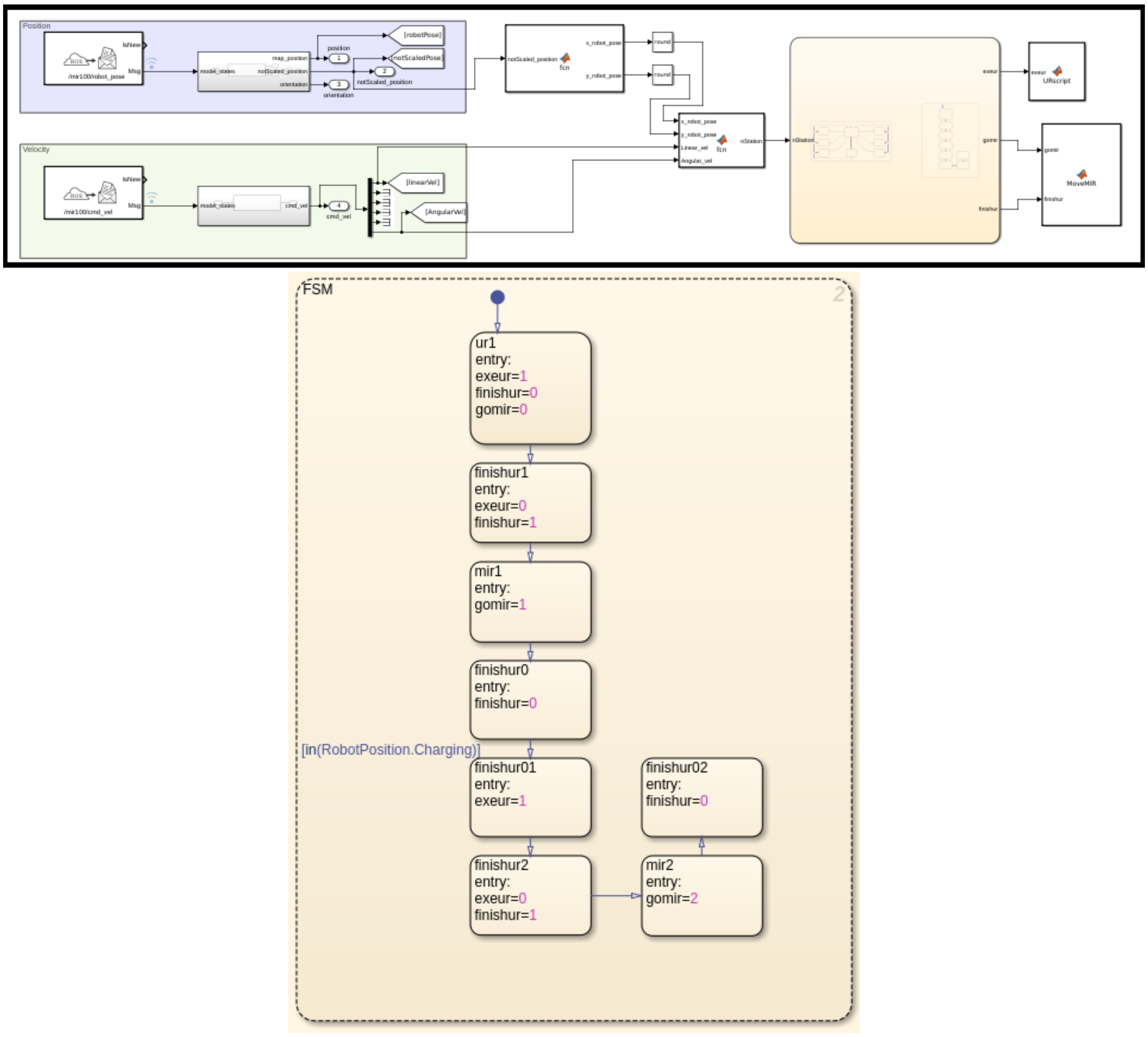
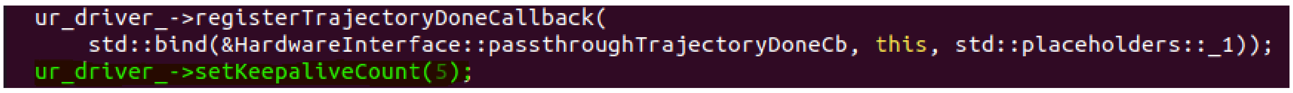


Figure 89: Test FSM of the entire system.

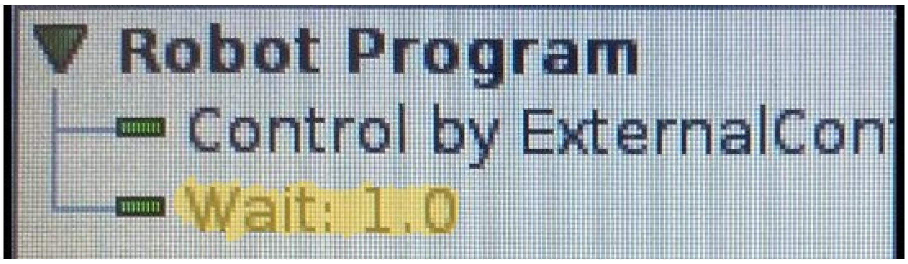
The latency problem was again relevant once the final network was set up, because connection between ROS and the UR3 wasn't stable enough without a direct cable connection between them.
The first thing that has been done to prevent crashes was to modify the 'hardware_interface.cpp' file, adding the following command:



```
ur_driver_->registerTrajectoryDoneCallback(
    std::bind(&HardwareInterface::passthroughTrajectoryDoneCb, this, std::placeholders::_1));
ur_driver_->setKeepaliveCount(5);
```

Figure 90: setKeepAliveCount added into the driver.

This command has been added because when the robot does not receive a command from the ROS machine every control cycle, it will end the program, so the robot stops, but now the program waits five control cycles without any feedback from the ROS machine.
The second adjustment to get less crashes was to add a Wait command into the Teach Pendant script, so that the reconnection to the external control program in the event of a crash was easier.



Robot Program

Control by ExternalCon

Wait: 1.0

Figure 91: Wait command added to the URcap program in the Teach Pendant.

After these arrangements, the MOMA has been tested with the complete FSM shown previously (Cap.5). During the simulation a visual check to the FSM has been done to debug any problem, because Simulink shows in real time all the activated signals and highlights the current state, so it was possible to check if every state was seen by the program, testing also all the non-ordinary conditions.
Has been noticed more stability in the connection between the UR3 and the Network, which led to several working cycles without crashes.

7 Conclusions

From the studies and tests conducted, the following conclusions were drawn.

