Politecnico Di Torino

Master's Degree in Mechatronic Engineering

Master Thesis

Design and fidelity evaluation of a Digital Twin of a Mobile Manipulator

Tutor

Prof. ANTONELLI Dario

Candidate

SORIANO Marco,
277607

Contents

1 - Introduction

1.1 - Overview on the digitalization phases

Nowadays, industrial automation, particularly Manufacturing 4.0 field, is eagerly heading toward the digitalization of the Product Life Cycle through the design of Digital Twins (DT).
The main benefit deriving from this transition is the possibility of analyzing different aspects of the production process such as:

- Potential system breakdowns due to harmful maneuvers or risks for operators health;
- Prediction of flaws both in the final product and in the processing stages;
- Quality study through virtual processing with the opportunity to optimize the procedures before performing them on the real system.

The applications are potentially endless and for this reason the definition of Digital Twin is easily subject to misconceptions.
However, in this paragraph we will try to delineate the concepts of Digital Model (DM), Digital Shadow (DS) and Digital Twin (DT) based on the available documentation[1] about the topic in order to better contextualize this thesis objective.

First of all, the three definitions refer to the stages marking the digitalization of a physical system.

The DM is a 3D virtual representation and it cannot send or receive data from the physical object. Therefore, we can see it as a static model that can be modified manually, a CAD.

On the other hand, the DS can communicate with the real system receiving and elaborating information both in real-time and previously acquired, while the data flow in the way back to the physical object is not available.

Finally, a DT is the ultimate connection between the virtual and the real world enabling the data exchange in both directions.

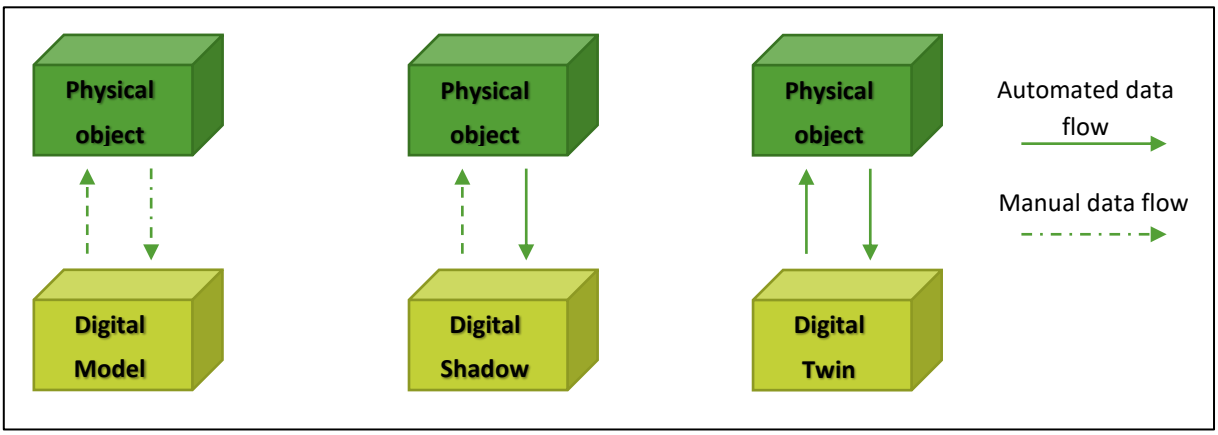

Figure 1. 1 - Digitalization steps

[1] 'A virtual commissioning based methodology to integrate digital twins into manufacturing systems' - G.Barbieri, A.Bertuzzi, A.Capriotti, L.Ragazzini, D.Gutierrez, E.Negri, L.Fumagalli .

While the DT of a physical object can be only one, multiple DS can exist depending on the detail level and the specific application. Indeed, different applications mean processing different set of data, from which it follows a different acquisition method and depending on it the required level of detail of the DS changes.

Therefore, if a DT is required to have the level of detail that is mandatory to be a complete two-way real-time communication between real and virtual worlds, the DS is expected to include just the features related to its specific application allowing a real-time data thread from the physical object.

Before proceeding, it might be important to make some considerations about the real-time feature that in most of the literature is described as a mandatory requirement.

To contextualize our choice of focusing more on data exchanged not in real-time during the analysis developed in this thesis, we can take as support an important clarification about this feature in *"Digital Twin: Generalization, characterization and implementation"*[2].

In the mentioned paper it is stated that even if in most of the literature the real-time data exchange between real and virtual worlds is perceived as a key-feature for both ideal DT and DS, having the physical object and the virtual representation working in synchronous is not practical or even required in some real case scenarios.

Since we will study an application regarding the industrial production field, to achieve the advantages of the digitalization, working with previously collected data or simulating scenarios in advance results more efficient than having both the physical and virtual object moving at the same time.

1.2 - Objective of this thesis

Now that the definitions of DT, DS and DM have been explained, we can proceed in describing the goal of this project.

In the laboratory in which this thesis has been developed, Mid4Lab of Politecnico di Torino, a Mobile Manipulator (MoMa) has been prototyped assembling a mobile robot MiR100 and a robotic manipulator UR3 CB3 series.

In a MoMa, the workspace is not fixed since the UR3 base is constantly moving along with the mobile robot.

This feature has the advantages of making the mobile robot fully autonomous being able to pick and delivering components across the production site without the need for the operators to place and collect objects on it.

On the other hand, allowing the manipulator to move in the space makes it able to execute tasks in different locations enhancing both the autonomous and collaborative sides.

[2] "Digital Twin: Generalization, characterization and implementation", Eric VanDerHorn, Sankaran Mahadevan, Decision Support Systems (2021).

Unfortunately, while the manipulator has a high precision when executing its missions, the mobile robot is not so reliable when reaching the requested positions.

The UR3 is usually mounted on fixed structures since to execute its tasks it needs highly precise reference systems for its own position and for the objects to be manipulated in the working space and this requirement is rarely delivered by the mobile robot.

Therefore, the Mobile Manipulators are scarcely used in the companies since a possible pose error of the mobile robot might put the manipulator in the wrong working position.

In some cases, this can simply result in a failed execution, but in a worst case scenario, a manipulator moving in the wrong location might cause damages to surrounding structures and itself or even hurt nearby operators.

From these premises, a powerful tool is needed to:

- Study different tasks minimizing the risks;
- Optimize those tasks without having to constantly move the robot during the testing process in order to foresee and eliminate potentially dangerous motions when possible.

For our purposes, designing a virtual robot that behaves as the real one would be an efficient asset for the future colleagues that will need to perform mission evaluations spacing from statistical studies to entire processes optimization in complete safety. Indeed, the workflow that we will follow to generate our model, can be applied not only to this specific system, but to all the laboratory robots and working cells.

Throughout this work, the first goal was to generate a Digital Twin able to execute different tasks in the simulation environment or to replicate the movements of the real-world robot.

If the latter feature is useful to calibrate the model and to execute troubleshooting analysis, the former one is functional in the perspective of optimizing various tasks predicting and analyzing the possible outcomes of the robot actions in terms of production quality and even avoiding the risks of damage, to both humans and robot, whenever a potentially harmful mission is designed.

To achieve this result, different control algorithms have been designed to generate the motion of the robot in response to different type of inputs.

Given a set of data for both the mobile robot and the manipulator, our system is able to move like the physical object and automatically execute different tasks.

The controlling algorithm design has been essential in the first part of the project in which a running-in phase of the system had to be performed and various tasks had to be studied directly in the virtual reality.

Therefore, this model has been designed both to receive signals directly from the physical object and to run on previously acquired or designed data.

It is important to underline a relevant obstacle encountered in the development of this thesis:

in case of the digitalization of commercial robots, those features have to be taken in consideration keeping in mind the limitations of the model deriving from the fact that it is not possible to acknowledge neither the real algorithms used to control the robots nor its specific mechanical details[3].

Moreover, the followed design method results proficient if we want to generate a DT of a customized robot of which we know with extreme precision all its mechanical parameters and, being open-systems, can be programmed with our own codes.

Considering that we developed our objective to be exploited in the industrial field, commercial robots like the available MoMa had to be used.

For this kind of robots, the producer does not provide the full specifications about the 3D design and the exact algorithms of the systems for copyright reasons. Thus, everything that could not be granted by the datasheets has been calibrated at its best through multiple analysis in order to deliver a level of detail that could be as precise as possible.

Furthermore, like all the industrial robots, our MoMa is not an open-system, so a data flow from our virtual model to the real object cannot be performed. In any case, the model we created can be easily modified in case of the application on a robot that allows this communication type.

Finally, for these reasons, we designed work-flow able to produce a model with the potentiality of working as Digital Twin, but for the physical object of this thesis it can be used as a Digital Shadow and in the following pages we will address it as such.

[3] The robot shown in *Figure 1.2,* has not been created by us, but is composed of two original robots from different companies.

1.3 – Brief description of the robot

The mobile manipulator is the assembly of two different robots:

- A mobile robot, model MiR100 by Mobile Industrial Robots, with two motor wheels and four caster wheels;
- A robotic arm, model UR3 by Universal Robots, with 6 degrees of freedom with a gripper RG2 by OnRobot mounted on.

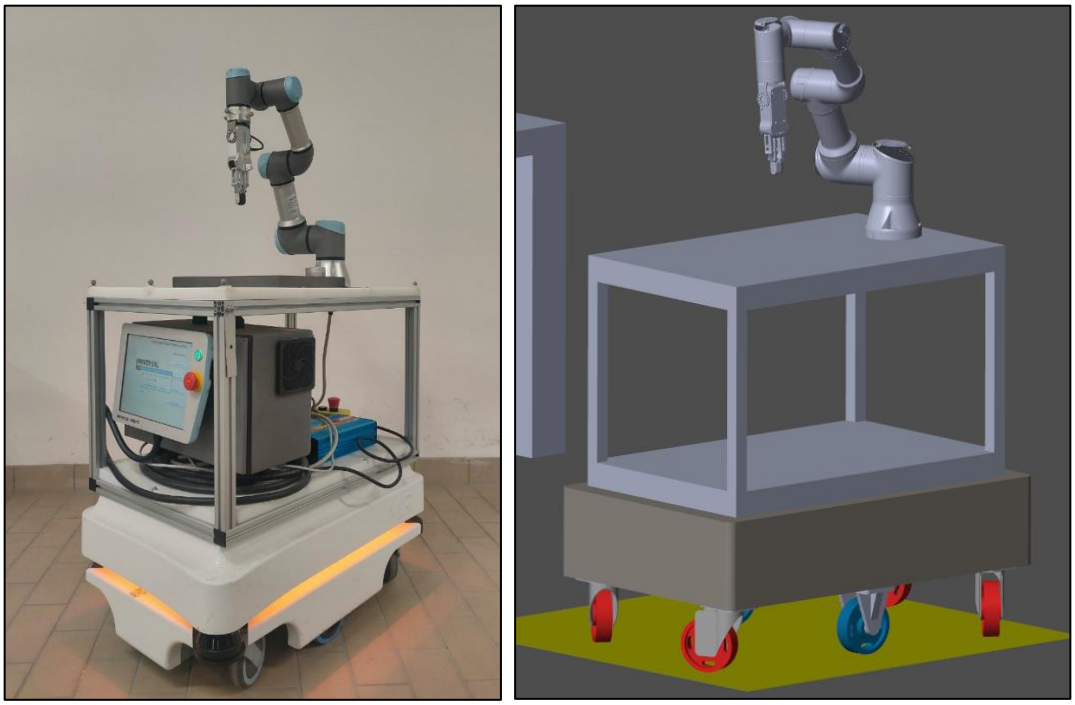

Figure 1. 2 – Mobile Manipulator and Digital Twin

1.3.1 – MiR100

The mobile robot allows the manipulator to execute its tasks in different parts of the laboratory or directly onto its own chassis. The motion is generated by only two motor wheels (in blue in both pictures of *figure 1.2*) while the others are caster wheels for support and dexterity.

The main features of interest for our study on this robot are:

- Collaborative mode: the robot is able to move in dynamic environments ;
- The robot plans its motion from the starting point to the destination finding the most efficient path. Whenever it encounters an obstacle not present in the map it is able to adjust its trajectory. Since we are going to analyse only static environment cases, this last feature has not been implemented;
- Automated transportation of loads up to $100\ kg$;
- Internal map: this robot can base its navigation on a given CAD of the surrounding or can generate its own map manually navigating around its workspace.

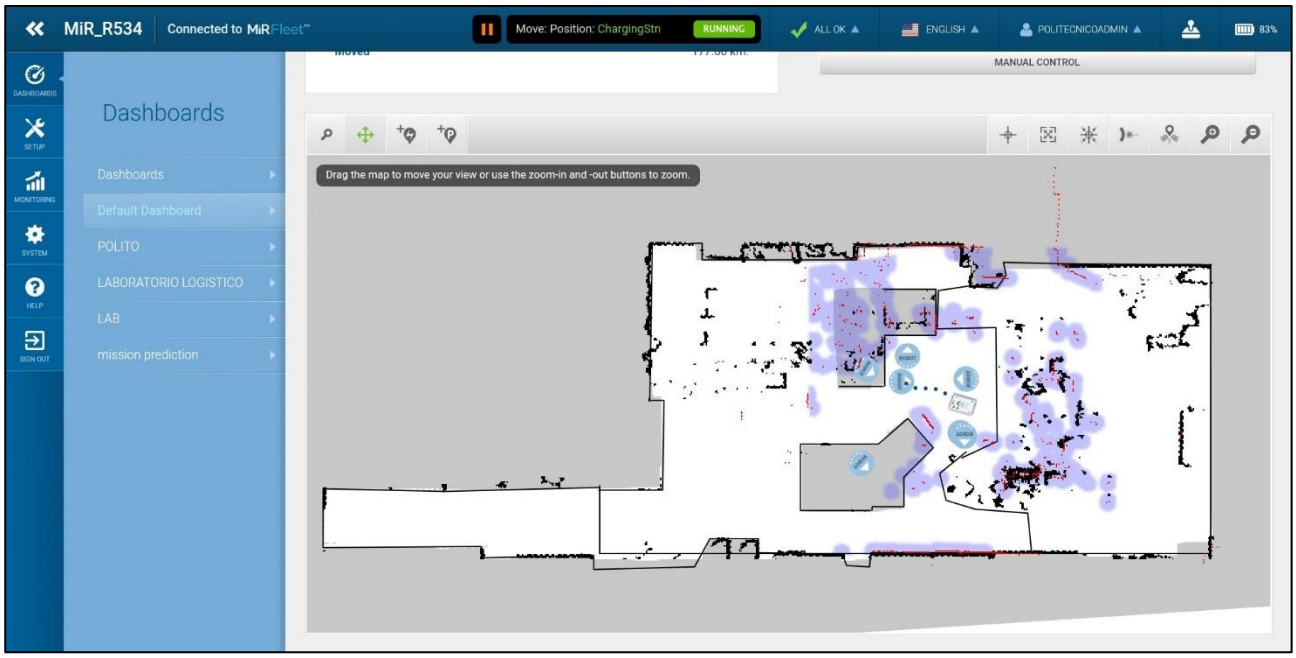

Figure 1. 3 - MiR100 live map

The whole mission can be designed on its software alternating MiR100 runs to UR3 sequences.

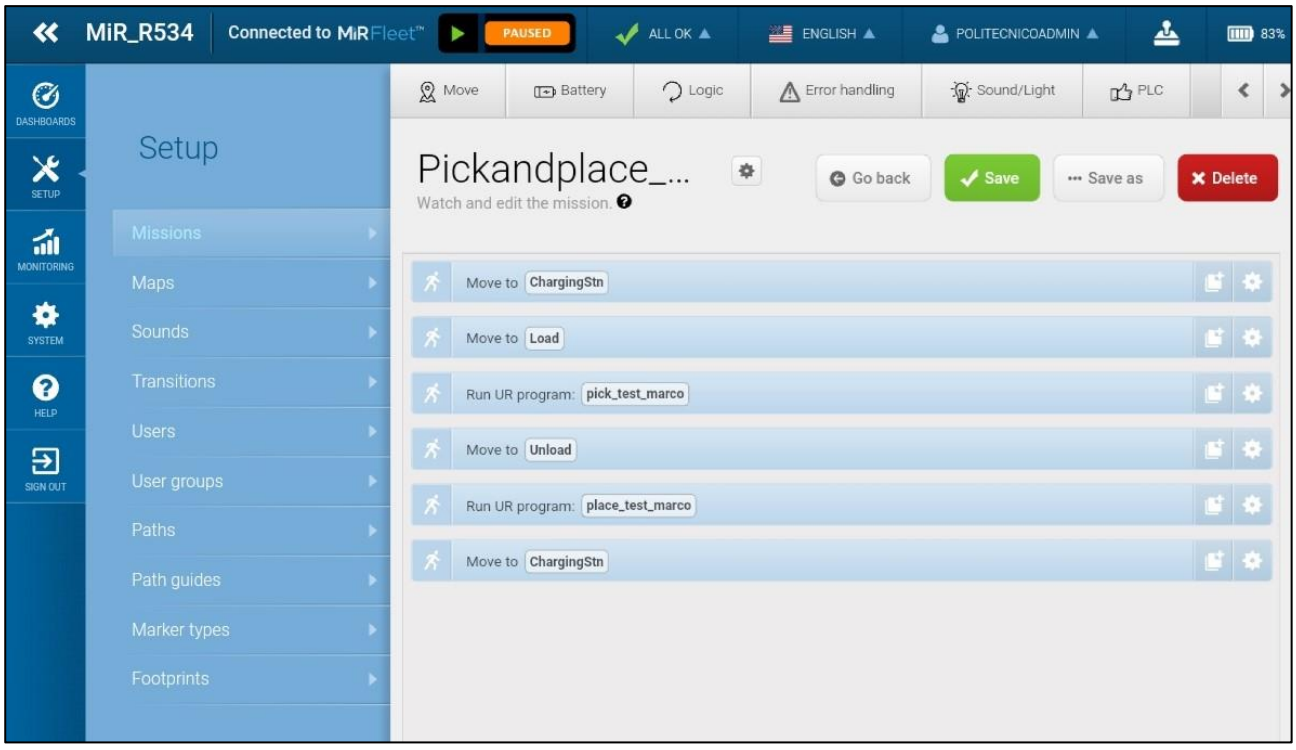

Figure 1. 4 - Example of scheduled mission with UR3 actions

As we can see, the task execution is sequential and the two robots cannot move simultaneously.