- It was inferred that programming the robots in ROS permits to program them at an higher level, being able to take advantage of all the parameters describing the robots and their movements. Thanks to the numerous packages that ROS offers, it is also really easy to exploit external devices to increase the possible uses of the MOMA and its functionality.
- Despite the Universal Robots Toolboxes provided in MATLAB, has been decreed that to make the most of UR3's potential, it is best to use it on ROS through the shell/python scripts, because the connection between the two is more stable and the response to commands is faster.
- The UR3 has latency problems. Is not possible to control the Cobot with ROS through a Wi-Fi connection, because it hasn't been optimized for it. The only way to control it is via wired connection, and the best way to do it is using ROS as main operating system, because even the Virtual Machine increase the latency between the two, adding an extra layer in the communication.
- The MIR100 can't be controlled properly in MATLAB/Simulink due to lack of packages provided online, but it works perfectly in ROS.
- By tracking the MOMA and the operator during the cycle of work it was possible to plot different graphs. The first one in Fig.(92) is called 'Spaghetti Chart' and shows all the movements done in the working environment during the execution of the tasks and it represents ten cycles.

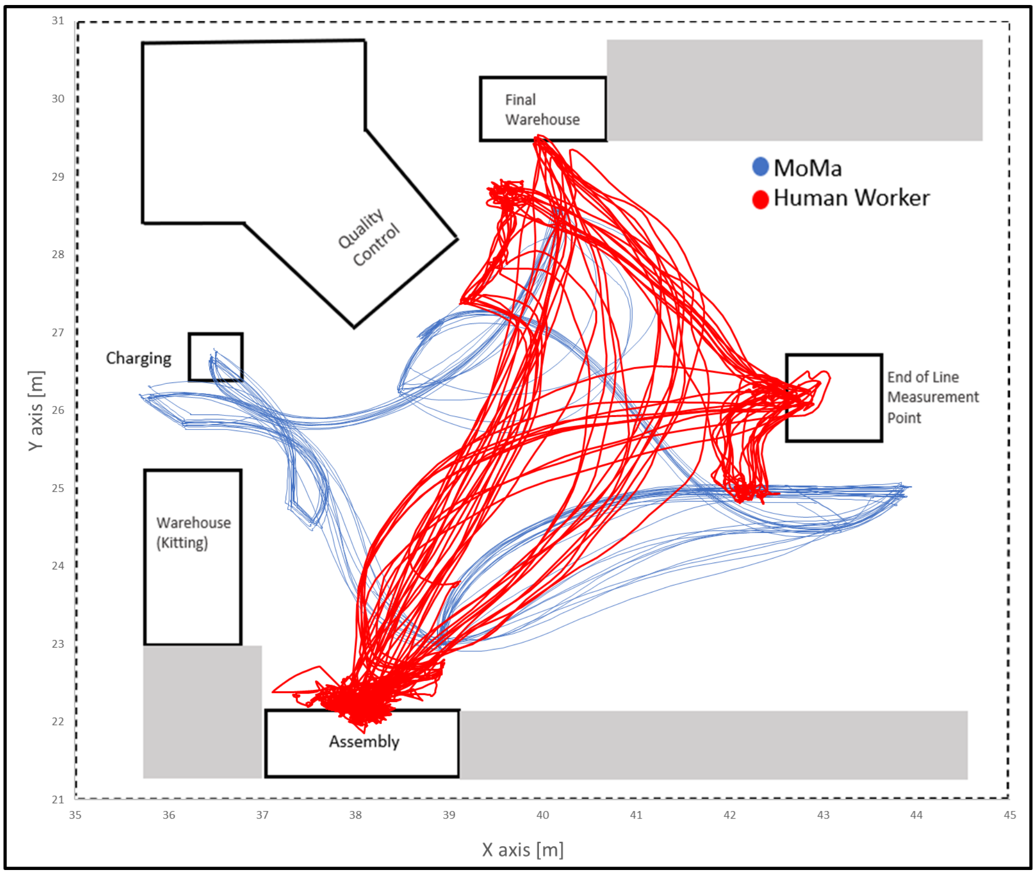


Figure 92: Spaghetti chart of the movements of the MOMA and the Human Worker in the working environment.

The next graphs are called 'Heat Map' graphs, and show the densities of occupation of the MOMA and Worker in the different stations:

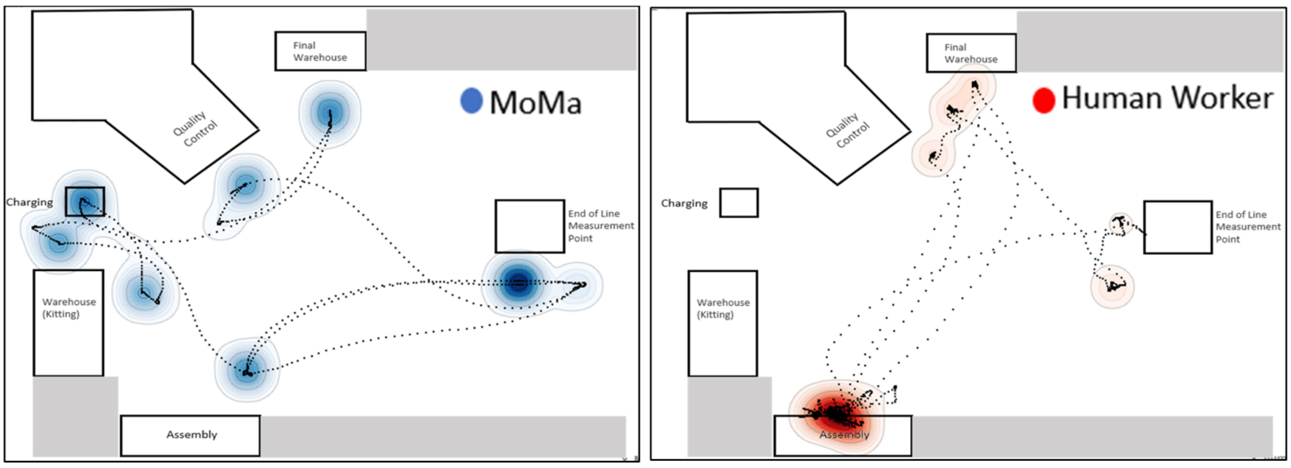


Figure 93: Heat map of the MOMA (Sx) and Human (Dx) for a successful cycle of work.

Thanks to these plots it is possible to:

- Identify useless movements.
- Optimize the workspace.
- Reducing the cycle time analyzing the tracks.
- Increase the Safety and reduce the collisions.
- Simulate improvement scenarios.
- Analyze non-standard situations in the line of production.

- Using the FSM is possible to handle different types of non-standard situations that can occur in the working environment, and it is possible, as cited before, to analyze the results of them in the line of production utilizing the above-mentioned Heat maps as shown in Fig.(94).