1.3.2 – UR3

The manipulator is mounted on the mobile robot and features a gripper RG2 by OnRobot.

Main features that worth mentioning:

- 500 mm reach;

- Maximum payload $3\ kg$;
- 6 degrees of freedom;
- All joints can rotate in a $\pm 360°$ range;
- Force and power limiting : reduced clamping forces for collaborative mode;
- Momentum limiting: joint speed reduction or immediate stop in case of collision between robot and operator. Once again, since we are simulating just the robot, this feature will be noticeable just in the torque saturation inside the control part of the model;

The UR3 uses PolyScope graphical user interface to design its task. Once saved, those file are compatible with MiR100 software to combine the actions.

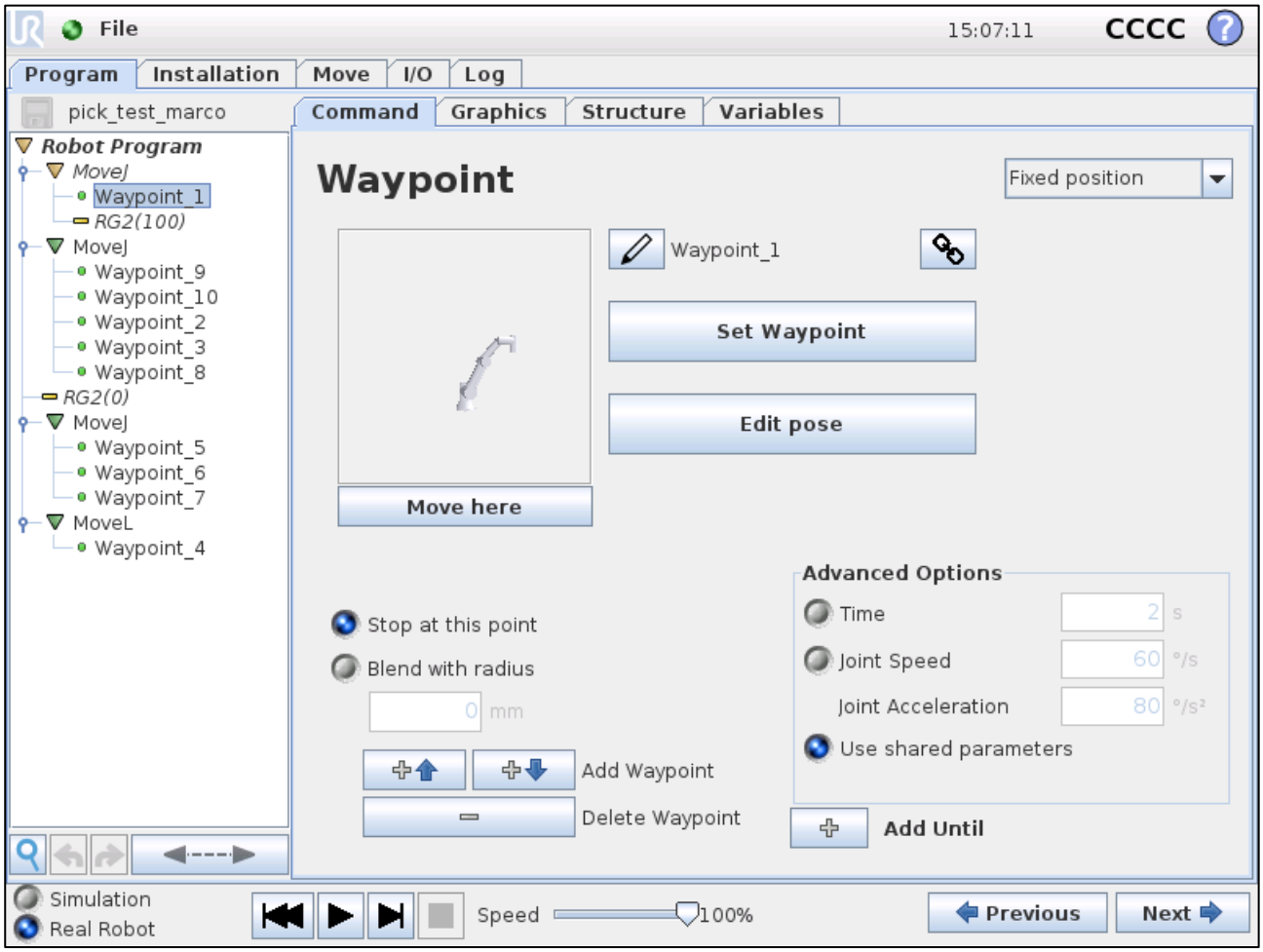

Figure 1. 5 - Example of UR3 task schedule

1.4 – Software and development tools

To develop the simulation system, the following software have been used:

- Matlab & Simulink;
- Simscape Multibody;
- SolidWorks;
- OptiTrack – Motive;
- Node-red.

In the continuation of this document, we will explain more in detail how those are used.

1.5 – Workflow

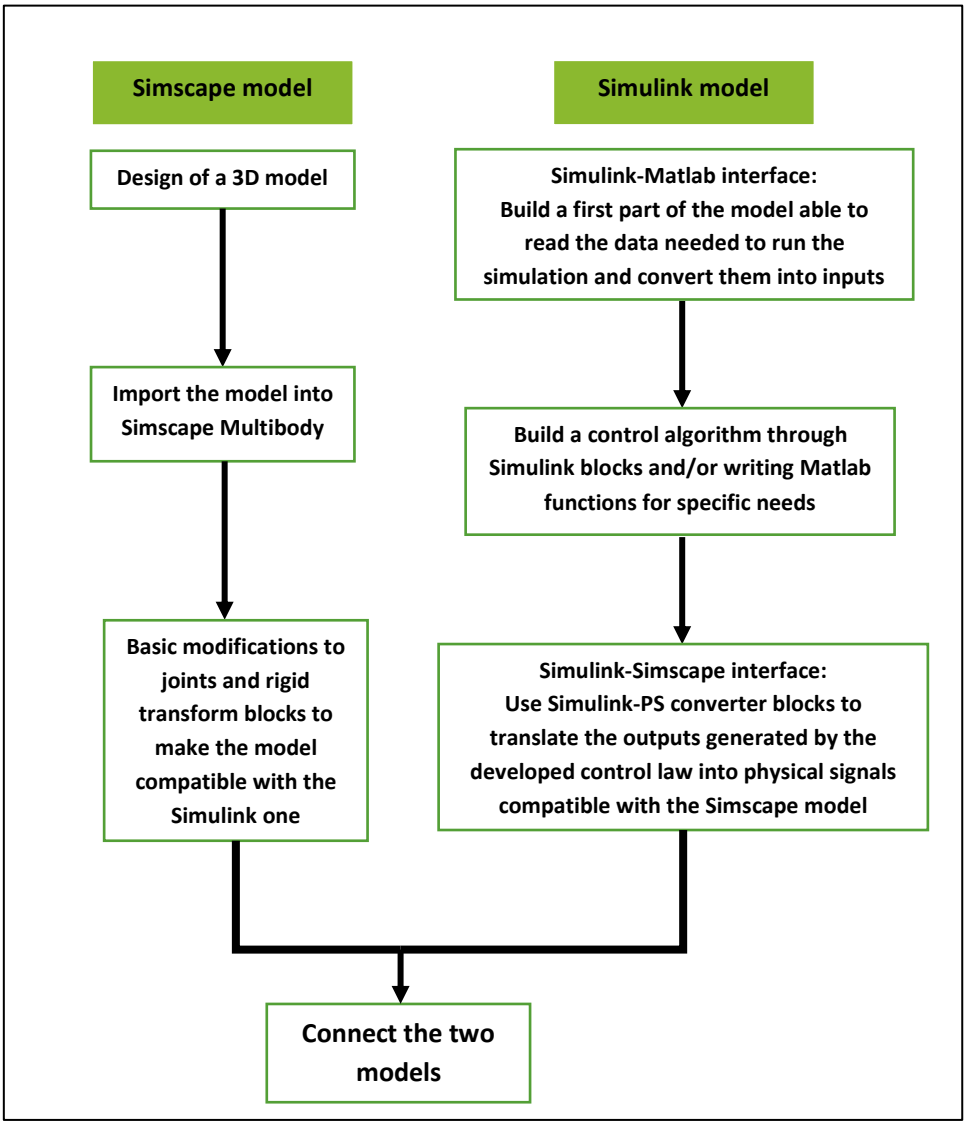

Figure 1. 6 -Workflow to build a Simulink-Simscape model

The realization of the DS has been carried out as the implementation of two separate prototypes, one for MiR100 and one for UR3 with an RG2 gripper mounted on. Finally, those two models have been assembled as a single one.

The project workflow started from the realization of a Simulink model to control a simple dot sketched robot moving from a point to another on an imported map of the laboratory.
For this step, we took inspiration from MathWorks case study *'Execute Tasks for a Warehouse Robot*[4]*'*.

Once the Simulink control logic has been completed, we designed a 3D model of the whole mobile manipulator removing all the unnecessary parts for the simulation and imported it as a Simscape Multibody model.

[4] Page link: https://it.mathworks.com/help/robotics/ug/execute-tasks-for-a-warehouse-robot.html

Thereafter, Simulink and Simscape models have been modified to communicate with each other through inputs and outputs.

After the MiR100 has been completed, with the same method we implemented the UR3 one. For this step, the MathWorks documentation exploited is the case study *'Model and Control a Manipulator Arm with Robotics and Simscape*[5]*'.*

For MiR100, UR3 and RG2, the function blocks written to meet the behavior requirements are not the official algorithm contained in the real robots but have been developed through a reverse-engineering process observing the real robots motion during the execution of various tasks.

[5] Page link: https://it.mathworks.com/help/robotics/ug/model-and-control-a-manipulator-arm-with-simscape.html

2 – Data acquisition

As previously described in the introduction chapter, there are two ways of using the Digital Shadow and depending on which one we choose, we have two types of data that will be used as input for our model:

1) Troubleshooting: data collected from the robot control-box;
2) Design of a new task: data written from scratch that will be given as reference at first to the simulation and then to the real robot.

The inputs needed to run the model are the poses of both MiR100 and UR3.
In the following paragraphs, it will be explained how those information are transformed into physical inputs able to move the DS in the virtual environment.

2.1 – Environment mapping

Before entering in the detail of the data collection, it is important to generate a map of the laboratory as a reference system to read the various poses.

MiR100, thanks to its sensors is able to depict a map of its surrounding specifying the areas in which it cannot enter.

Acquiring the .png image file generated by the robot and processing it inside a Matlab script[6] named 'binaryoccupancygrid.m' we obtain an approximation of the map in which it is possible to mark the various positions of interest depending on the task.

```
image = imread('new_lab_4.png');

grayimage = im2gray(image);
bwimage = grayimage < 9;
scaledimage = imresize(bwimage, 0.06);
grid = binaryOccupancyMap(scaledimage);

mapMatrix = grid.occupancyMatrix > 0.5;

fig = figure("Name","scaledMap");
set(fig, "Visible", "on");
ax = axes(fig);
scaledmap=binaryOccupancyMap(mapMatrix);
show(scaledmap,"Parent",ax);
hold on;
```

With command *im2gray* we simply convert the RGB map in grayscale and with *imresize* we scale it to process it with *binaryOccupancyMap*.

This last command processes the image translating it into a grid in which each cell contains the value 0 in case the related coordinate is a free space or 1 if it is occupied by something.

[6] All the codes that will be mentioned in this document are contained inside the project directory.

In our analysis, since the path is the one of the real robot this feature will not be important, but in case of other simulated tasks, knowing if a cell is occupied or not will be important to trace obstacle-free paths.

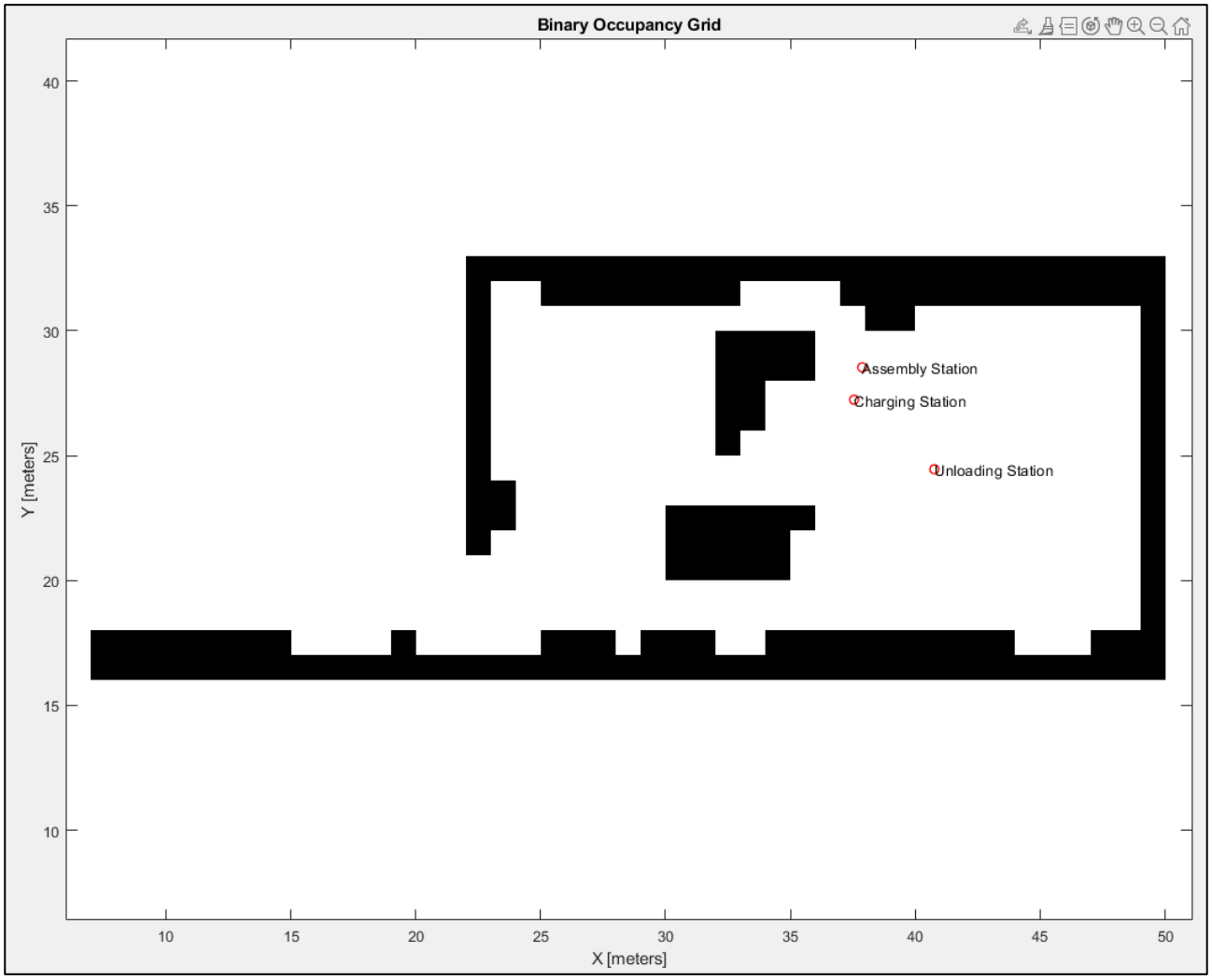

Figure 2. 1 - Scaled map of the Laboratory

Once the simulation has terminated, the simulated itinerary is traced on the map to compare it with the ones obtained through the control-box and the cameras data.

2.2 – Data from real to virtual robot

To obtain these data we have to implement a task for the mobile robot and run it.
As the MoMa executes it, we acquire information on both robots in two ways:

1) Poses from control-boxes through Node-Red development tool;
2) Poses from OptiTrack cameras.

2.2.1 – Control-box data

To extrapolate robots positioning and orientations, the following Node-Red flows have been implemented.
The one shown in *Figure 2.2* registers the pose of MiR100 as $[x; y; \phi]$ with a polling rate of 100ms.

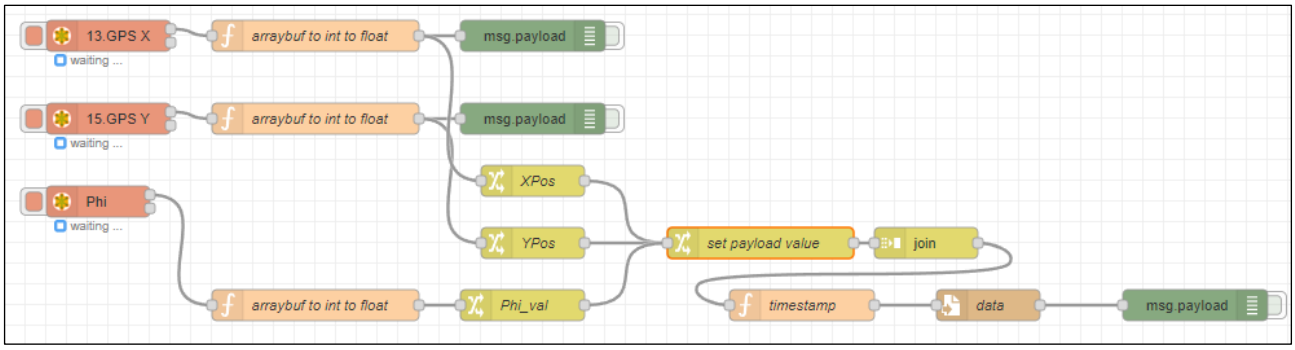

Figure 2. 2 - Node-Red scheme for MiR100 pose

For UR3 we implemented similar schemes to register the angles of the arm joints. Moreover, in order to compare the results with the OptiTrack data (described later), we designed a scheme to measure the $[x; y; z]$ position of the TCP.

The polling rate is always 100ms.