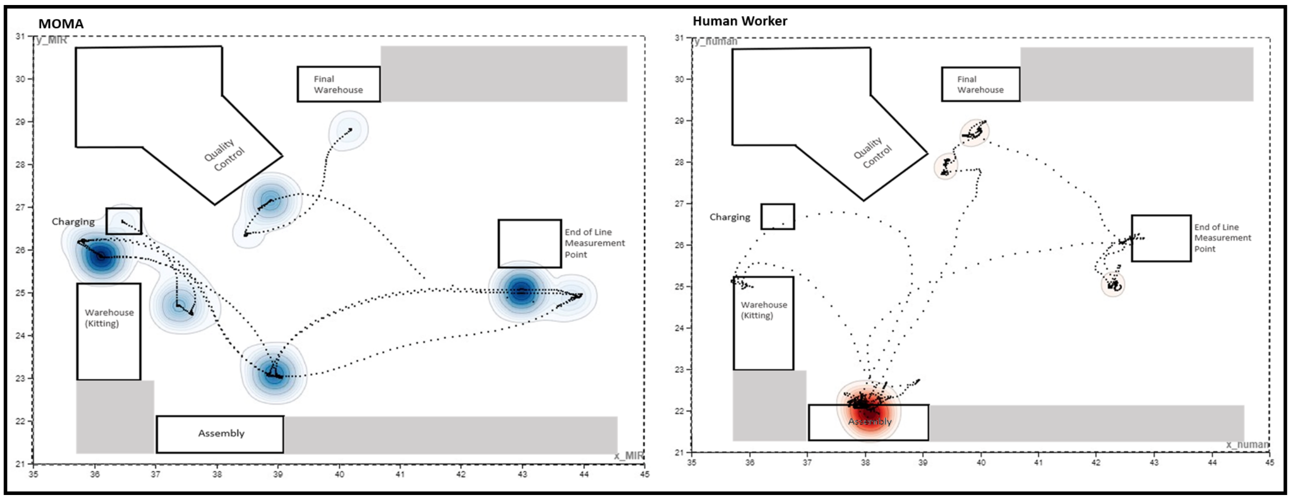


Figure 94: Heat map of a non-standard situation: the MOMA fail the 'Pick Box' task.

In conclusion this approach permits to constantly increase the efficiency of a working environment, by making the MOMA increasingly autonomous, by exploiting the data acquired in real time and adding devices to support it, in order to increase the productivity reducing the human effort.

A Appendix

A.1 MATLAB/Simulink scripts

A.1.1 FSM-Robot Pos MATLAB function

```
function nStation = fcn(x_robot_pose,y_robot_pose, Linear_vel, Angular_vel)

if (Linear_vel==0 && Angular_vel==0)
    if(x_robot_pose==30.0 && y_robot_pose==18.0)
            nStation=1; % Charging station

    elseif(x_robot_pose==32.0 && y_robot_pose==18.0)
            nStation=2; %Pick skate station

    elseif(x_robot_pose==31.0 && y_robot_pose==17.0)
            nStation=3; %Pick box station

    elseif(x_robot_pose==33.0 && (y_robot_pose==19.0 || y_robot_pose==20.0))
            nStation=4; %Assembly station

    elseif(x_robot_pose==32.0 && (y_robot_pose==22.0 || y_robot_pose==23.0))
            nStation=5; %EOL station

    elseif(x_robot_pose==30.0 && (y_robot_pose==19.0 || y_robot_pose==20.0))
            nStation=6; %Quality control station

    elseif(x_robot_pose==29.0 && (y_robot_pose==20.0 || y_robot_pose==21.0))
            nStation=7; %Final Warehouse station
    else
        nStation=0;
    end
else
    nStation=0;
end
```

A.1.2 FSM-Robot Battery percentage MATLAB function

```
function batterypercentage=GetBatteryPercentage

bp=rossubscriber('/MC/battery_percentage','std_msgs/Float64');
batterymsg=receive(bp,10);
batterydata=struct();
batterydata.percentage=batterymsg.Data;
perc=batterydata.percentage;
batterypercentage=perc;

return;
```

A.1.3 FSM-movingMIR MATLAB function

```
function succ=MovingMIR(Position,stopfunc,eol_err,finish_pick_skate,
   finish_pick_box, finish_drop_box, finish_hold_skate, finish_drop_skate)
if(Position==1 && stopfunc==0) %Moving to the charging position
system('sh /home/dmros/move_in_pos_files/CS.sh')
elseif (Position==2 && stopfunc==1) %Moving to the Pick skate position
system('sh /home/dmros/move_in_pos_files/WH_S.sh')
elseif (Position==3 && (finish_pick_skate==1 && stopfunc==2) ) %Moving to
   the Pick box position
```

```
system('sh /home/dmros/move_in_pos_files/WH_B.sh')
elseif (Position==4  && ((finish_pick_box==1 && stopfunc==3 ) || eol_err==1
    || (finish_drop_box==1 && stopfunc==3 ) || (finish_hold_skate==1 &&
    stopfunc==4 ))) %Moving to the Assembly position
system('sh /home/dmros/move_in_pos_files/AS.sh')
elseif (Position==5 && stopfunc==4) %Moving to the EOL station position
system('sh /home/dmros/move_in_pos_files/EOL.sh')
elseif ((Position==6 && stopfunc==5) || (Position==6  && finish_hold_skate
    ==1)) %Moving to the Quality control state position
system('sh /home/dmros/move_in_pos_files/QC.sh')
elseif (Position==7  && (finish_hold_skate==1 && stopfunc==6 )) %Moving to
    the Final warehouse position
system('sh /home/dmros/move_in_pos_files/FW.sh')
elseif(Position==8 && (finish_drop_skate==1 && stopfunc==7 )) %Moving to the
     charging position
system('sh /home/dmros/move_in_pos_files/CS.sh')
end
succ=1;
```

A.1.4 FSM-URscript MATLAB function

```
function  finish=UR3_scripts(execute_pick_skate, execute_pick_box,
    execute_drop_box, execute_hold_skate, execute_drop_skate)
finish=0;
if (execute_pick_skate==1)
  system('sh /home/dmros/move_in_pos_files/UR_moves/Complete_UR_scripts/PSK.
     sh');
  pause(2);
  finish=1;
elseif (execute_pick_box==1)
  system('sh /home/dmros/move_in_pos_files/UR_moves/Complete_UR_scripts/PBX.
     sh');
   pause(2);
   finish=2;
elseif (execute_drop_box==1)
  system('sh /home/dmros/move_in_pos_files/UR_moves/Complete_UR_scripts/DB.
     sh');
   pause(2);
   finish=3;
elseif (execute_hold_skate==1)
  system('sh /home/dmros/move_in_pos_files/UR_moves/Complete_UR_scripts/HSK.
     sh');
   pause(2);
   finish=4;
elseif (execute_drop_skate==1)
  system('sh /home/dmros/move_in_pos_files/UR_moves/Complete_UR_scripts/DSK.
     sh');
   pause(2);
   finish=5;
end
```

A.1.5 Motive-MATLAB connection

```
% From NatNetPollingSample library
clear
clc
freq = 180; %ms : take data every freq ms
durata_task= 600; %tempo di acquisizione=durata_task*100ms
% 1000/(freq) = "samples" , "samples" is samples taken in 1 second
% total acquisition time (sec) = durata_task / ("samples")


%Create table to save data of the rigid bodies:
rb_names = ["1_hat1","1_MIR100-2"];
%!!write the names of rigid bodies in alphabetic order (with respect to
    MOTIVE's order)

empty_table = table;
name_row = [];
coord_row = [];
for i=1:length(rb_names)
    name_row = cat(2, name_row, [rb_names(i),'','','','','','']);
    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];

[rigid1,rigid2,rigid3,data_Optitrack]=NatNetPollingSampleCustom_xyphii(
    length(rb_names), freq, empty_table,durata_task);
```

A.1.6 Motive-MATLAB coordinates conversion

```
%% 1 RIGID BODY (ROBOT)
%On motive you have to select just the rigid body that we want to measure
optitrack_data = load('C:\');
%return
scaling_factor = 1.297;
x_MIR = optitrack_data.data_Optitrack.intestazione3(3:end);
y_MIR = optitrack_data.data_Optitrack.intestazione1(3:end);
x_MIR = str2double(x_MIR)/1000;
y_MIR = str2double(y_MIR)/1000;

x_opt_offset = 30.9085;
y_opt_offset = 19.3607;
x_MIR = (-x_MIR + x_opt_offset).*scaling_factor;
y_MIR = (-y_MIR + y_opt_offset).*scaling_factor;
% smoothing:
x_MIR = smooth(x_MIR);
y_MIR = smooth(y_MIR);

table_opt = table(x_MIR,y_MIR);
writetable(table_opt,'C:\')
%% 2 RIGID BODIES (HAT-ROBOT)
optitrack_data = load('C:\');

scaling_factor = 1.297;
x_MIR = optitrack_data.data_Optitrack.intestazione10(3:end);
y_MIR = optitrack_data.data_Optitrack.intestazione8(3:end);
x_MIR = str2double(x_MIR)/1000;
y_MIR = str2double(y_MIR)/1000;
%Human
x_human = optitrack_data.data_Optitrack.intestazione3(3:end);
y_human = optitrack_data.data_Optitrack.intestazione1(3:end);
x_human = str2double(x_human)/1000;
y_human = str2double(y_human)/1000;

%In optitrack, data are wrt center of the area (30.838, 19.645)
%Move this center in the working station, s.t. all data are wrt to the
```

```
%starting point:
x_opt_offset = 30.9085;
y_opt_offset = 19.3607;
x_MIR = (-x_MIR + x_opt_offset).*scaling_factor;
y_MIR = (-y_MIR + y_opt_offset).*scaling_factor;
%Human
x_human = (-x_human + x_opt_offset).*scaling_factor;
y_human = (-y_human + y_opt_offset).*scaling_factor;
% smoothing MIR:
x_MIR = smooth(x_MIR);
y_MIR = smooth(y_MIR);
% smoothing Human
x_human = smooth(x_human);
y_human = smooth(y_human);

% table_opt = table(x_opt,y_opt,phi_opt,t_opt);
table_opt = table(x_human,y_human,x_MIR,y_MIR);
writetable(table_opt,'C:\')


%% Precision positioning robot

%On motive you have to select just the rigid body that we want to measure
optitrack_data = load('C:\');
%return
scaling_factor = 1.297;
x_MIR = optitrack_data.data_Optitrack.intestazione10(3:end);
y_MIR = optitrack_data.data_Optitrack.intestazione8(3:end);
x_MIR = str2double(x_MIR)/1000;
y_MIR = str2double(y_MIR)/1000;

x_human = optitrack_data.data_Optitrack.intestazione3(3:end);
y_human = optitrack_data.data_Optitrack.intestazione1(3:end);
x_human = str2double(x_human)/1000;
y_human = str2double(y_human)/1000;

x_opt_offset = 30.9085;
y_opt_offset = 19.3607;
x_MIR = (-x_MIR + x_opt_offset).*scaling_factor;
y_MIR = (-y_MIR + y_opt_offset).*scaling_factor;
x_human = (-x_human + x_opt_offset).*scaling_factor;
y_human = (-y_human + y_opt_offset).*scaling_factor;
% smoothing:
x_MIR = smooth(x_MIR);
y_MIR = smooth(y_MIR);
x_human = smooth(x_human);
y_human = smooth(y_human);
x_human_mean=mean(x_human);
y_human_mean=mean(y_human);
x_MIR_mean=mean(x_MIR);
y_MIR_mean=mean(y_MIR);
table_opt = table(x_human_mean,y_human_mean,x_MIR_mean,y_MIR_mean);
writetable(table_opt,'C:\')
%% COORDINATES EOL/FW/QC/RM+AS STATIONS

optitrack_data = load('C:\');

scaling_factor = 1.297;
%t4
x_opt_t4 = optitrack_data.data_Optitrack.intestazione24(3:end);
y_opt_t4 = optitrack_data.data_Optitrack.intestazione22(3:end);
```

```
x_opt_t4 = str2double(x_opt_t4)/1000;
y_opt_t4 = str2double(y_opt_t4)/1000;
%t3
x_opt_t3 = optitrack_data.data_Optitrack.intestazione17(3:end);
y_opt_t3 = optitrack_data.data_Optitrack.intestazione15(3:end);
x_opt_t3 = str2double(x_opt_t3)/1000;
y_opt_t3 = str2double(y_opt_t3)/1000;
%t2
x_MIR = optitrack_data.data_Optitrack.intestazione10(3:end);
y_MIR = optitrack_data.data_Optitrack.intestazione8(3:end);
x_MIR = str2double(x_MIR)/1000;
y_MIR = str2double(y_MIR)/1000;
%t1
x_human = optitrack_data.data_Optitrack.intestazione3(3:end);
y_human = optitrack_data.data_Optitrack.intestazione1(3:end);
x_human = str2double(x_human)/1000;
y_human = str2double(y_human)/1000;

x_opt_offset = 30.9085;
y_opt_offset = 19.3607;
x_opt_t4 = (-x_opt_t4 + x_opt_offset).*scaling_factor;
y_opt_t4 = (-y_opt_t4 + y_opt_offset).*scaling_factor;
x_opt_t3 = (-x_opt_t3 + x_opt_offset).*scaling_factor;
y_opt_t3 = (-y_opt_t3 + y_opt_offset).*scaling_factor;
x_MIR = (-x_MIR + x_opt_offset).*scaling_factor;
y_MIR = (-y_MIR + y_opt_offset).*scaling_factor;
x_human = (-x_human + x_opt_offset).*scaling_factor;
y_human = (-y_human + y_opt_offset).*scaling_factor;

% t4
x_opt_t4 = smooth(x_opt_t4);
y_opt_t4 = smooth(y_opt_t4);
% t3
x_opt_t3 = smooth(x_opt_t3);
y_opt_t3 = smooth(y_opt_t3);
% t2
x_MIR = smooth(x_MIR);
y_MIR = smooth(y_MIR);
% t1
x_human = smooth(x_human);
y_human = smooth(y_human);


table_opt = table(x_human,y_human,x_MIR,y_MIR,x_opt_t3,y_opt_t3,x_opt_t4,
    y_opt_t4);
writetable(table_opt,'C:\')
```