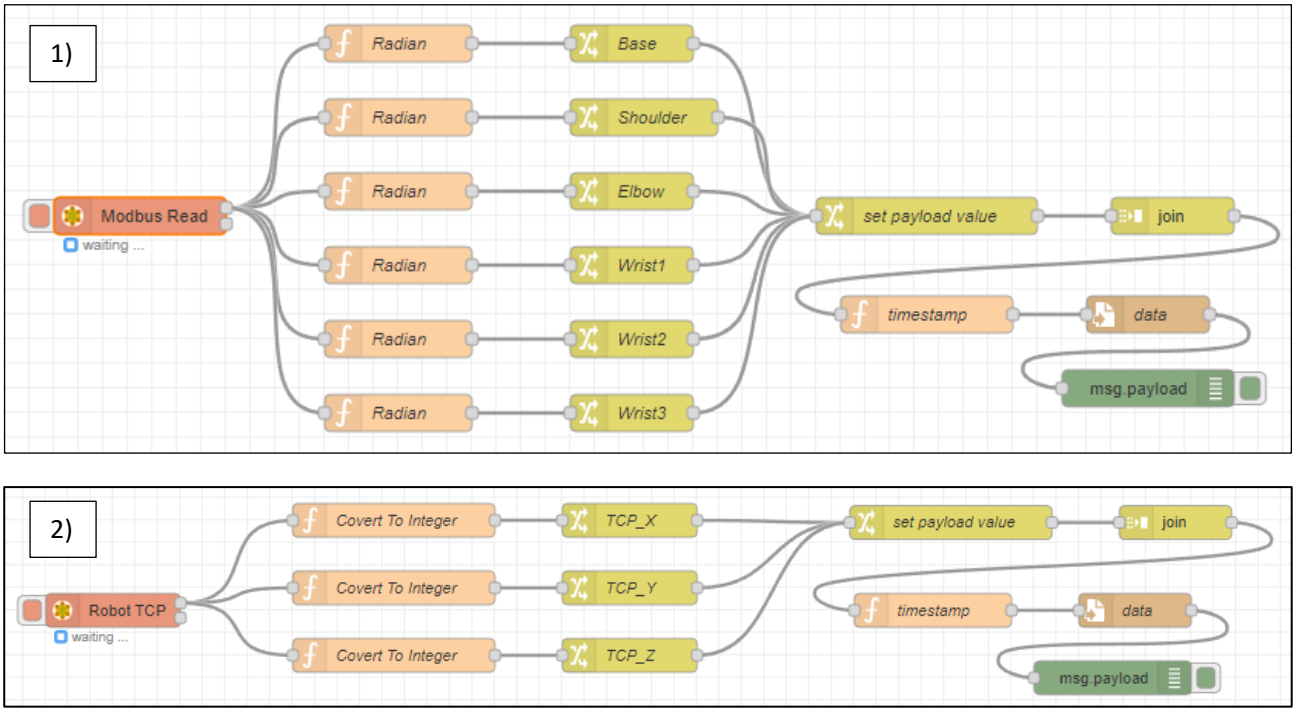

Figure 2. 3- UR3 Node-Red schemes for:
1) Robot pose;
2) TCP position.

Finally, all the information are gathered inside separate excel files that will be read by a suitably written Matlab code.

The outputs generated by this code will be the inputs for our model. Basic conversions from meters to millimetres, from degrees to radians, or offsets in the coordinates to place all the data with respect to the same reference frame will be necessary.

2.2.2 – OptiTrack data

While the robot is executing its task, the OptiTrack cameras keep track of all its actions and save everything inside a Matlab table.

In particular, what we save are $[x; y; \phi]$ for MiR100 and $[x; y; z]$ for the TCP.

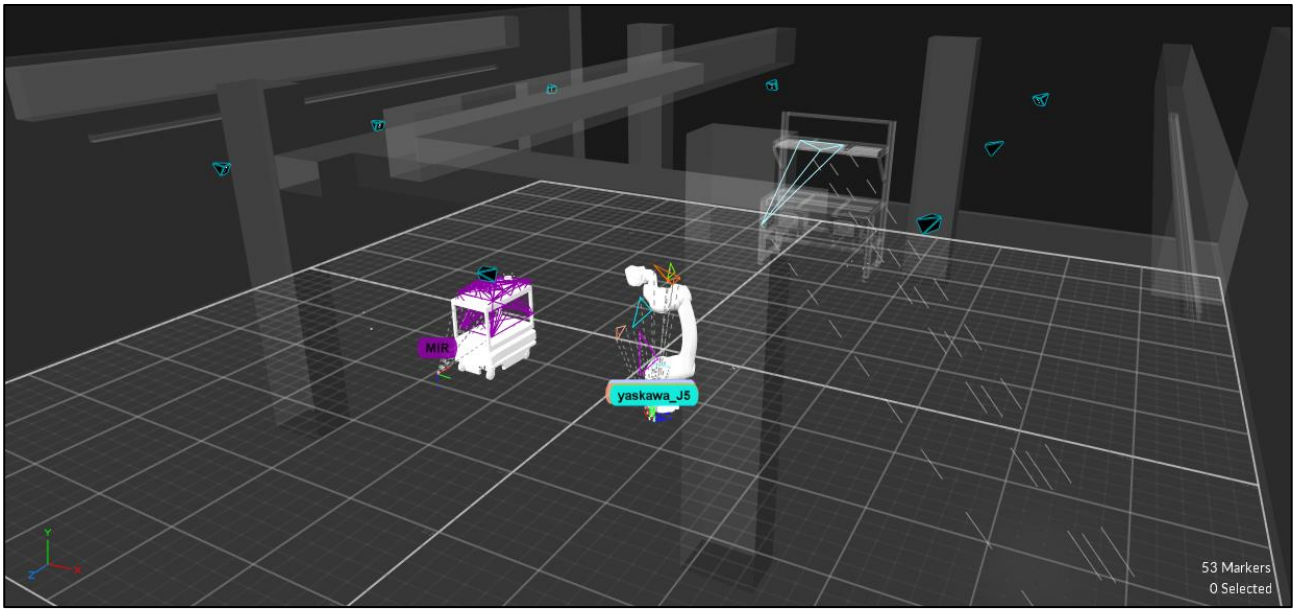

Figure 2. 4 - OptiTrack view of MiR100, UR3 TCP and a Yaskawa manipulator in the Laboratory

Since these 3D cameras have a higher precision with respect to the control-boxes of the robots, we use their data as an external check on the real robot behavior.

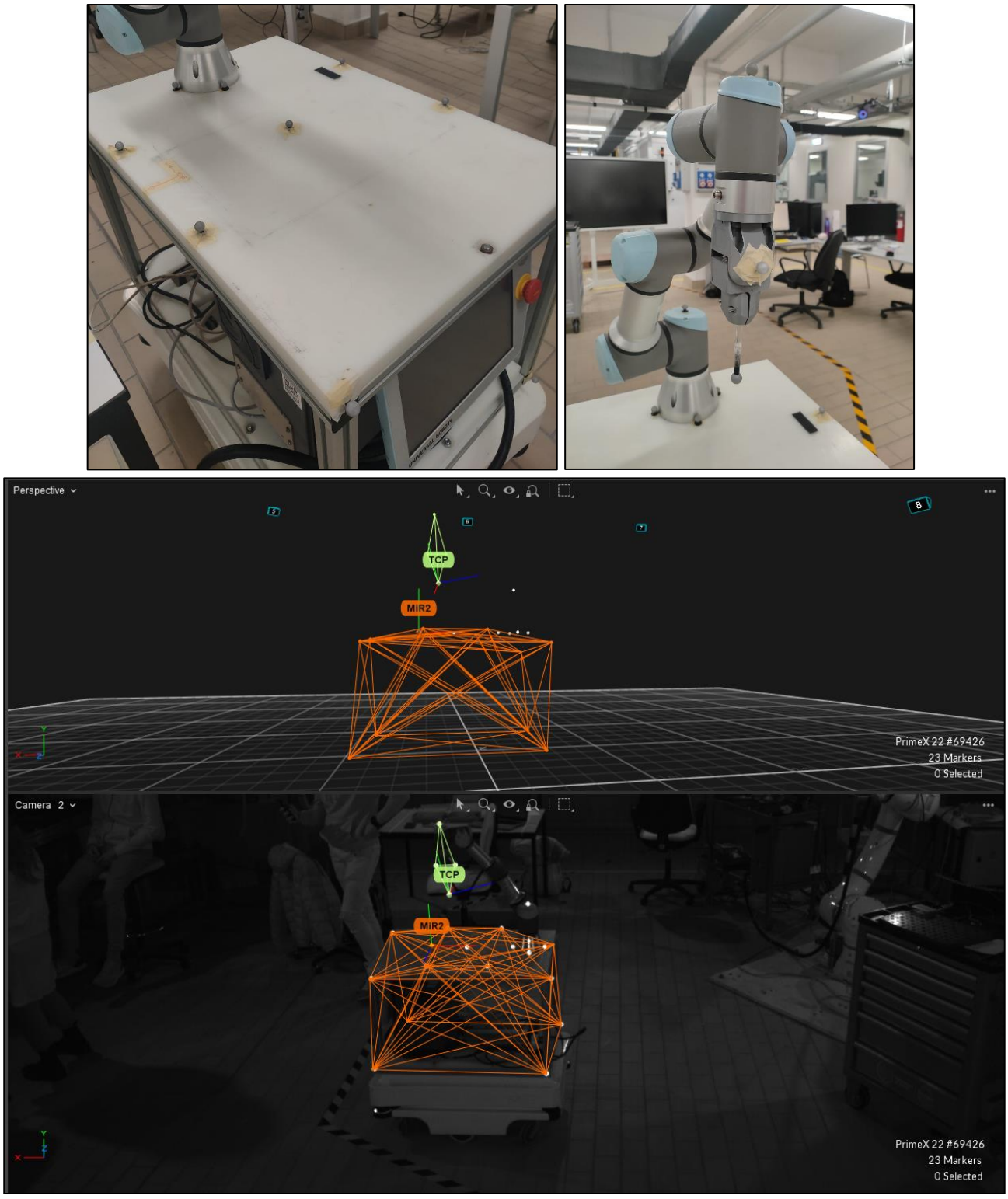

Figure 1. 7 - MiR100 structure and TCP recreated by the markers

In *Figure 1.7*, we can see how, based on the markers placed on the physical object, Motive is able to recreate a rigid body structure and detect a pivot point with respect to which all the poses will be measured.

The scripts used for this data collection are named 'manufacturingEnv.m' and 'NatNetPollingSample.m'.

```
% manufacturingEnv.m

clear
clc

freq = 100; %ms
durata_task=400; %tempo di acquisizione=durata_task*100ms

%Create table to save data:
rb_names = ["mir", "UR3"];
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","PHI", "TIME"]);
end
intestazione = cat(1, name_row, coord_row);
intestazione = array2table(intestazione);
empty_table = [empty_table; intestazione];

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

With this first code, we generate a Matlab table in which every 100ms a pose is registered.

To do so, a function 'NatNetPollingSampleCustom' has to be designed in order to connect Matlab to Motive, the software that exploits the OptiTrack cameras.

This function will be called at every sample time step.

In the following code section, we can see how the function works.

```
% NatNetPollingSampleCustom.m

function [final_table] = NatNetPollingSampleCustom(nbodies, sample_time,
data_table,task_time)

% [... system configuration code section ...]
        rb_pose = [];
        pose_row = [];

        %Timestamp:
        timestamp = datestr(clock, "HH:MM:SS.FFF");
        split_time = split(timestamp, ':');
        h_in_sec = str2double(split_time(1)).*60^2;
        m_in_sec = str2double(split_time(2)).*60;
        sec=str2double(split_time(3));
        poll_time = h_in_sec+m_in_sec+sec;
```

First, it initializes two empty arrays to contain the pose acquisitions.

For each acquisition, a timestamp has to be saved for future plotting on the time axis.

```
        for k=1:nbodies
            %Orientation:
            q_x = data.RigidBodies( k ).qx;
            q_y = data.RigidBodies( k ).qy;
            q_z = data.RigidBodies( k ).qz;
            q_w = data.RigidBodies( k ).qw;
            q = quaternion(q_x, q_y, q_z, q_w);

            angles = EulerAngles(q,'xyz')*360/(2*pi);
            %The first element is the phi:
            phi = angles(1);

            %Pose:
            pose = [data.RigidBodies(k).x*1000, data.RigidBodies(k).y*1000,
            data.RigidBodies(k).z*1000, phi];
            pose_row = cat(2, pose_row, pose, poll_time);

        end
```

At every sample time a pose is taken and to analyse multiple bodies in the system we need a for cycle that for every active asset on Motive saves its pose $[x, y, z, \phi]$.

The number of iterations at each sampling time depends on how many active assets are being registered by Motive.

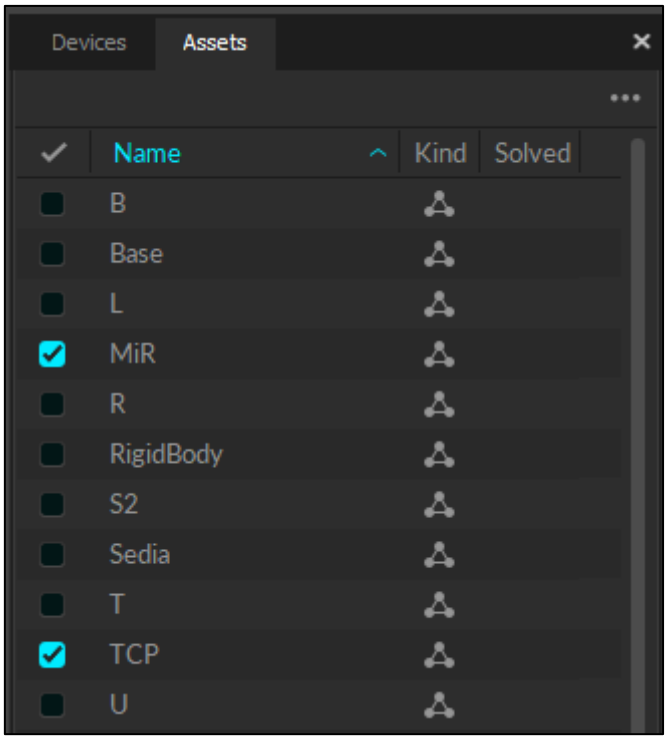

Figure 1. 8 - Example of assets list in Motive interface

```
    rb_pose = cat(1, rb_pose, pose_row);

  end
        data_row = array2table(rb_pose);
        allVars=1:width(data_row);
        newNames=append("intestazione",string(allVars));
        data_row=renamevars(data_row,allVars,newNames);
        final_table = [data_table; data_row];

    disp('NatNet Polling Sample End' )
end
```

After all the active assets poses have been wrote down, everything is put together in the same Matlab table.

2.2.3 – Data comparison

Briefly, in this paragraph we will explain how the aforementioned data will be compared with respect to each other.

For MiR100, $[x; y; \phi]$ acquired through Node-red is fed to the simulation system to replicate the motion.

Once the simulation has terminated, the obtained output is another set of $[x; y; \phi]$. In the end, three sets of data are plotted on the map of the laboratory to obtain a visual comparison.

About the UR3, the comparison is:

- On the positioning of the TCP in the workspace, $[x; y; z]_{TCP}$ from Node-red, OptiTrack and Simulation system. The latter set is obtained sending to the virtual model the joints configuration sequence registered in the UR3 software;
- Between the poses reached by the joints of the digital manipulator and the reference poses from the UR3 HMI. For this comparison, Node-red and Simulation data are not compared with respect to OptiTrack since due to the dimensions of the UR3 it is not possible to efficiently place the markers on each joint to acquire the rotation angles during the task.

In the following chapters, a case study will be discussed with its data comparison.

3 – Simulation system

3.1- Model generation

The generation of the Simscape multibody model starts from the 3D design of the system, in our project this step has been carried out on SolidWorks.

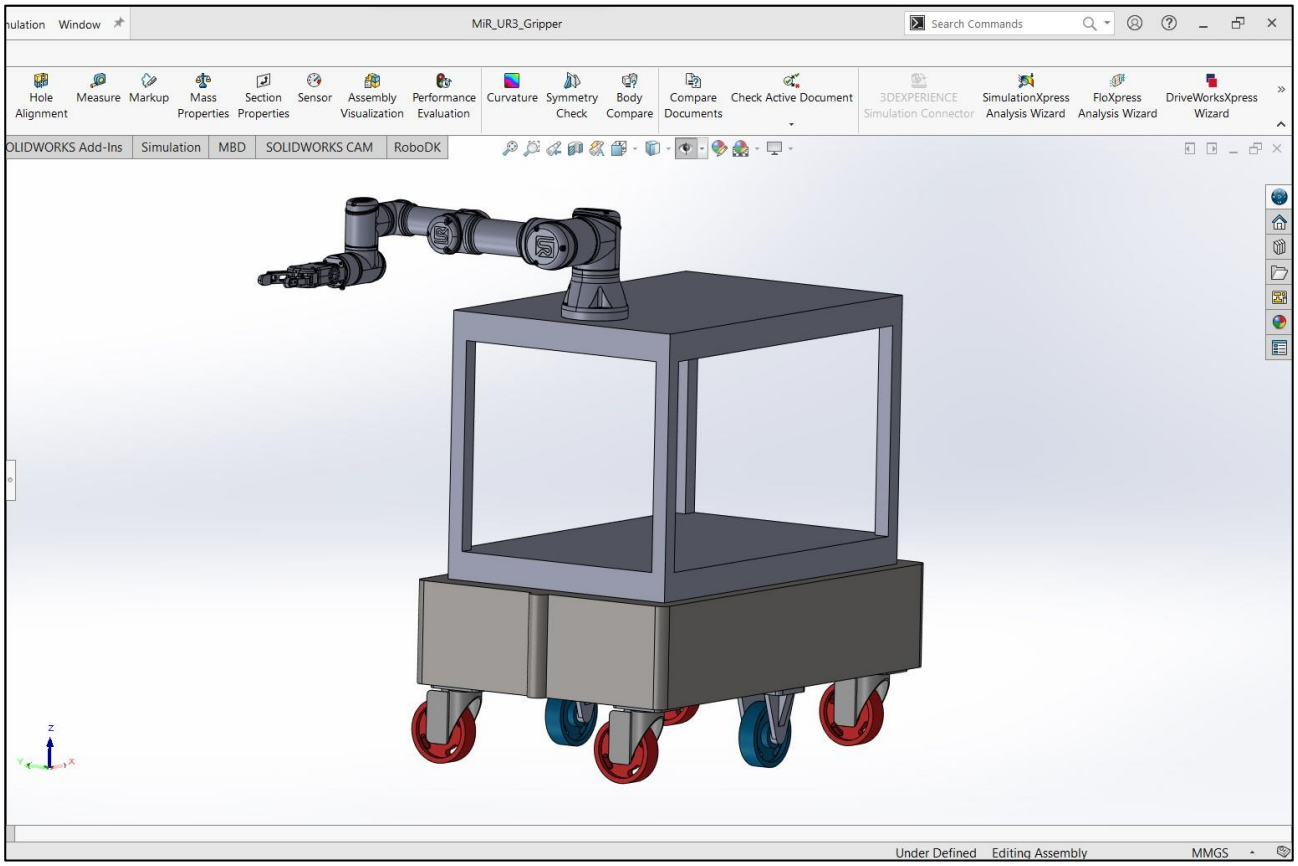

Figure 3. 1 - 3D design of the mobile manipulator on SolidWorks

In some cases, the .STEP files are provided by the manufacturer, whenever they are not, a design approximation has to be performed.

Designing the model, it is important to properly set the various constraints and set as ‘mobile’ the moving parts of the robot in order to generate a Simscape model with the same degrees of freedom of the real robot.

Moreover, it is very important to set the 3D model in its zero-pose before importing it in Simscape multibody to avoid rotation offsets of random value in case the import is executed in a generic pose. This aspect is of vital importance especially in case of multiple DOFs manipulators.

Once the 3D design is complete, the Simscape Multibody extension on SolidWorks allows us to import the whole model generating the virtual robot.

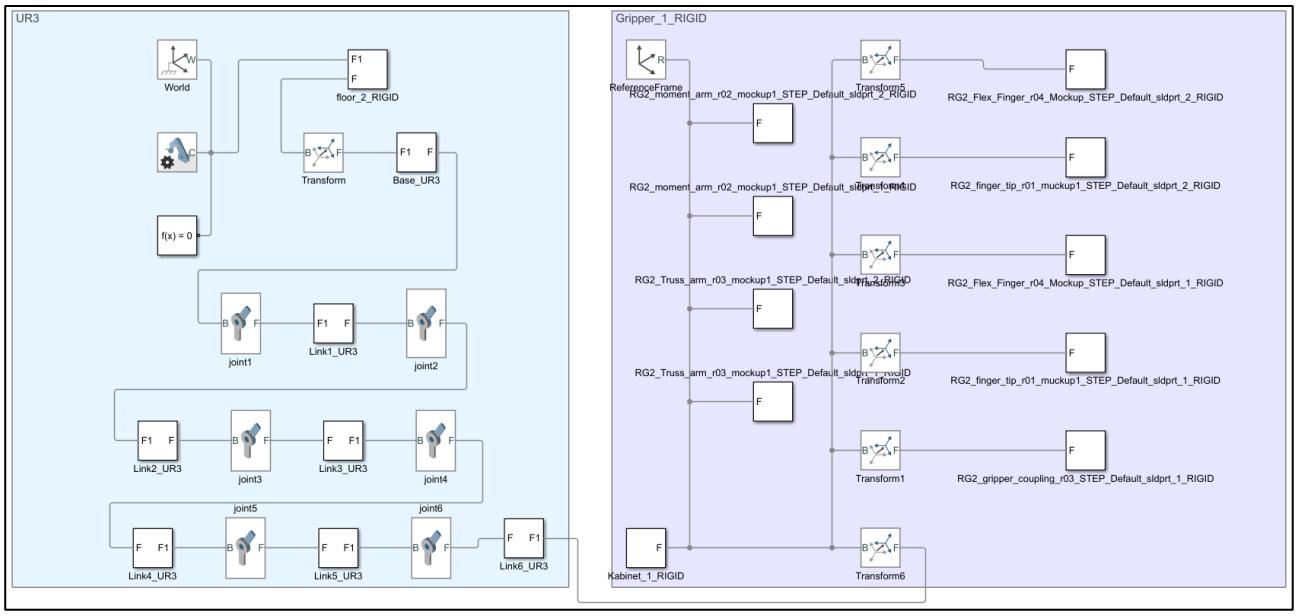

Figure 3. 2 - Example of raw Simscape model

What we obtain after these first steps is a raw Simscape Multibody model that still needs modifications to obtain a useful prototype. We will see them more in depths when discussing the models of this project.

Afterwards, as we already mentioned in *Chapter 1.5*, a control system has to be implemented. The algorithm behind this model may be the original one of the robot if available or can be designed to satisfy the behavior that we think the digital robot should have.

3.2 – Overview of the complete Simscape model

Before entering the detail of each model, we can show the whole MoMa Digital Shadow and give a schematic explanation on how it works.

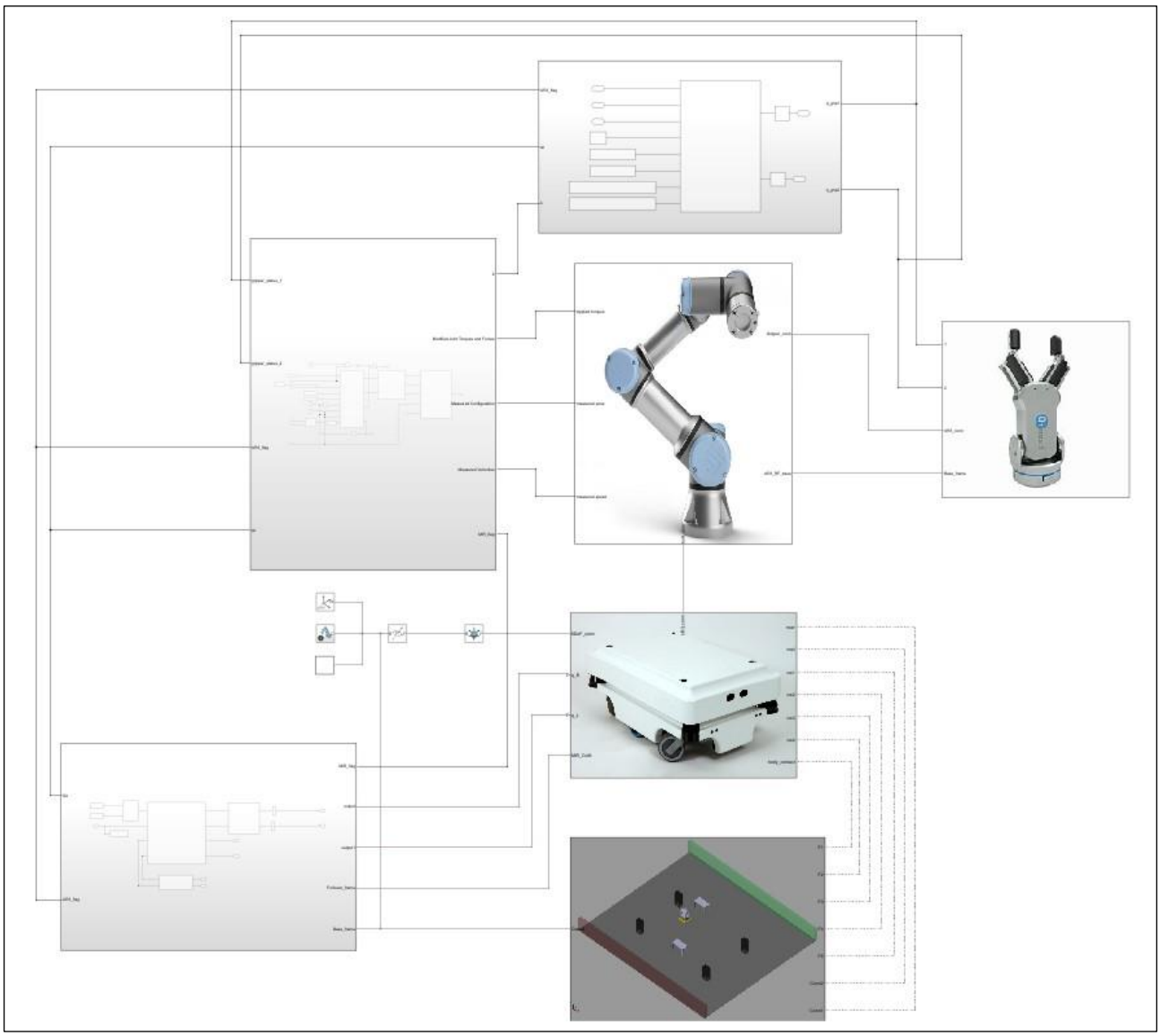

Figure 3. 3 - Complete MoMa Simscape model

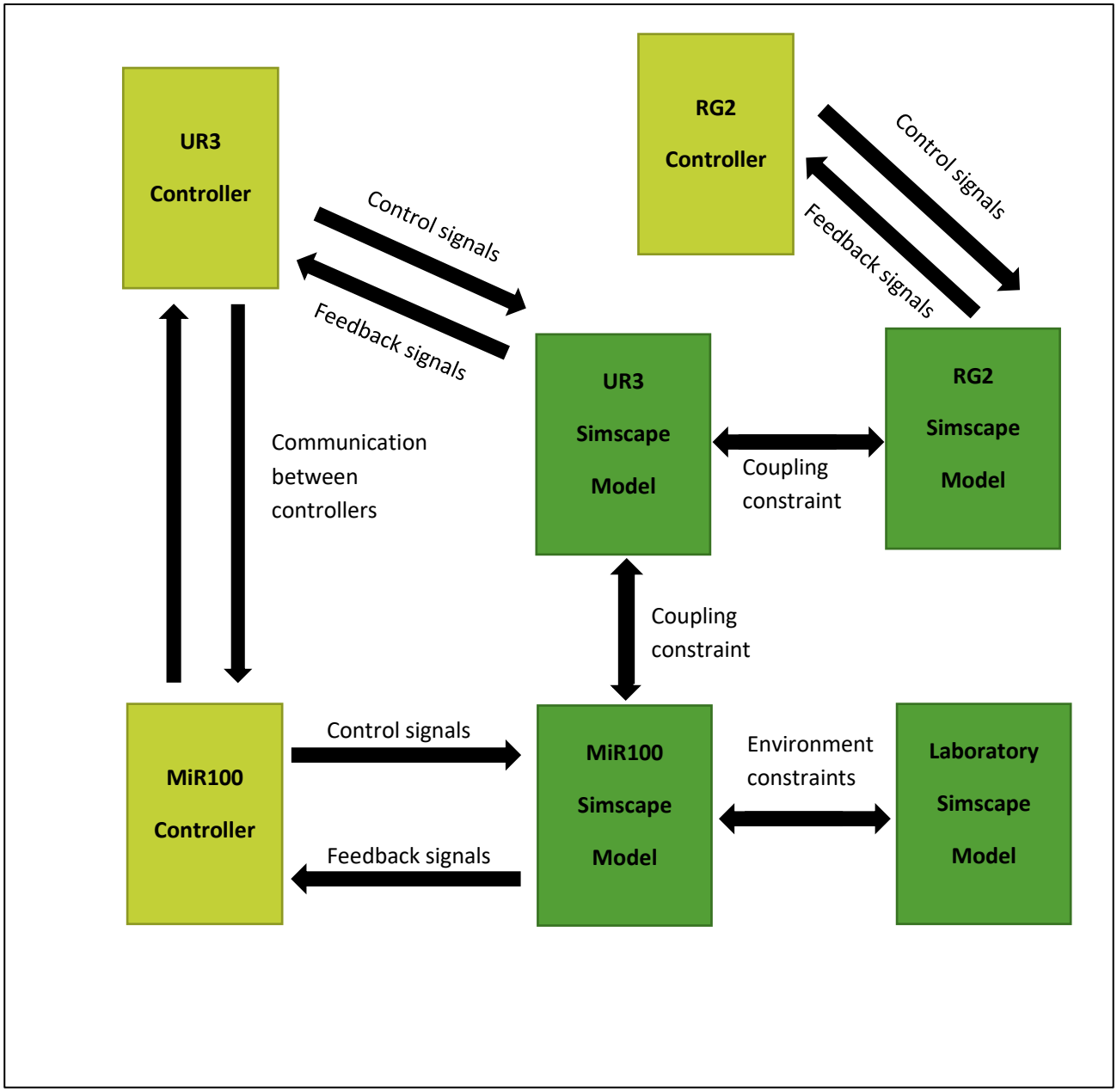

Figure 3. 4 - Block scheme

The figures above show a screenshot of the overall Simulink-Simscape model of the mobile manipulator and a block scheme to explain the basic working principle of the system.

From the second one, it is understandable that each Simscape model, laboratory aside, has its own controller. All of them communicate with each other to allow the different parts composing the digital MoMa to work in harmony. This communication happens through flags and constant values. Specifically, flags are used to trigger or stop the motion of the Simscape model connected to the controller, while the constant values are passed from a controller to another to communicate what the actual pose of the MoMa is and based on that value how to behave.

Each controller receives feedback signals about the pose of the model connected to it and based on these values combined with the flags and constants mentioned, the control function sends signals to the model.

Finally, environment and coupling constraints have the function of avoiding bodies compenetration and allow them to move together or with respect to each other.

For each block we dedicated a specific paragraph to enter more in depth the working principle description.

3.3 – Environment Simscape model

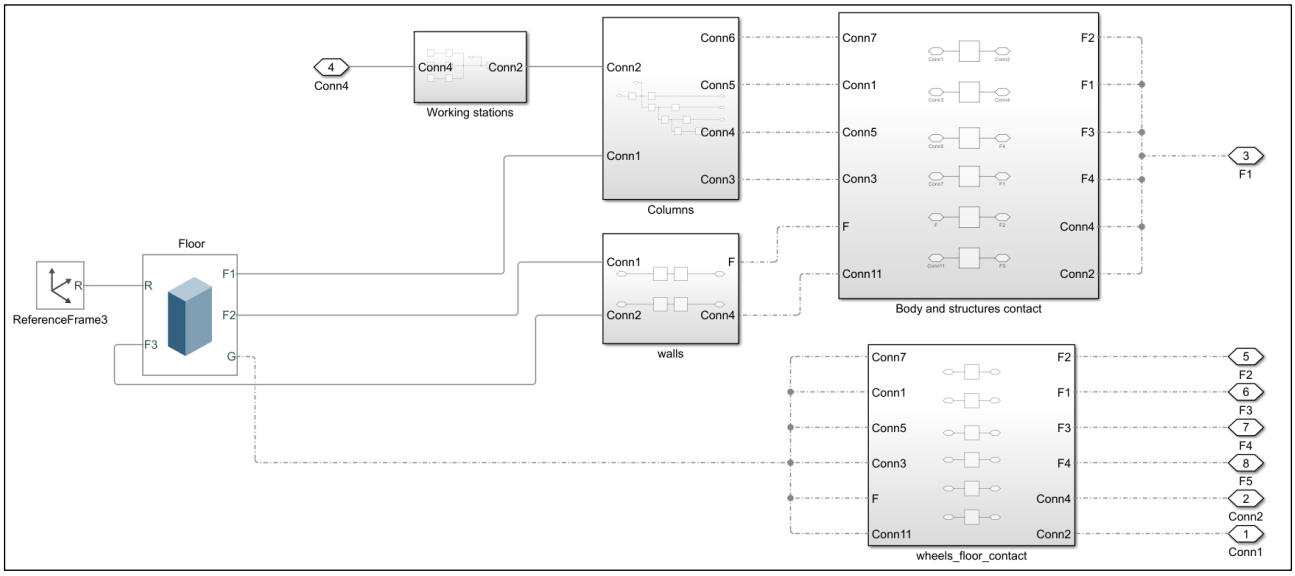

Figure 3. 5 - Laboratory Simscape model

This Simscape model of the laboratory is modeled with solid blocks and works as a visual reference while the Mobile Manipulator is moving.

Walls, working stations and columns are placed on a floor based on the real measures of the laboratory scaled to respect the binary occupancy grid map scaling factor.

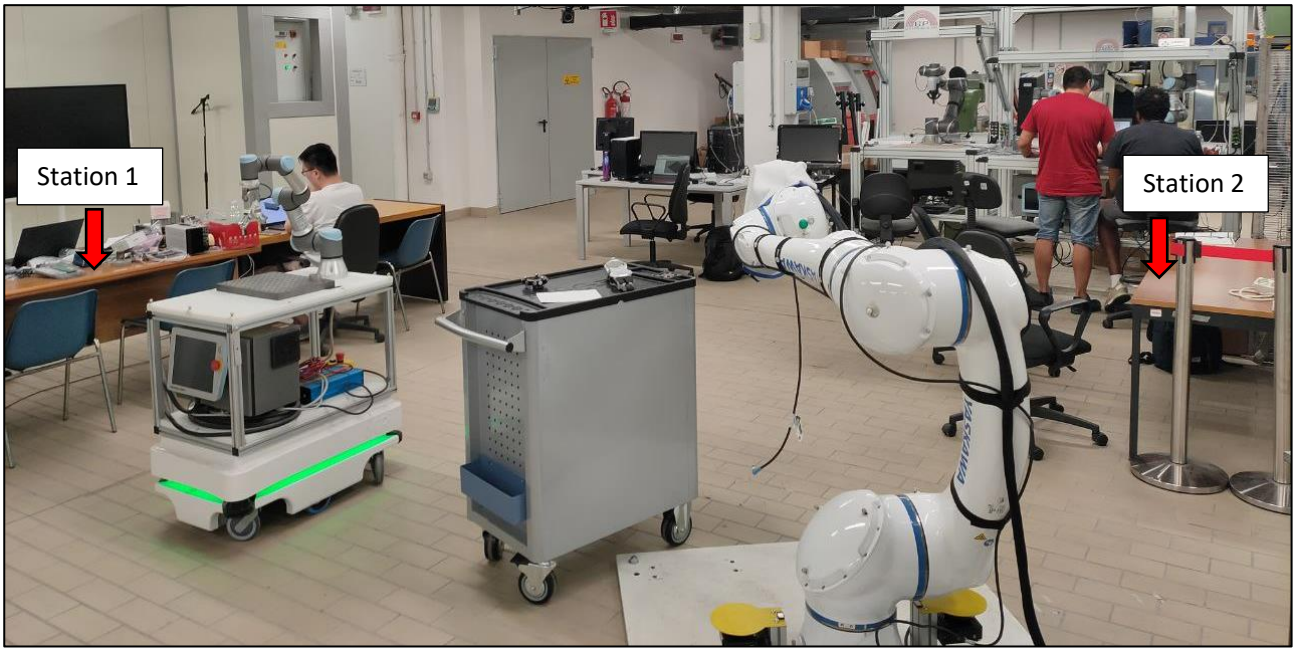

Figure 3. 6 - Laboratory

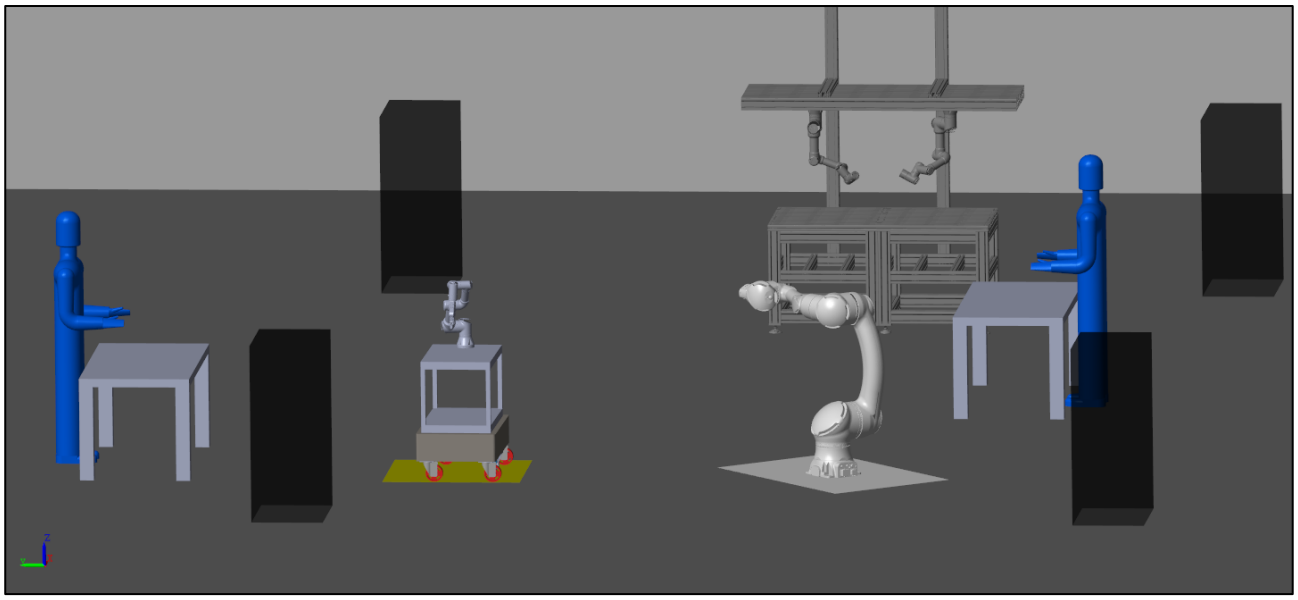

Figure 3. 7 – Laboratory Simscape model

As we can see from *figure 3.7*, all the mobile obstacles have not been recreated, but just the fixed reference points that are present on the map. The reason behind this choice is because for this thesis we will analyze cases related to an obstacle-free working area of the MoMa since we are still in the first digitalization phases and we have to study the robot trajectories in standard scenarios and not random trajectories due to moving obstacles.

In different situations, it is possible to deepen the obstacle-avoidance feature adding multiple impediments on the path.

3.4 – MiR100 Simscape model

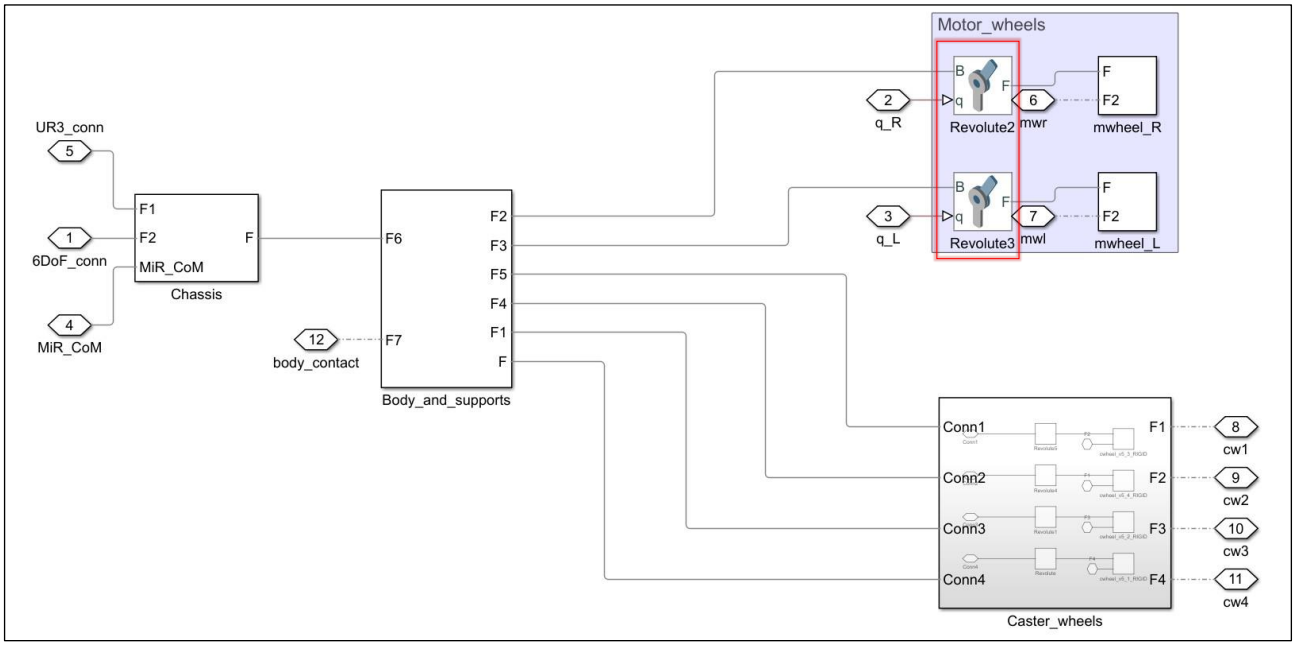

Figure 3. 8 – Simscape model of MiR100

MiR100 model has been designed with SolidWorks and then imported into Simscape. We remark that the original and detailed 3D-model was not available in a movable configuration, thus an approximation has been designed.

Before describing the Simscape model, to explain the meaning of the input/output blocks we report a zoom of *figure 3.3* to explain the essential blocks that are present in each model:

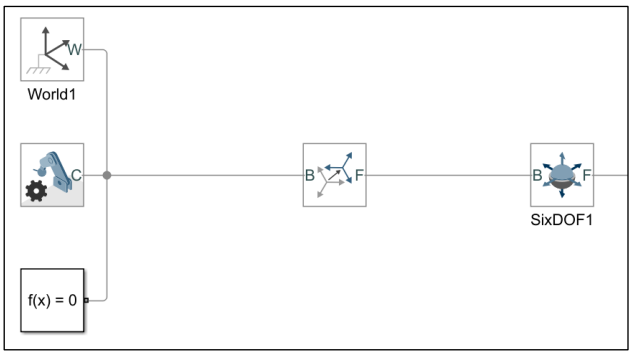

Figure 3. 9 - Detail of figure 3.3

The blocks shown are:

- The 'World' reference frame block: the whole system moves with respect to this reference frame. With respect to this frame, both MoMa and laboratory models are placed to be consistent with the Map generated by binaryoccupancygrid.m;
- The rigid transform block is frequently used inside the model to rotate/translate parts with respect to a base reference frame. In this case, the block places MiR100 in a given starting pose that we decide and can change depending on where we want it to start its task;

- The 6DoF block allows the robot to move freely in all 6 degrees of motion and the environment constraints described before avoiding the MoMa to pass through the modelled floor falling into the void;

Now, we can describe the model of *figure 3.8:*