A.2 UR3 scripts

A.2.1 UR-Pick Skate

```
#!/bin/bash
sleep 3
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #rest position
    positions:[1.4258852005004883, -1.8647444883929651, -1.6875112692462366,
    -1.158304516469137, 4.768599033355713, -0.06314641634096319],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #near_pick position
    positions:[1.4324641227722168, -1.667170826588766, -1.0293996969806116,
    -1.9583905378924769, 4.77227783203125, -0.06315833726991826],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #move_pick position
    positions:[1.9724540710449219, -0.8914125601397913, -0.6777694861041468,
    -4.058575455342428, 5.415105819702148, -1.7229622046100062],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  },
   { #pick
    positions:[1.6749558448791504, -0.7677267233477991, -0.5935838858233851,
    -3.895484749470846, 5.331813335418701, -1.706062142048971],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 4}
  }
  ]
 }
}
}'
sleep 4

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[ { #go_away position
    positions:[1.0263161659240723, -0.741772476826803, -0.4537261168109339,
    -3.4058311621295374, 5.169745922088623, -1.6949833075152796],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
   {  #move_place position
    positions:[2.2203030586242676, -1.7592466513263147, -1.5882533232318323,
    -3.6169355551349085, 4.647643566131592, -1.5786836783038538],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  },
   {  #place position
    positions:[1.89410400390625, -1.18825608888735, -1.798060719166891,
    -3.030355755482809, 4.802003860473633, -1.7815106550799769],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 7}
  },
  ]
 }
}
}'
sleep 6

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[ {  #go_away_p position
    positions:[1.7429461479187012, -1.5067437330829065, -1.6389244238482874,
    -3.49024469057192, 4.614010810852051, -1.5785635153399866],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
},
{  #rest position
    positions:[1.4259095191955566, -1.86473256746401, -1.6876195112811487,
    -1.1582205931292933, 4.76869535446167, -0.06306249300111944],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  }
  ]
 }
}
}'
```

A.2.2 UR-Pick Box

```
#!/bin/bash
sleep 3
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #rest position
    positions:[1.4633111953735352, -2.1497429052936, -1.6837218443499964,
    -0.965468708668844, 4.753849506378174, -0.20865947404970342],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
{   #move_pick_bag position
    positions:[1.8253560066223145, -1.236744228993551, -3.0882864634143274,
    -3.6560805479632776, 4.727214813232422, -1.3915303389178675],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  },
{   #get_bag position
    positions:[1.3143529891967773, -0.8536008040057581, -3.157572333012716,
    -3.5399044195758265, 4.76446533203125, -1.5209577719317835],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.001, 0.001, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 4}
  }
  ]
 }
}
}'

sleep 4

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_up position
    positions:[0.524322509765625, -0.9939268271075647, -3.1571765581714075,
    -2.663018528615133, 4.875055313110352, -1.520705525075094],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
{ #move_place PART1 (not in the UR program)
    positions:[0.5243105888366699, -0.9939149061786097, -2.8816760222064417,
    -2.6629942099200647, 4.875067234039307, -1.5207174460040491],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 3}
  },
{ #move_place PART2 (not in the UR program)
    positions:[0.5243105888366699, -0.9939268271075647, -2.3976710478412073,
    -2.6629942099200647, 4.875030994415283, -1.5206816832171839],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 4}
  },
{   #move_place position
    positions:[0.34441328048706055, -1.1471198240863245, -2.376200501118795,
    -2.3670671621905726, 5.556877613067627, -1.5497186819659632],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  },
{   #drop_bag position
    positions:[2.2883987426757812, -1.8719432989703577, -1.9167488257037562,
    -3.4820006529437464, 4.920488357543945, -1.5497782866107386],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 7}
  }
  ]
 }
}
}'

sleep 7

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_down position
    positions:[2.5790584087371826, -1.8156960646258753, -2.0436366240130823,
    -3.8401725927936, 5.047478199005127, -1.567700211201803],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
{   #go_away position
    positions:[2.5790703296661377, -1.8157079855548304, -1.4466207663165491,
    -3.840160671864645, 5.047346115112305, -1.5678442160235804],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 4}
  },
{   #go_away2 position
    positions:[1.567549705505371, -1.8163917700396937, -1.4464409987079065,
    -2.8487351576434534, 5.187333583831787, -1.7409184614764612],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  },
{   #repositioning position
    positions:[1.8772883415222168, -2.4938390890704554, -1.901278320943014,
    -2.22577411333193, 4.89088249206543, -3.263024870549337],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 6}
  },
{   #rest position
    positions:[1.463179588317871, -2.149646584187643, -1.683770004902975,
    -0.9653967062579554, 4.753777980804443, -0.20862323442567998],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 8}
  }
  ]
 }
```

```
}
}'
```

A.2.3 UR-Drop Box

```
#!/bin/bash
sleep 3
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #rest position
    positions:[1.456312656402588, -1.8304055372821253, -1.6829193274127405,
    -1.3822601477252405, 4.753909587860107, -0.20870715776552373],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #rotate_tcp position
    positions:[1.460339069366455, -1.829493824635641, -1.6829193274127405,
    -1.3837421576129358, 4.718911647796631, -1.5761626402484339],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #back_low_tcp position
    positions:[2.222172737121582, -2.8105643431292933, -1.6829193274127405,
    -2.24892503419985, 4.718851566314697, -1.5754182974444788],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  },
   { #positioning_tcp position
    positions:[2.6229560375213623, -1.7300213019000452, -2.2252610365497034,
    -3.9115293661700647, 5.263833045959473, -1.5754182974444788],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 4}
  },
   { #get_bag position
    positions:[2.216299533843994, -1.318439785634176, -2.144779984151022,
    -3.943787399922506, 5.278737545013428, -1.5754063765155237],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  }
  ]
 }
}
}'
sleep 3

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

sleep 1

```

```
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[ { #go_up position
    positions:[1.628023624420166, -1.361995045338766, -2.144275967274801,
    -3.426528278981344, 5.327248573303223, -1.4919918219195765],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
   {  #move_place PART1 (not in UR program)
    positions:[1.6280474662780762, -1.3619831244098108, -2.4732866922961634,
    -3.426624123250143, 5.3272247314453125, -1.4920762220965784],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 2}
  },
   {  #move_place PART2 (not in UR program)
    positions:[1.6280837059020996, -1.362018887196676, -2.7685423533069056,
    -3.426636044179098, 5.327177047729492, -1.4920404593097132],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 3}
  },
   {  #move_place PART3 (not in UR program)
    positions:[1.6280717849731445, -1.361995045338766, -2.884878460560934,
    -3.4265640417682093, 5.327212810516357, -1.4920404593097132],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 4}
  },
   {  #move_place PART4 (not in UR program)
    positions:[1.6280717849731445, -1.3619831244098108, -3.0316274801837366,
    -3.426612202321188, 5.327248573303223, -1.4920523802386683],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 5}
  },
   {  #move_place PART5 (not in UR program)
    positions:[1.6280956268310547, -1.362006966267721, -3.1743386427508753,
    -3.4265998045550745, 5.327248573303223, -1.4920404593097132],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 6}
  },
   {  #move_place PART6 (not in UR program)
    positions:[1.6280593872070312, -1.3620427290545862, -3.3095324675189417,
    -3.4265998045550745, 5.327236652374268, -1.4920042196856897],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 7}
  },
   {  #move_place position
    positions:[0.8827190399169922, -0.3570674101458948, -3.1319425741778772,
    -3.6117146650897425, 4.6941328048706055, -1.5356743971454065],
    velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 9}
  },
   {  #drop_bag position
     positions:[0.5164270401000977, -0.04943687120546514, -3.1797717253314417,
     -3.5649998823748987, 4.683013916015625, -1.5271876494037073],
     velocities:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
     accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
     time_from_start:{secs: 10}
  },
  ]
 }
}
}'

sleep 9

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_low position
     positions:[0.6749053001403809, -0.06462985674013311, -3.1583996454821985,
     -3.749782387410299, 4.68302583694458, -1.5272234121905726],
     velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
     accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
     time_from_start:{secs: 1}
  },
  { #Go_back position
     positions:[1.6169977188110352, -0.5199864546405237, -3.1459150950061243,
     -4.086238686238424, 4.683169841766357, -1.5272591749774378],
     velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
     accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
     time_from_start:{secs: 2}
  },
  { #repositioning position
     positions:[1.563547134399414, -1.4042933622943323, -2.851785484944479,
     -3.1457112471209925, 6.059781551361084, -1.5273917357074183],
     velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
     accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
     time_from_start:{secs: 3}
  },
   { #rest position
     positions:[1.456336498260498, -1.8304179350482386, -1.682883087788717,
     -1.3822715918170374, 4.753933429718018, -0.20864755312074834],
     velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
     accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
     time_from_start:{secs: 4}
  }
  ]
 }
}
}'
```

A.2.4 UR-Hold Skate

```
#!/bin/bash
sleep 3
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #rest position
    positions:[1.4563965797424316, -1.830357853566305, -1.6829069296466272,
    -1.3822954336749476, 4.753789901733398, -0.20870715776552373],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #move_pick_skate position
    positions:[-0.4829595724688929, -0.6144579092608851, -1.8012507597552698,
    -0.7306163946734827, 4.734020233154297, -1.7416146437274378],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 3}
  },
  { #get_skate position
    positions:[-0.5429132620440882, -0.30494767824281865, -1.815221134816305,
    -0.7664230505572718, 4.684619903564453, -1.7641194502459925],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  }
  ]
 }
}
}'
sleep 4

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_up position
    positions:[0.37998485565185547, -1.0697100798236292, -1.8050764242755335,
    -0.9262631575213831, 4.760332107543945, -1.7517688910113733],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #holding position
    positions:[0.550382137298584, -1.4535864035235804, -1.804896656666891,
    -1.1672700087176722,4.867698669433594, -1.7520206610309046],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  }
  ]
 }
}
}'

sleep 6

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #drop_back position
    positions:[-0.4528396765338343, -0.3843620459185999, -1.8149455229388636,
    -0.7836197058307093, 4.690299034118652, -1.7519963423358362],
    velocities:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    accelerations:[0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
    time_from_start:{secs: 2}
  }
]
 }
}
}'

sleep 1

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_up position
    positions:[0.3800086975097656, -1.0697701613055628, -1.80504018465151,
    -0.926239315663473, 4.760332107543945, -1.751852814351217],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #rest position
    positions:[1.4563007354736328, -1.8304417769061487, -1.6829312483416956,
    -1.382200066243307, 4.753849506378174, -0.20859939256776983],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  }
  ]
 }
}
}'

```

A.2.5 UR-Drop Skate

```
#!/bin/bash
sleep 3
rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #rest position
    positions:[1.4563846588134766, -1.8304298559771937, -1.6828950087176722,
    -1.3822835127459925, 4.753825664520264, -0.2086113134967249],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #move_pick_skate position
    positions:[-0.48297149339784795, -0.6144221464740198, -1.8012388388263147,
    -0.7305801550494593, 4.734044075012207, -1.7415302435504358],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #get_skate position
    positions:[-0.48324710527528936, -0.3428877035724085, -1.8151491324054163,
    -0.7604315916644495, 4.685146808624268, -1.7519004980670374],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 3}
  }
  ]
 }
}
}'

sleep 2

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 1.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_up position
    positions:[-0.5970805327044886, -0.615021053944723, -1.815101448689596,
    -0.7604315916644495, 4.685182571411133, -1.7517927328692835],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #move_place position
    positions:[-0.5681698958026331, -0.7036932150470179, -4.826844994221823,
    -0.9166620413409632, 4.5825581550598145, -1.6390369574176233],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 5}
  },
  { #drop_bag position
    positions:[-0.5680258909808558, -0.35476762453188115, -4.828761402760641,
    -1.4781163374530237, 4.741940498352051, -1.6386283079730433],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 6}
  }
  ]
 }
}
}'

sleep 6

rosservice call /ur3/ur_hardware_interface/set_io "fun: 1
pin: 1
state: 0.0"

sleep 1

rostopic pub -1 /ur3/scaled_pos_joint_traj_controller/follow_joint_trajectory/goal
control_msgs/FollowJointTrajectoryActionGoal '{
goal: {
 trajectory: {
  joint_names: ["elbow_joint", "shoulder_lift_joint","shoulder_pan_joint", "wrist_1_joint",
  "wrist_2_joint", "wrist_3_joint"],
  points:[{ #go_low position
    positions:[-0.3553264776812952, -0.6490100065814417, -4.828581635151998,
    -1.1210663954364222, 4.829705238342285, -1.6387007872210901],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 1}
  },
  { #Waypoint1 position
    positions:[0.37001657485961914, -1.412281338368551, -4.822353188191549,
    -0.8893287817584437, 4.830316543579102, -1.6391566435443323],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 2}
  },
  { #rest position
    positions:[1.4564085006713867, -1.8304179350482386, -1.682978932057516,
    -1.3822835127459925, 4.753825664520264, -0.20868331590761358],
    velocities:[0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
    accelerations:[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    time_from_start:{secs: 4}
  }
  ]
 }}}'
```