- Wheels joints: highlighted in *figure 3.8* are joints corresponding to the motor wheels from which the motion of the MiR100 model starts. Those joints receive the input to rotate and move the whole system in the simulated laboratory. Joint blocks can be modified to choose between motion or torque input. In this case we chose a motion input with automatically computed torque;
- The caster wheels joints are contained in another subsystem, namely 'Caster_wheels'. Those do not impose any motion but work as support for the body and motion dexterity.

The rest of the system is composed of subsystems containing the other physical parts of the 3D model.

3.4.1 – Control system

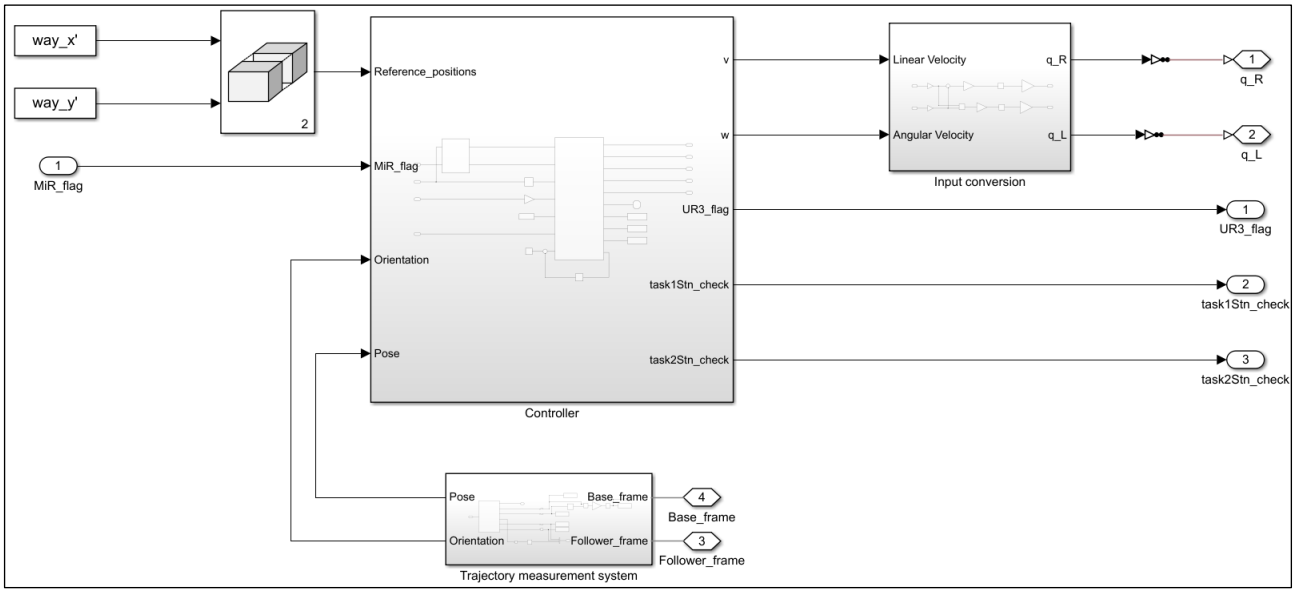

Figure 3. 10 - Simulink control system

Figure 3.10 shows the control system implemented on Simulink. Each part of it will be described leaving at last the Controller macro-block.

- Input Conversion:
  The linear and angular velocities generated by the controller refer to the center of mass of the MiR100, so we have to convert them into right and left wheel rotations to obtain coherent inputs for our Simscape model.

The conversion can be done through the Differential Drive Mobile Robot Equations:

$$\omega_R = \frac{2v + \omega L}{2R}$$

$$\omega_L = \frac{2v - \omega L}{2R}$$

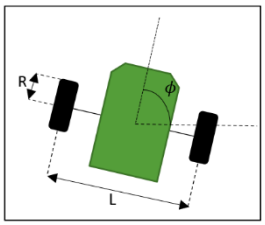

Figure 3. 11 - Differential drive robot scheme

The Differential Drive Mobile Robot Equations implementation is obtained through Simulink elementary blocks.

- The 'Trajectory measurement system' acts as a feedback path and contains the blocks needed to measure the pose of the Simscape model with respect to the World reference frame.

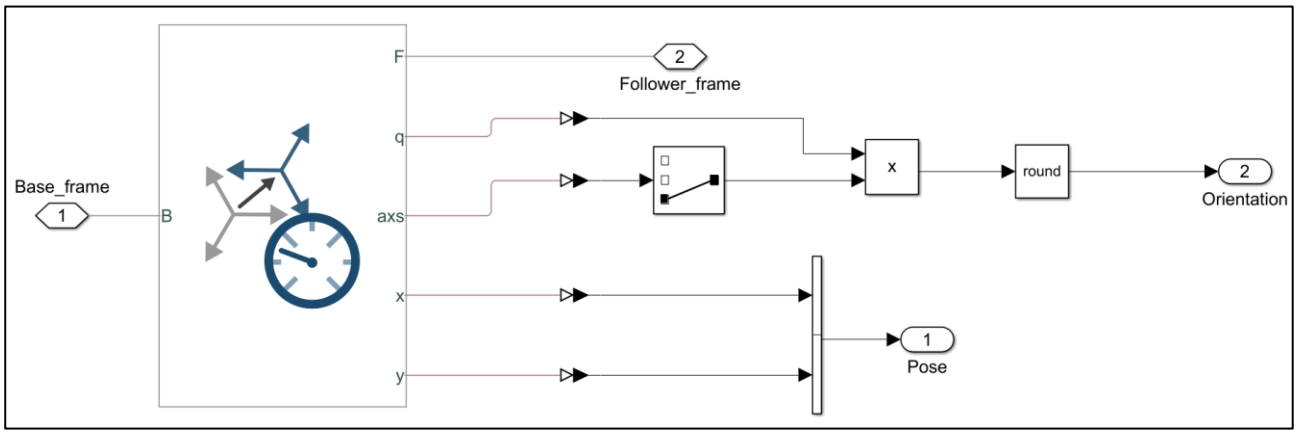

Figure 3. 12 - Pose measurement

Since the laboratory is placed in the Simscape world to respect the coordinates of the map, all the data gathered by this block are already consistent with the map reference frame.

The position and the orientation are then sent back to the function block as feedback and saved in Matlab workspace for later comparison.

- Controller, in which is contained the actual controller, *figure 3.13*:

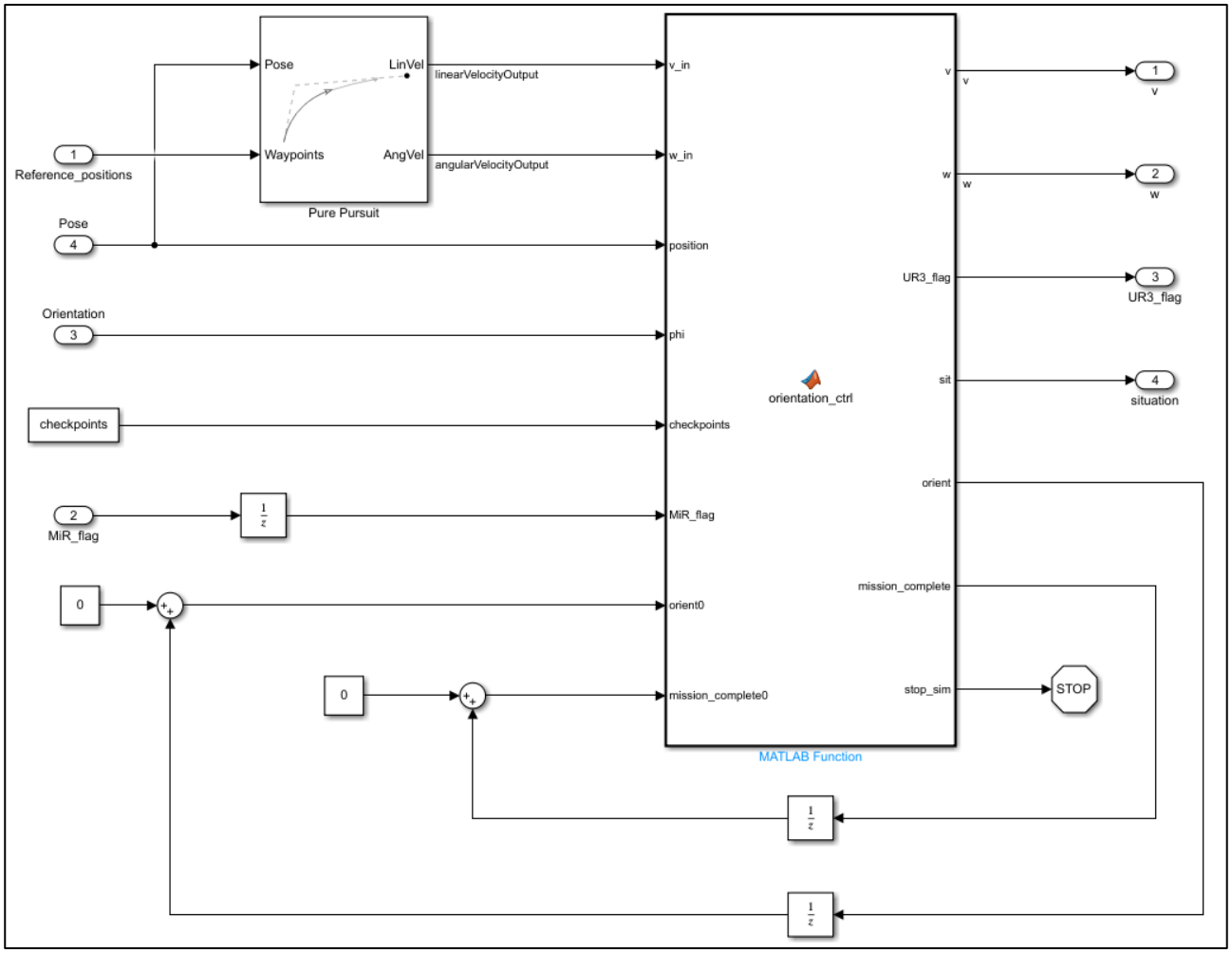

Figure 3. 13 - Controller

Before entering the details of the control function, we can examine the PurePursuit[7] block whose inputs are the current position of the robot and the sequence of positions $[x; y]$ acquired through Node-red during a real robot motion used as reference path.
Given the waypoints, the corresponding linear velocity and angular velocity needed to follow the traced path will be generated.

The settings we used are such that the DS maximum speeds can be $0.5\frac{m}{s}$ and $1\frac{rad}{s}$, this restriction differs from the actual values of the real robot that are $1.5\frac{m}{s}$ and $3\frac{rad}{s}$. But, in an actual collaborative mode in a small workspace, we could see that the MiR100 never reached those speeds and moved at values closer to the ones we set.

[7] For additional documentation: https://it.mathworks.com/help/robotics/ref/purepursuit.html

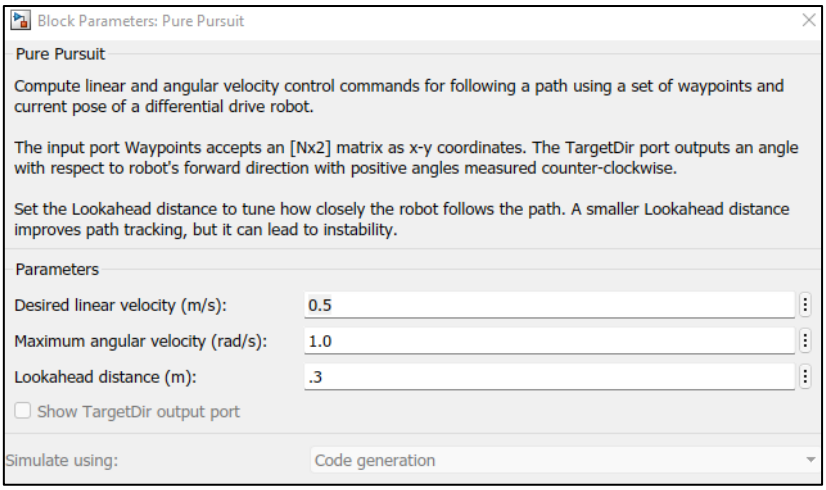

Figure 3. 14 - Pure Pursuit settings

Another important value is the Lookahead distance. This parameter is used by the controller to understand how far on the path it has to track the next waypoint to reach.

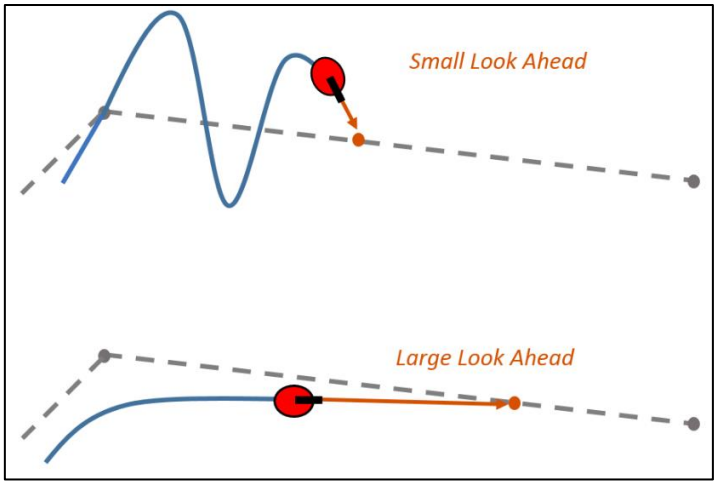

Figure 3. 15 - Different values of Look Ahead distance[8]

From *figure 3.15* , we can observe how changing this parameter can affect the resulting motion:

- Small Look Ahead : everytime the DS moves outside the path it will try to get back on it. But considering that the pose of the robot is acquired from a single point and that rarely it will execute a linear motion, the PurePursuit controller will often detect the MoMa outside of the path and will send velocity values to regain the desired position. This will result in multiple oscillations from left to right with the mobile robot continuously overshooting the trajectory. Moreover, the resulting behavior will not resemble the real one;
- Large Look Ahead: a bigger value will solve the oscillations problem, but it will results in wide turns when approaching a corner in the path resulting in missing possible important points that have to be reached to execute the task. In conclusion, an intermediate value has to be chosen.

[8] Figure taken from 'https://it.mathworks.com/help/robotics/ug/pure-pursuit-controller.html' .

The effect of this parameter will be relevant when discussing the DS positioning precision in *Chapter 4*.

It is important to remark that the Pure Pursuit algorithm generates v and ω in order to follow the path but does not follow the various orientations of the real MiR100, neither it stops it in the interesting points to allow the UR3 to operate.

To make the model able to replicate the attitude of the real one, we implemented a Matlab Function that overwrites the velocities values each time a working station is reached.

The inputs of this function block are:

- v_in and w_in: these are the linear and angular velocities sent by the PurePursuit algorithm in order to follow the reference path. Their values update depending on the model current pose;
- position : this input is generated selecting just the x and y coordinate from the pose vector acquired by the Trajectory measurement system described earlier;
- phi : from the same system, we can measure the actual orientation of the robot and send it to the controller;
- checkpoints : everytime we run the model we have to upload a task. To allow the model to execute it, we have to specify the location of the points of interest to be reached such as stations or generic stops and the orientation that needs to be assumed to let the manipulator work;
- MiR_flag : this flag is sent by the UR3 controller and allows the mobile robot to perform its actions if and only if the UR3 is not moving. Specifically, MiR_flag is always set to 0 and becomes 1 if and only if the manipulator ended its task. Thus, MiR_flag = 1 when the approaching orientation phase and UR3 task are completed, allowing MiR100 to start the leaving orientation phase;
- Orient0 : this is just an initialization values to be updated during the simulation run. Its value will trigger or stop different actions inside the code;
- Mission_complete0 : as the previous value, this input initializes a value at 0. Once all the tasks of the mission have been executed, the value corresponding to it is updated at 1 stopping the simulation instantly or in this case when the robot has got back to the starting point with the desired orientation.

Once the function block has read the inputs, the following outputs are sent to the Simscape model depending on the part of code that is activated[9]:

- v and w : at every step of the simulation, the function block compares the digital robot actual position with the data contained in the input checkpoints matrix. When the CoM of the robot reaches a point of interest within a predefined tolerance that, at start, we set at $\pm 0.15\ m$, the velocity commands v_in and w_in are overwritten and set to 0 before proceeding in the orientation phase.

[9] To avoid adding the complete code, just interesting parts and singular cases are used as explanation support.

```
% CHECK POSITION:
if norm(position-checkpoints(1,1:2)',2) <= pos_thr && mission_complete <= 0
    situation = 1; %Charging station
    sit = situation;
    phi_arr = phi-checkpoints(1,3);
    phi_res = phi-checkpoints(1,4);

    delta_arr1 = abs(phi-checkpoints(1,3));
    delta_arr2 = abs(abs(phi)-abs(checkpoints(1,3)));

    delta_res1 = abs(phi-checkpoints(1,4));
    delta_res2 = abs(abs(phi)-abs(checkpoints(1,4)));

    v = 0;
    w = 0;
```

In this first part of code it is shown the aforementioned action. Moreover, the 'sit' output is set depending on the station we are in and the difference between the actual orientation and the desired orientation, approaching or leaving the station, is computed.

It is appropriate to clarify that this code section refers to the case in which the robot is in its charging station, but the same structure is used for all the stations and more in general it applies to different environments or missions with other points of interests. It all depends on the checkpoint matrix that we compute everytime we design a mission.

```
[…]
else
    situation = 8;
    sit = situation;
    v = v_in;
    w = w_in;
[…]
```

Next, whenever the robot is moving from a point to another, velocity signals are not overwritten and the motion control is given back to the PurePursuit block.

Once the robot has been stopped, the orientation phase starts. In the following section of code it is showed the imposed motion on a general station approaching:

```
[…]
    case 2
        if phi*checkpoints(2,3) > 0
            if phi_arr < 0 && abs(phi_arr) > phi_thr
                w = 0.2;
                UR3_flag = 0;
            elseif phi_arr > 0 && abs(phi_arr) > phi_thr
                w = -0.2;
                UR3_flag = 0;
            elseif abs(phi_arr) < phi_thr
                w = 0;
                UR3_flag = 1;
            end

        else
            if delta_arr1 < delta_arr2
                w = 0.2;
                UR3_flag = 0;
            elseif delta_arr2 < delta_arr1
                w = -0.2;
                UR3_flag = 0;
            elseif delta_arr1 < phi_thr || delta_arr2 < phi_thr
                w = 0;
                UR3_flag = 1;
            end
        end

[…]
```

These lines of code are written with the purpose of computing the verse of the angular speed to reach the desired attitude in the fastest way.

The maximum speed at which the model is able to orient itself is $\pm 0.2 \frac{rad}{s}$. Over this value, it is not able to reach the goal at the first try.

Once the oncoming attitude is reached, the function block sets UR3_flag = 1 and enables the manipulator action whose first command is to set MiR_flag = 0 to avoid irregular movements of the mobile part during its actions.

Once the UR3 has finished its task, it sets MiR_flag = 1 resulting in the MiR moving again to re-orient itself, with a similar code, before leaving the station. This last action is essential to avoid the PurePursuit algorithm to lead the MoMa back to the traced path with unplanned maneuvers.

- UR3_flag : as discussed in the previous point, this flag is set to 0 when the mobile robot is moving and to 1 when the setup for the manipulator task is ready;
- sit : this value corresponds to an integer number related to a checkpoint or to the in between movement case and is used not only inside the function block to trigger different sections of code but it is also sent to the other controllers that will use it for the same purpose;
- orient : a constant used to avoid unexpected activations of lines of code each time the CoM is close to a station. Indeed, it may happen that during the leaving phase the robot is still inside the positioning tolerance and its motion toggles between the command of the PurePursuit block and the orientation part of the function block;

- mission_complete : it is a variable initialized at 0 that becomes 1 after the leaving phase on the last station has terminated, meaning that even the last task of the UR3 has been done;
- stop_sim : this value is initialized at 0, but when mission_complete is set to 1 and the robot has reached the desired pose in the conclusive station, that could be back to the starting point or another one that we choose, stop_sim is set to 1 triggering the Stop Simulation block outside the function block.

3.4.2 - Limitations

As mentioned in the introduction chapter, the biggest limitations of this model are basically:

- The fact that the real control algorithm of the mobile robot is not available for our use, thus we had to implement an approximation of it on Simulink using basic blocks and block functions resulting in a similar but not exact control algorithm;
- The fact that the robot design parameters are not totally available. Thus, from a dynamical point of view the Simscape model lacks of precision. Indeed, the simulation model cannot replicate all the velocities reached by the real robot, but has a maximum linear speed of $0.5\frac{m}{s}$ and maximum angular velocity of $1\frac{rad}{s}$. If those values were not saturated at those levels, the virtual robot would reach an unstable behavior;
- Finally, the rear-driving mode used by the real robot to execute some maneuvers is not implemented in our model.

In the following *chapter 4* those points will be discussed more in depth through the analysis of the obtained results.

3.5 – UR3 Simscape model

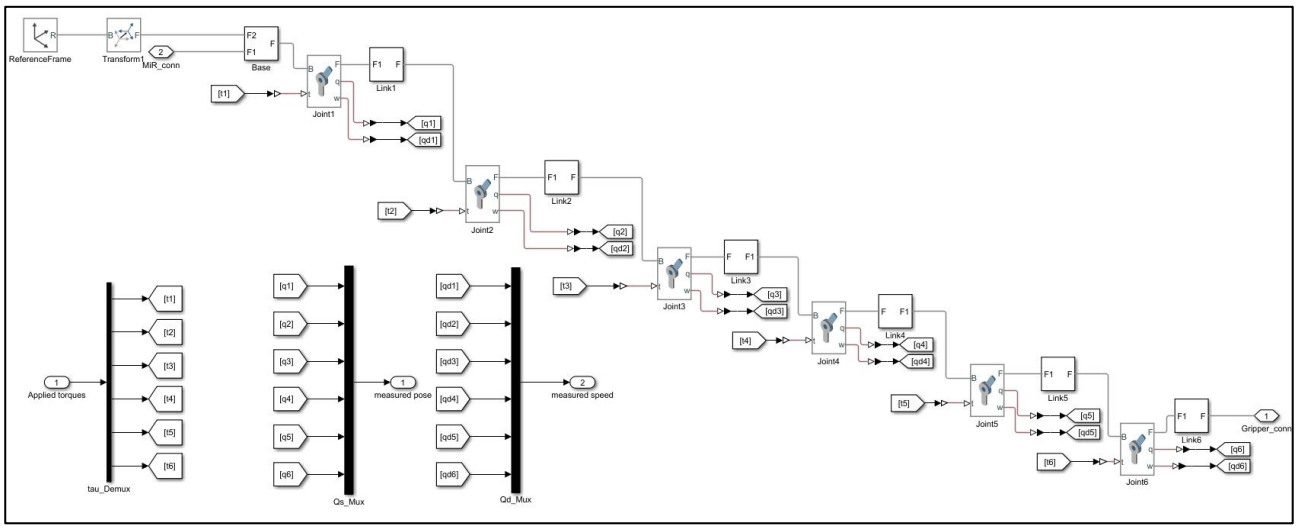

Figure 3. 16 - UR3 Simscape model

The same working principle applied to design the MiR100 model has been used to design the UR3 model.

In this case, each joint is set to receive a torque as input and sense their own rotation angle and speed. These last two data are sent back to the controller that will regulate the torque depending on the pose to be reached.

3.5.1 – Control system

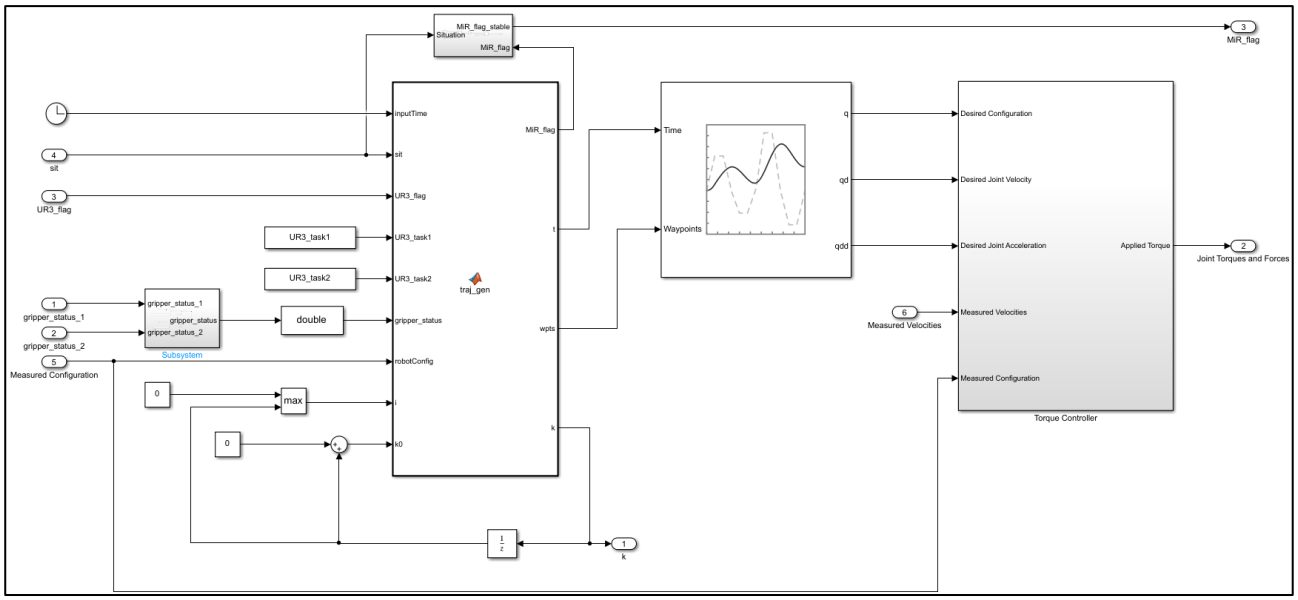

Figure 3. 17 - UR3 control system

Let us explain all the different parts composing the controlling system:

- Control Function Block : this is the block containing the function that based on the combination of different inputs, decides the pose to be reached next.
  Aside from the already explained sit and UR3_flag, the inputs are:
  - inputTime : through a clock block the current simulation time is sent to the control function to mark the time whenever a pose of the sequence has been performed;
  - UR3_task1 and UR3_task2 : the control function receives as input the pose sequences to be performed. In this specific case two different sequences are requested, but in a general mission a single sequence or more could be sent;
  - gripper_status : this value can be [0;1] and is sent by a subsystem that checks if the gripper joints are rotated in open or closed position.

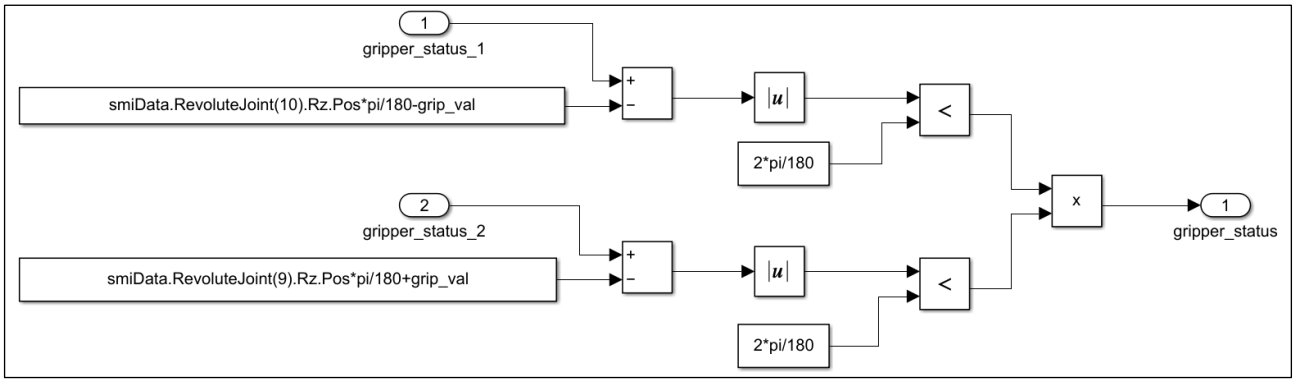

Figure 3. 18 - Subsystem to check the gripper status

  - robotConfig : an array composed by the position sensed by each joint;
  - i and k : simple variables used as indexes to perform for loops inside the function.

At each simulation step the controller elaborates the entering signals to send the following outputs:

- MiR_flag : this flag is set to 0 for the whole time the UR3 is moving, but once the task is completed it set to 1 to allow the MiR100 to resume its motion;
- t and wpts : the first one is the time that the Trapezoidal Velocity Profile needs to compute the joint rotation speed to go from a pose to the following one.

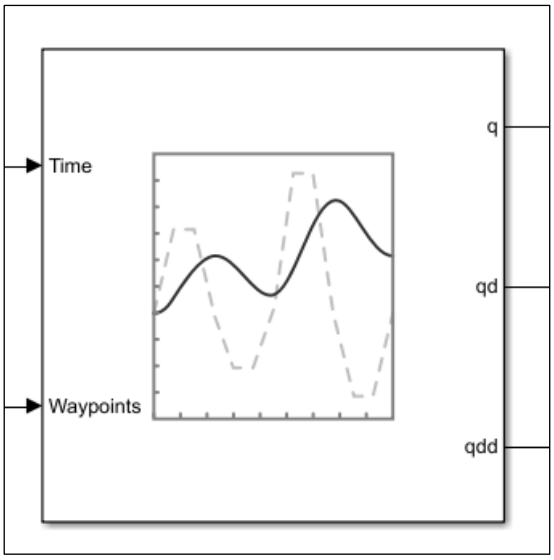

Figure 3. 19 - Trapezoidal Velocity Profile block

wpts are instead the waypoints of the trajectory, specifically, is an array composed of two columns. The first one contains the current pose of the robot and the second one contains the next desired pose.