A.3 MIR scripts

A.3.1 MIR-Moving to Charging Station

```
#!/bin/bash
echo "Moving to Charging Station"
ARRIVED_CS=0
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
```

```
{stamp: now, frame_id: "map"}, pose: {position: {x: 30.206781063839337, y: 17.604832063011,
z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.39302114997813575, w: 0.9195294316496149}}}'
ARRIVED_CS=1
```

A.3.2 MIR-Moving to Initial Warehouse in skate position

```
#!/bin/bash
echo "Moving to Pick Skateboard"
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 31.82, y: 18.194, z: 0.0},
orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}}'
```

A.3.3 MIR-Moving to Initial Warehouse in box position

```
#!/bin/bash
echo "Moving to Pick Box position"
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 30.873053428374384,
y: 17.226480213929726, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: -0.7130658154298102,
w: 0.7010970994558599}}}'
```

A.3.4 MIR-Moving to Assembly Station

```
#!/bin/bash
echo "Moving to Assembly"
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 33.21599952100866,
y: 19.402769129476916, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.9997689474464594,
w: -0.021495388384455815}}}'
```

A.3.5 MIR-Moving to EOL Station

```
#!/bin/bash
echo "moving to EOL station"
rostopic pub -1  /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 31.719087414982802,
y: 22.49061557540885, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.718537187607647,
w: 0.6954885405417498}}}'
```

A.3.6 MIR-Moving to Quality Control Station

```
#!/bin/bash
echo "Moving to Quality Control"
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 30.01859596035526,
y: 19.38327834619194, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: -0.39703516231431274,
w: 0.9178033993650532}}}'
```

A.3.7 MIR-Moving to Final Warehouse

```
#!/bin/bash
echo "Moving to Final Warehouse"
rostopic pub -1 /mir100/move_base_simple/goal geometry_msgs/PoseStamped '{header:
{stamp: now, frame_id: "map"}, pose: {position: {x: 28.871050383495483,
y: 20.373164163212923, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 1.0, w: 0.0}}}'
```

A.4 Test scripts

A.4.1 UR3-Pick Skate and Pick Box test

```
% The angles assigned in the movements correspond to [BASE SHOULDER ELBOW
    WRIST1 WRIST2 WRIST3]
% Inside the parenthesis put the same angles of the teach pendant
%% Prova lancio programma ros
finish=system('sh /home/dmros/move_in_pos_files/UR_moves/completeprog.sh');

%% LOAD UR3
clear;
ur3 = loadrobot('universalUR3');
%% Selecting the interface
Interface="UR3 Hardware";
%%  Connect to ROS host (PC-UR)
% Provide parameters of the host machine with ROS. Provide robot IP address
    in RobotAddress
deviceAddress = '192.168.0.101';
username = 'dmros';
password = 'Polito2021';
ROSFolder = '/opt/ros/noetic';
WorkSpaceFolder = '~/catkinUR_ws';
RobotAddress = '192.168.0.100';
% Connect to ROS device.
device = rosdevice(deviceAddress,username,password);
device.ROSFolder = ROSFolder;
%% Load robot RigidBodyTree model and initialize the universalrobot
    interface
%Initialize the communication interface using universalrobot API.
ur = universalrobot(device.DeviceAddress,'RigidBodyTree',ur3);

%% Pick Skate Waypoints
PS_WayP_rest=[-96.69 -106.85 81.7 -66.36 273.22 356.38]*pi/180;
PS_WayP_near_pick=[-58.99 -95.52 82.07 -112.21 273.43 356.38]*pi/180;
PS_WayP_move_pick=[-38.84 -51.08 113.01 -232.54 310.26 261.28]*pi/180;
PS_WayP_pick=[-34.01 -43.99 95.97 -223.19 305.49 262.25]*pi/180;
PS_WayP_go_away=[-26 -42.51 58.8 -195.14 296.2 262.89]*pi/180;
PS_WayP_move_place=[-91 -100.8 127.21 -207.24 266.29 269.55]*pi/180;
PS_WayP_place=[-103 -68.08 108.52 -173.63 275.13 257.92]*pi/180;
PS_WayP_go_away_p=[-93.9 -86.33 99.87 -199.98 264.36 269.55]*pi/180;

%% Pick Box Waypoints
PB_WayP_rest=[-96.47 -123.16 83.83 -55.32 272.37 348.05]*pi/180;
PB_WayP_move_pick_bag=[-176.94 -70.86 104.58 -209.48 270.85 280.27]*pi/180;
PB_WayP_get_bag=[-180.91 -48.91 75.31 -202.82 272.98 272.86]*pi/180;
PB_WayP_go_up=[-180.9 -56.95 30.04 -144.99 279.32 272.86]*pi/180;
PB_WayP_move_place=[-136.15 -65.72 19.74 -135.62 318.39 271.21]*pi/180;
PB_WayP_drop_bag=[-109.82 -107.25 131.12 -199.5 281.92 271.2]*pi/180;
PB_WayP_go_down=[-117.09 -104.03 147.77 -220.03 289.2 270.17]*pi/180;
PB_WayP_go_away=[-82.89 -104.03 147.77 -220.03 289.19 270.17]*pi/180;
PB_WayP_go_away2=[-82.87 -104.07 89.82 -163.22 297.21 260.25]*pi/180;
PB_WayP_repositioning=[-108.94 -142.89 107.56 -127.53 280.22 173.05]*pi/180;
%% Executing Pick Skate program
PSrest=sendJointConfigurationAndWait(ur,PS_WayP_rest,'EndTime',5);
waitfor(PSrest==1)
PSnear_pick=sendJointConfigurationAndWait(ur,PS_WayP_near_pick,'EndTime',5);
waitfor(PSnear_pick==1)
PSmove_pick=sendJointConfigurationAndWait(ur,PS_WayP_move_pick,'EndTime',5);
waitfor(PSmove_pick==1)
PSpick=sendJointConfigurationAndWait(ur,PS_WayP_pick,'EndTime',5);
```

```matlab
waitfor(PSpick==1)
PSgo_away=sendJointConfigurationAndWait(ur,PS_WayP_go_away,'EndTime',5);
waitfor(PSgo_away==1)
PSmove_place=sendJointConfigurationAndWait(ur,PS_WayP_move_place,'EndTime'
    ,5);
waitfor(PSmove_place==1)
PSplace=sendJointConfigurationAndWait(ur,PS_WayP_place,'EndTime',5);
waitfor(PSplace==1)
PSgo_away_p=sendJointConfigurationAndWait(ur,PS_WayP_go_away_p,'EndTime',5);
waitfor(PSgo_away_p==1)
PSrest=sendJointConfigurationAndWait(ur,PS_WayP_rest,'EndTime',5);
%% Executing Pick Box program
PBrest=sendJointConfigurationAndWait(ur,PB_WayP_rest,'EndTime',5);
waitfor(PBrest==1)
PBmove_pick_bag=sendJointConfigurationAndWait(ur,PB_WayP_move_pick_bag,'
    EndTime',5);
waitfor(PBmove_pick_bag==1)
PBget_bag=sendJointConfigurationAndWait(ur,PB_WayP_get_bag,'EndTime',5);
waitfor(PBget_bag==1)
PBgo_up=sendJointConfigurationAndWait(ur,PB_WayP_go_up,'EndTime',5);
waitfor(PBgo_up==1)
PBmove_place=sendJointConfigurationAndWait(ur,PB_WayP_move_place,'EndTime'
    ,5);
waitfor(PBmove_place==1)
PBdrop_bag=sendJointConfigurationAndWait(ur,PB_WayP_drop_bag,'EndTime',5);
waitfor(PBdrop_bag==1)
PBgo_down=sendJointConfigurationAndWait(ur,PB_WayP_go_down,'EndTime',5);
waitfor(PBgo_down==1)
PBgo_away=sendJointConfigurationAndWait(ur,PB_WayP_go_away,'EndTime',5);
waitfor(PBgo_away==1)
PBgo_away2=sendJointConfigurationAndWait(ur,PB_WayP_go_away2,'EndTime',5);
waitfor(PBgo_away2==1)
PBreporisioning=sendJointConfigurationAndWait(ur,PB_WayP_repositioning,'
    EndTime',5);
waitfor(PBreporisioning==1)
PBrest=sendJointConfigurationAndWait(ur,PB_WayP_rest,'EndTime',5);
```

References

[1] Stefan-Octavian Bezrucav, Malte Kaiser, and Burkhard Corves. Case study: Ai task planning setup for an industrial scenario with mobile manipulators. In *Proceedings of the Scheduling and Planning Applications Workshop of The Thirty-First International Conference on Automated Planning and Scheduling, Guangzhou, China*, pages 2–13, 2021.

[2] ZE Chebab, JC Fauroux, N Bouton, Y Mezouar, and L Sabourin. Autonomous collaborative mobile manipulators: State of the art. In *Symposium on Theory of Machines and Mechanisms/UMTS2015/TrISToMM*, 2015.

[3] Automation Distribution. Universal robots ur3 cb3 - light assembly table top precision robot arm. https://automationdistribution.com/universal-robots-ur3-light-assembly-table-top-precision-robot-arm/.

[4] Engineering.com. Mathworks introduces a robotic system toolbox, 2015. https://www.engineering.com/mathworks-introduces-a-robotic-system-toolbox/.

[5] Felix Exner. Universal robots ros driver, 2024. https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.

[6] Nooshin Ghodsian, Khaled Benfriha, Adel Olabi, Varun Gopinath, and Aurélien Arnou. Mobile manipulators in industry 4.0: A review of developments for industrial applications. *Sensors*, 23(19):8026, 2023.

[7] Martin Günther. mir_robot. https://github.com/DFKI-NI/mir_robot.

[8] Mukhammadrizo Karimov. *Mobile collaborative robot.* PhD thesis, Politecnico di Torino, 2024.

[9] Giuseppe La Spisa. *Design and evaluation of different digital control technologies of a mobile manipulator.* PhD thesis, Politecnico di Torino, 2023.

[10] Matthieu Herrb LAAS/CNRS. The mobile robots at laas, 1997. https://homepages.laas.fr/taix/robots/h2bis.html.

[11] Mathworks. rosinit. https://it.mathworks.com/help/ros/ref/rosinit.html.

[12] Mathworks. setenv. https://it.mathworks.com/help/matlab/ref/setenv.html.

[13] MathWorks. Robotics system toolbox, 1994-2024. https://it.mathworks.com/products/robotics.html.

[14] MathWorks. Ros toolbox, 1994-2024. https://matlab1.com/matlab-consulting-services/.

[15] MathWorks. Matlab® urcap for external control, 2023. https://github.com/mathworks/MATLAB-URCap-for-External-Control/tree/main.

[16] MathWorks. Robotics system toolbox™ support package for universal robots ur series manipulators, 2024. https://it.mathworks.com/matlabcentral/fileexchange/117530-robotics-system-toolbox-support-package-for-universal-robots-ur-series-manipulators.

[17] MathWorks. Simulink, 2024. https://it.mathworks.com/products/simulink.html.

[18] Giovanni Mazzeo and Mariacarla Staffa. Tros: Protecting humanoids ros from privileged attackers. *International Journal of Social Robotics*, 12, 07 2020.

[19] Nutai. What is a urcap?, 2023. https://nutai.com/en/what-is-a-urcap/.

[20] Optitrack. Natnet sdk, 2024. https://optitrack.com/software/natnet-sdk/.

[21] Ravi Raj and Andrzej Kos. A comprehensive study of mobile robot: history, developments, applications, and future research perspectives. *Applied Sciences*, 12(14):6951, 2022.

[22] Universal Robot. Urcap-basics, 2022. https://www.universal-robots.com/articles/ur/urplus-resources/urcap-basics/: :text=A%20URCap

[23] robotnik. Mobile manipulators: the intelligent production for your factory, 2022. https://robotnik.eu/mobile-manipulators-the-intelligent-production-for-your-factory.

[24] ROS.org. Ros-robot operating system. https://wiki.ros.org.

[25] ROS.org. Services, 2011. https://wiki.ros.org/rosservice.

[26] ROS.org. Ubuntu install of ros noetic, 2021. https://wiki.ros.org/noetic/Installation/Ubuntu.

[27] TechSurce. Stateflow for logic driven system modeling, 2024. https://www.techsource-asia.com/our_courses/stateflow-for-logic-driven-modeling/.

[28] UniversalRobots. Matlab for universal robots, 2024. https://www.universal-robots.com/plus/products/mathworks/matlab-for-universal-robots/.

[29] Wikipedia. Mobile manipulator, 2024. https://en.wikipedia.org/wiki/Mobile_manipulator.

[30] Wikipedia. Robot operating system, 2024. https://en.wikipedia.org/wiki/Robot_Operating_System.

[31] Manman Yang, Erfu Yang, Remi Christophe Zante, Mark Post, and Xuefeng Liu. Collaborative mobile industrial manipulator: a review of system architecture and applications. In *2019 25th international conference on automation and computing (ICAC)*, pages 1–6. IEEE, 2019.