```
% Initialize outputs to starting state
wpts = cat(2,robotConfig,UR3_task1(1:end-1,1));

if (sit == 2 && UR3_flag == 1)
    motionState = 1;
    wpts = cat(2,robotConfig,UR3_task1(1:end-1,1));
    MiR_flag = 0;
elseif (sit == 3 && UR3_flag == 1)
    motionState = 2;
    wpts = cat(2,robotConfig,UR3_task2(1:end-1,1));
    MiR_flag = 0;
elseif (sit == 8 || sit == 1)
    %Reset counter and flag
    MiR_flag = 0;
    k = 0;
    motionState = 8;
end
```

At the very beginning of the task, the function verifies at which checkpoint is the MoMa and if the UR3 is enabled. Based on that information it builds the wpts array with the robot current configuration and the first pose of the task. This preset is used to avoid the Digital Shadow to start from unexpected poses due to possible hard oscillations generated by MiR100 abrupt maneuvers.

If we are instead in the middle of the MiR100 motion, the k index is reset and also MiR_flag is set to 0 to avoid it from reaching a station and skipping the approaching orientation state.

```
switch motionState
    case 1
        MiR_flag = 0;
        if i < length(UR3_task1(1,:)) && i >= 1
        %Pick task:
            MiR_flag = 0;
            wpts(:,1)=UR3_task1(1:end-1,i);
            wpts(:,2)=UR3_task1(1:end-1,i+1);
            [trajEndConfigReached] = endStateReached(robotConfig,
gripper_status, UR3_task1(1:end-1,i+1), UR3_task1(end,i), posTgtThreshold);
        elseif i < 1 %Preliminary adjustment
            MiR_flag = 0;
            wpts(:,1)=robotConfig;
            wpts(:,2)=UR3_task1(1:end-1,1);
            [trajEndConfigReached] = endStateReached(robotConfig,
gripper_status, UR3_task1(1:end-1,i+1), UR3_task1(end,1), posTgtThreshold0);
        elseif i >= length(UR3_task1(1,:))
            MiR_flag = 1;
        end
```

In the code section below it is shown how the configurations are read in the case of a particular task. If the index is still 0, the preset motion from actual configuration to the starting pose is performed. Otherwise the normal sequence is carried on until we reach the final configuration and MiR_flag is set 1 to start the leaving phase of the mobile robot.

- k : this index is sent back inside the function to proceed in the pose sequence execution and also to RG2 controller let it know at which pose the UR3 is and eventually if it has to open or close.

```
if trajEndConfigReached
  resetTimeValue = inputTime;
  k = i+1;
end
```

Whenever a desired pose is set, a function 'trajEndConfigReached' checks if the joints have all reached it within the set tolerance of $\pm\ 0.004\ rad$ and the gripper is correctly open or closed.

Once those two aspects have been verified, the index is increased to continue the task.

- Trapezoidal Velocity Profile and Torque Controller : the first one is showed in *figure 3.13* and it sends q, dq and qdd. Namely, the desired configuration, the joints velocity and accelerations needed to reach it.
  The Torque Controller gets those inputs and computes the torques that will be sent to the joints to perform the requested motion.

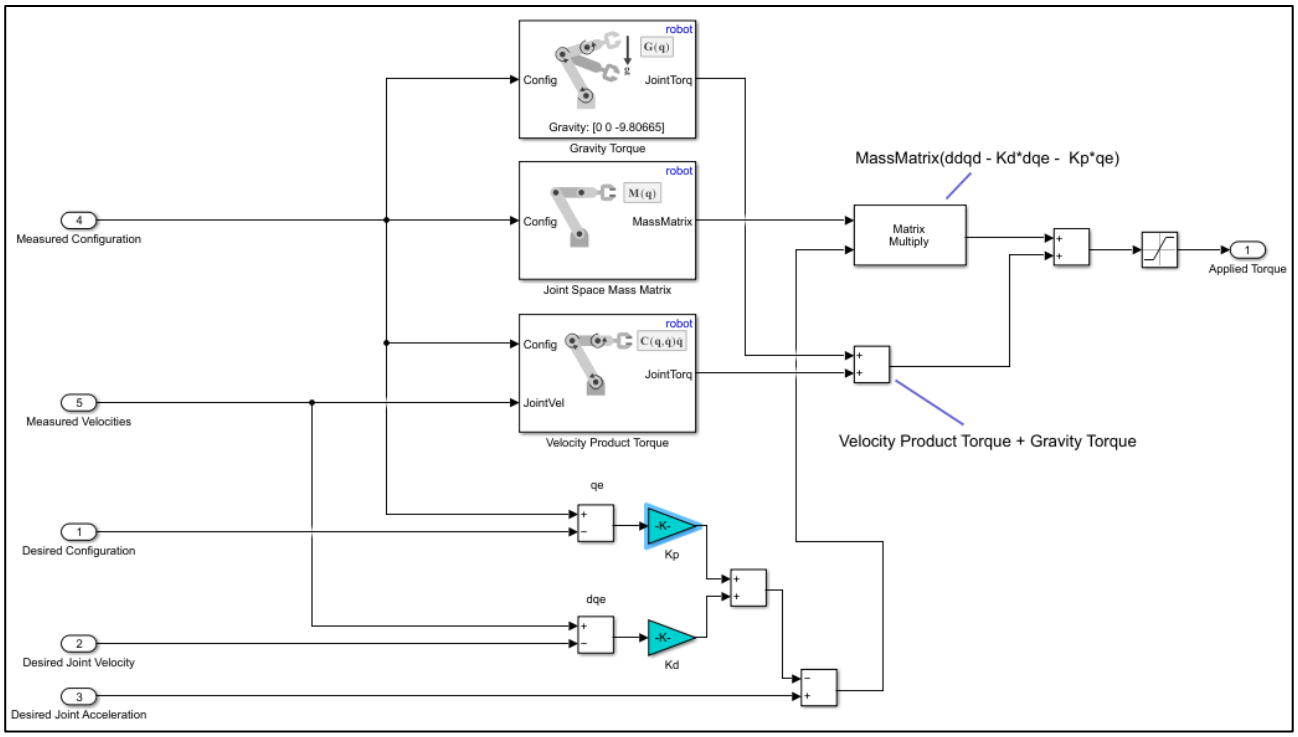

Figure 3. 20 - Torque Controller

The torque controller blocks send as output the applied torque depending on the type of robot. Indeed, unlike MiR100 case, Matlab has more information about the UR3.

3.5.2 – Limitations

For this model, the internal mechanics parameter of the joints were not given by the UR3 datasheet. However, we supposed them after running multiple tests in order to find a good compromise between a rigid structure able to maintain its pose while moving but flexible enough to replicate a smooth motion.

Moreover, the Simscape model has been generated by importing the official .step file of the UR3-CB3 series from the Universal Robots site. However, we found out a discrepancy between the 3D model measures and the real robot.

From the measures of *figure 3.9*, we can see how the error in the 3D space is about $2.853\ mm$. This misalignment with respect to the real robot will have relevant effects on the analysis of the TCP trajectory, but will not affect the joint rotations that are the controlled parameter of this model.

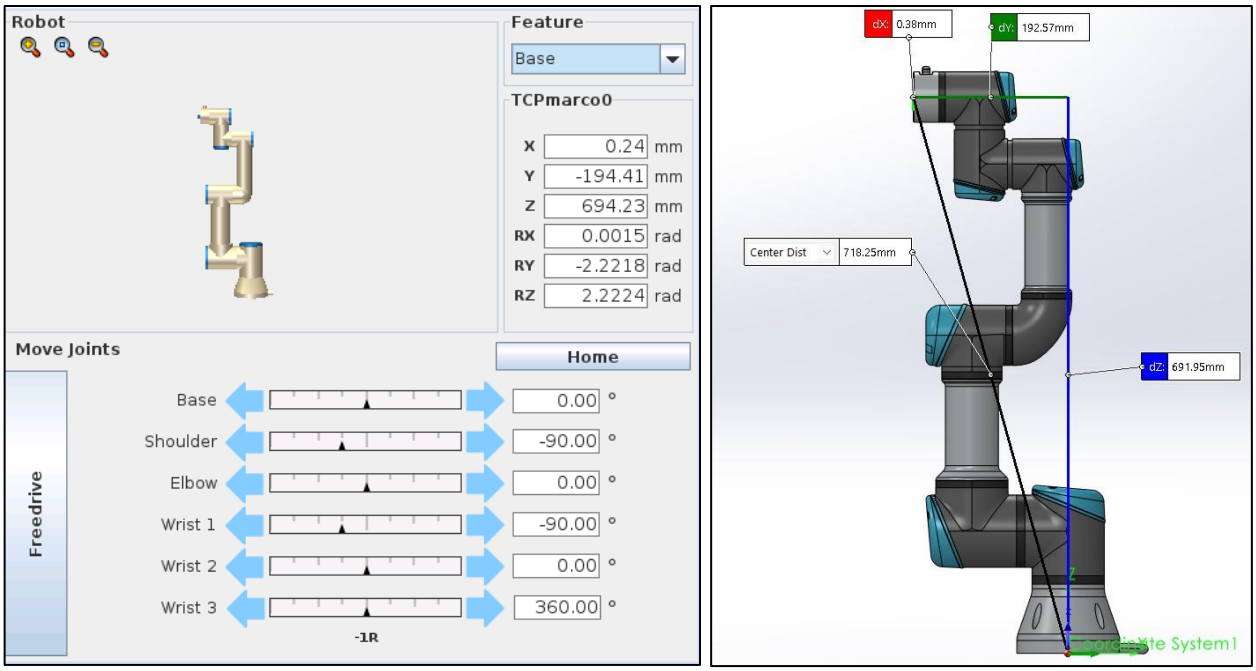

Figure 3. 21 - UR3 step file discrepancies

Those limitations and their impact on the results will be further discussed in *chapter 4*.

3.6 – RG2 Simscape model

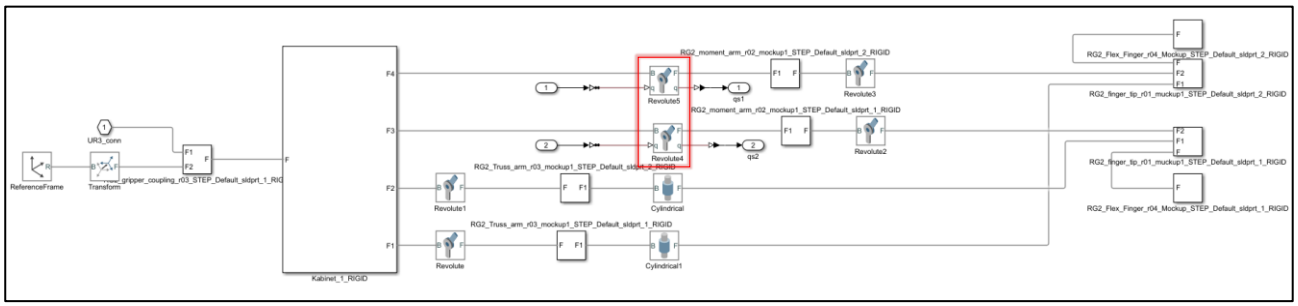

Figure 3. 22 - RG2 Simscape model

3.6.1 – Control system

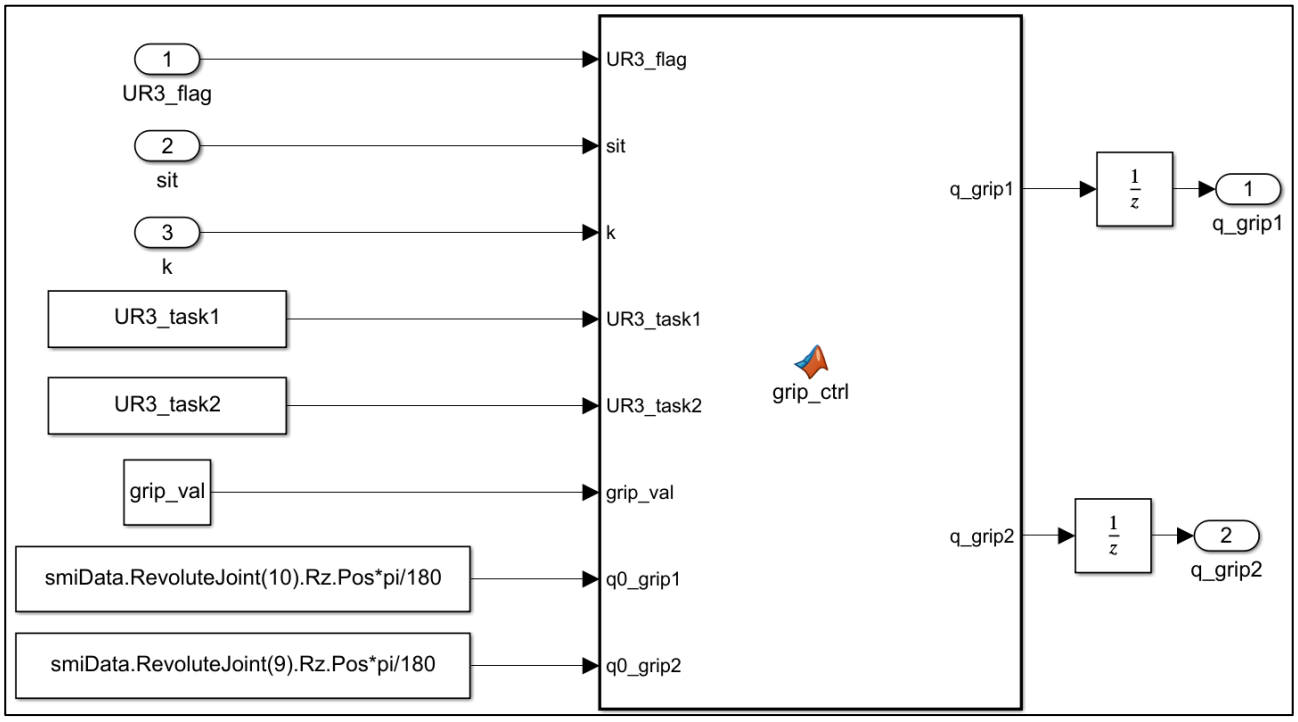

Figure 3. 23 - RG2 control system

To simulate the gripping action during the task, a basic controller that communicates with the UR3 and MIR100 controllers has been implemented.

As we can see in *figure 3.23* and in the following lines of code contained in the function block, the input signals are:

- UR3_flag : with this flag we make sure that the RG2 operates only when the UR3 is enabled;
- sit : since different tasks may be executed during the same mission, the gripper might open or close depending on it. Thus, it is required for its controller to know in which task it is being activated;
- k : the index value passed from UR3 controller. Thanks to the communication of this value, RG2 controller will know when the UR3 has reached the required pose and based on that it will send to the Simscape model the rotation value of the joints;
- Tasks : the gripper reads the same sequence of poses of the UR3. Doing so, it will know if it is required to open or close depending on the UR3 actual pose;

- grip_val : this value is defined as the angle that the highlighted joints in *figure 3.11* must rotate in order to open/close the gripper;
- q0_grip1 and q0_grip2 : those are the standard positions of the RG2 that will be sent by controller or overwritten when motion is required.

```
function [q_grip1, q_grip2] = grip_ctrl(UR3_flag, sit, k, grip_val, UR3_task1,
UR3_task2, q0_grip1, q0_grip2)

q_grip1 = q0_grip1 - grip_val;
q_grip2 = q0_grip2 + grip_val;

if (sit == 2 && UR3_flag == 1)
    if k < length(UR3_task1(1,:)) && k >= 1
        if UR3_task1(end,k) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    elseif k < 1
        if UR3_task1(end,1) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    elseif k >= length(UR3_task1(1,:))
        if UR3_task1(end,end) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    end

elseif (sit == 3 && UR3_flag == 1)
    if k < length(UR3_task2(1,:)) && k >= 1
        if UR3_task2(end,k) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    elseif k < 1
        if UR3_task2(end,1) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    elseif k >= length(UR3_task2(1,:))
        if UR3_task2(end,end) == 0
            q_grip1 = q0_grip1;
            q_grip2 = q0_grip2;
        end
    end
end
```

In these lines of code it is possible to better understand how the controlling signals are elaborated and sent.

4 – Case study results

To provide a practical use of this simulation system, a Pick and Place task has been designed. Specifically, the MiR100 will have to move from its charging station to a hypothetic 'assembly station'. Once position and orientation are the desired one, the UR3 will pick an object from that station.
At this point, MiR100 will start moving again toward the 'unloading station', where UR3 will place the object previously picked.
Finally, MiR100 will go back to its charging station.

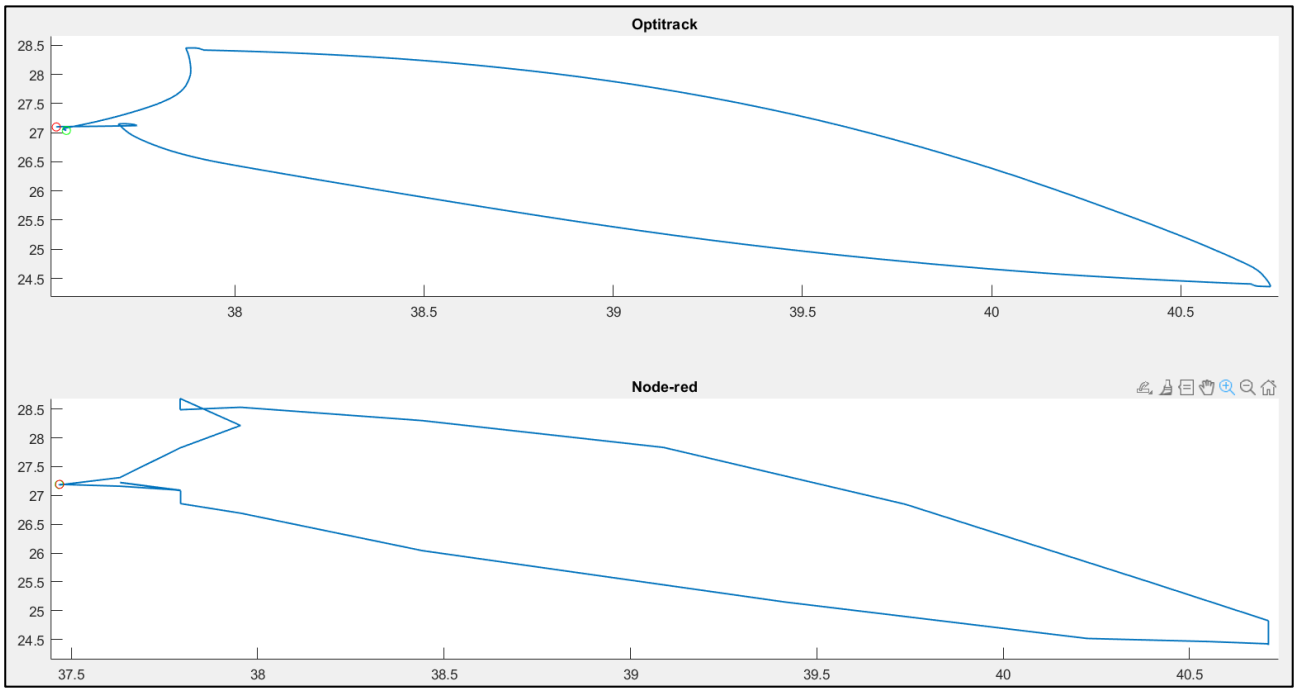

Figure 4. 1 – MiR100's positions registered from OptiTrack and Node-red on and XY plane

In *figure 4.1* the expected itinerary to be followed by the DS is shown.

The upper one has been registered through OptiTrack, while the other plot has been taken directly from the robot control-box through Node-red.

In these paragraphs we will discuss the output data obtained from the simulation of the task comparing them with respect to the input acquired from the real robots as we already explained in *Chapter 2*.

All the results are analyzed from three different systems point of view (Node-red, OptiTrack and Simulation) with respect to the ideal reference positions and orientations.

Before entering the result discussion, it is important to specify that the real utility of OptiTrack would be to compute accuracy, precision and repeatability studies on the robot that are not the objective of this thesis. Thus, what we are about to examine, in particular the plots, intend to give a tangible explanation of the ability of the Digital Shadow of behaving as the real robot. While, the analysis on the numerical results are to confirm the model perks and limitations previously mentioned.

4.1 – MiR100 results

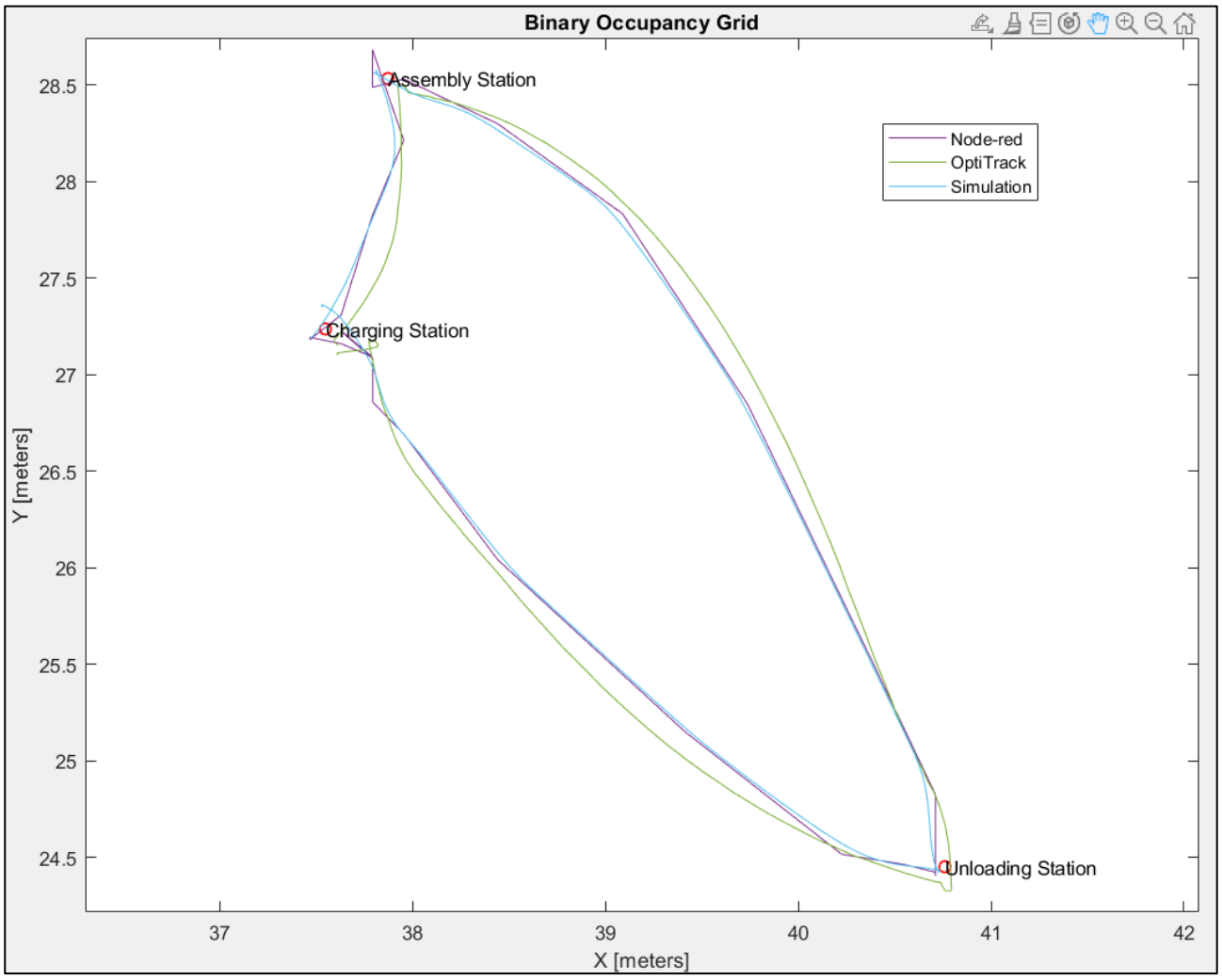

Figure 4. 2 - MiR100 path plotted on the map

In *figure 4.2* the trajectories registered with Node-red and OptiTrack are plotted and compared with the one generated by the simulation system.

We can notice that since the simulation system took its data from Node-red, it is more similar to its trajectory.

On the other hand, OptiTrack measurement is the most precise between the three, but it is not composed of data coming directly from the robot control box such as the Node-red ones. In other words, it can be used as an external check.

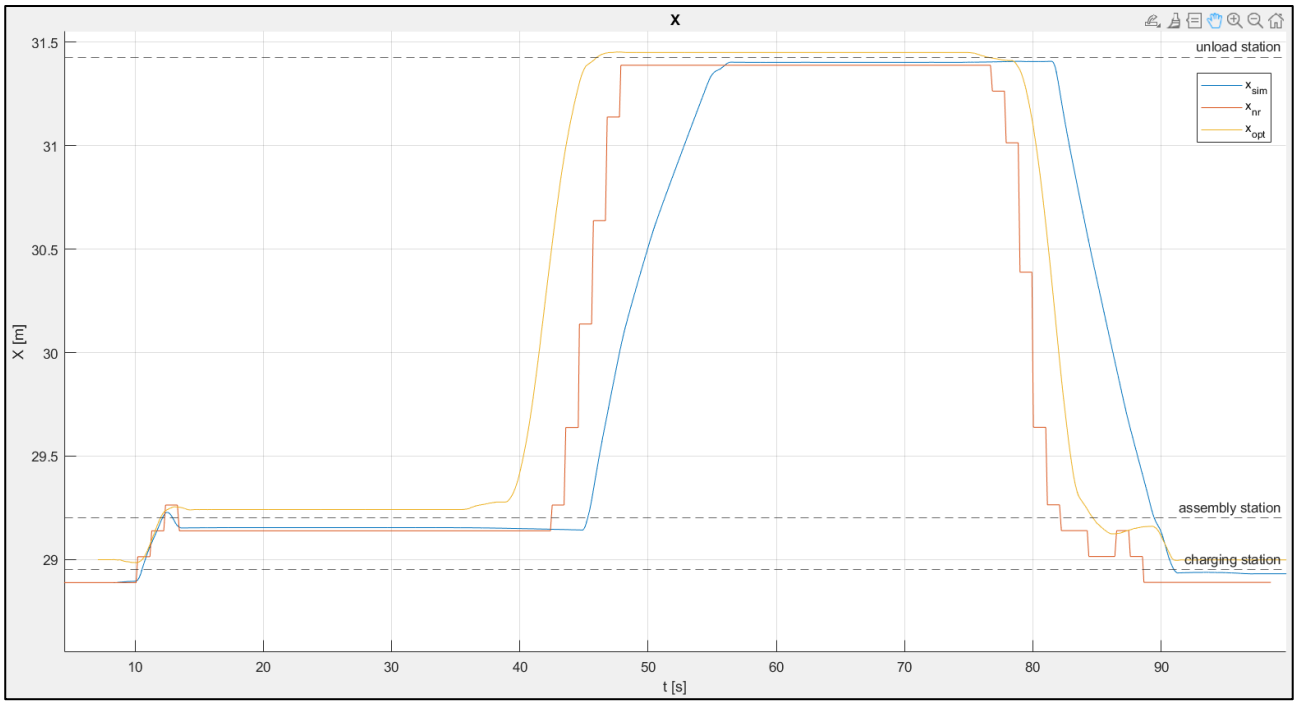

Figure 4. 3 – Coordinate X plot in time

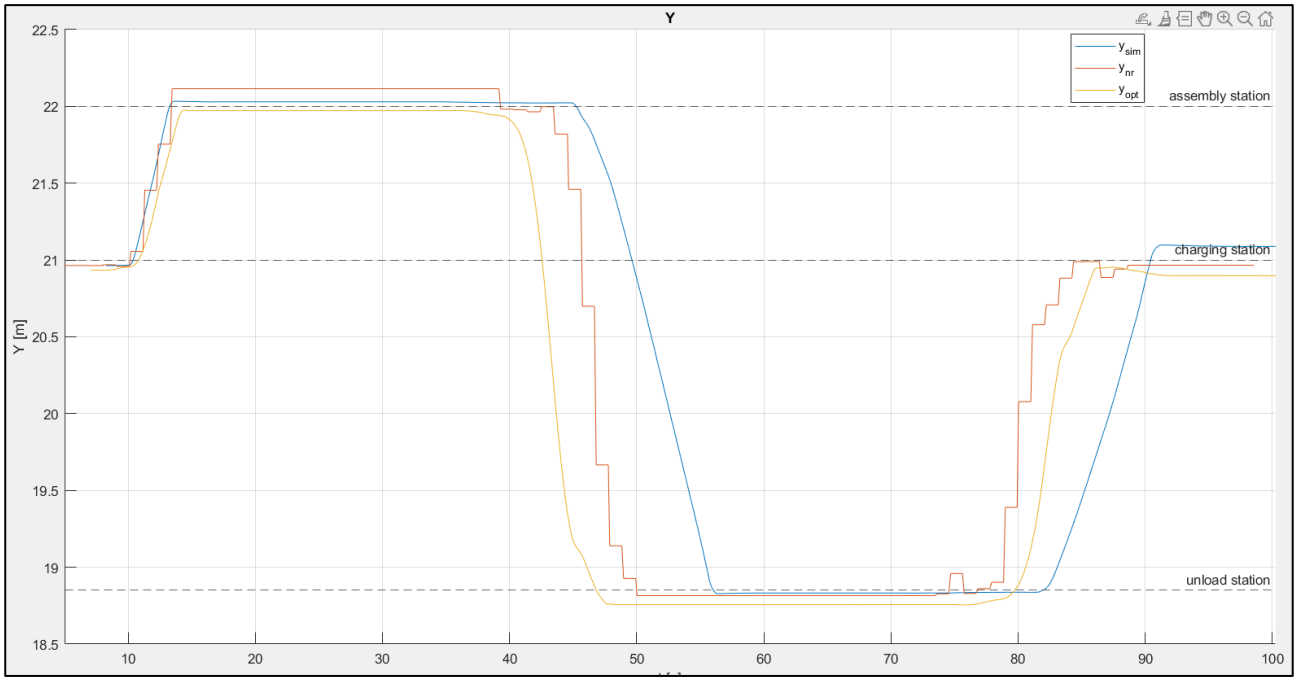

Figure 4. 4 - Coordinate Y plot in time

Referring to the figures above, we can see how the simulation system tries to repeat the positions covered by the Node-red data. It does, but not with the same speed. In other words, as we previously specified, we realized a simulation able to follow the poses of the real robots, but since some critical data and the exact control algorithm is not given, we could not guarantee the same timing.

Anyway, the time shifting is not so big since rarely the real robot goes at full speed during a task. The important result is that, aside from the timing, the virtual robot is able to reach all the positions within an error tolerance specified in the algorithm generated to control it.

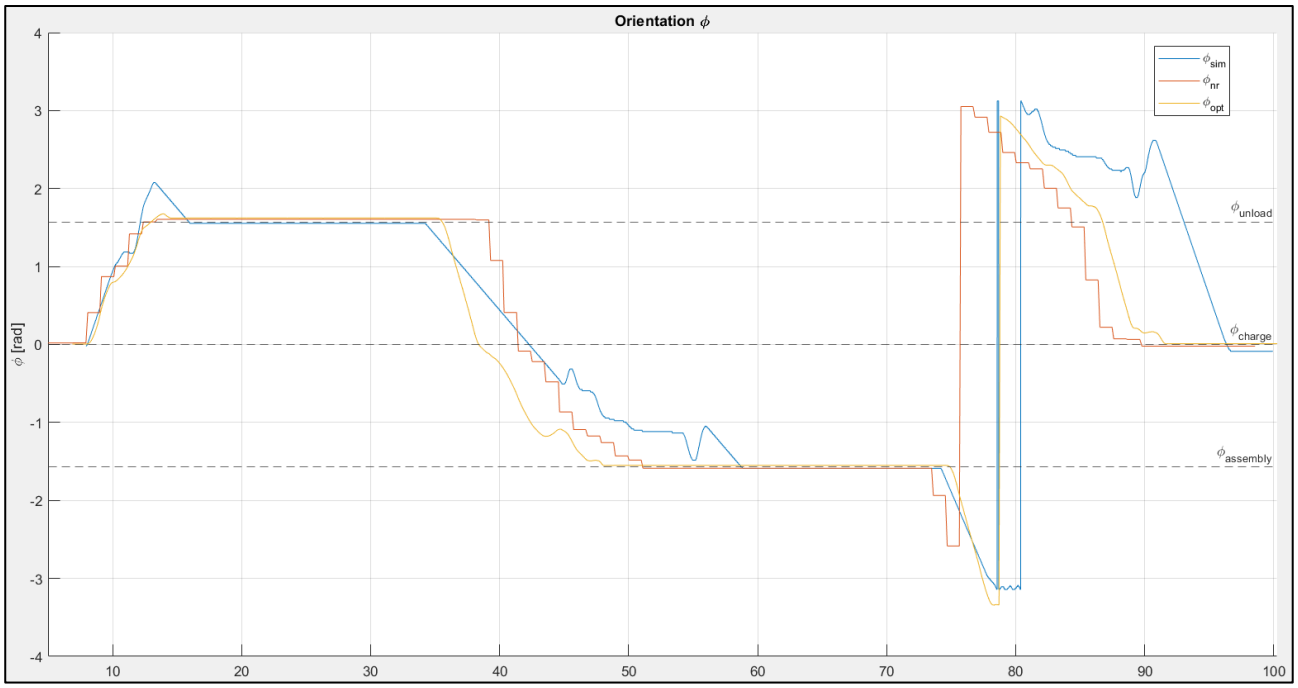

Figure 4. 5 - Orientation plot in time

The real orientation of the MiR100 is not duplicated by the simulation for the whole time, but just in the working stations. Specifically, the poses at the different stations have been passed to the control system with the correct position and orientation, not from Node-red.

The reason of the difference between the waypoints from Node-red and the stations poses imposed as exact coordinates is that the Digital Shadow will have to trace the same itinerary of the real one, but there would be no point in setting the error inside the stations coordinates since those errors are caused by the inaccuracy of the robot and the model already has its own positioning and orientation tolerance based on the inaccuracy of the physical object. Setting errors in the arrival stations would result in summing up all those imprecisions generating an even worst motion.

4.1.1 – Discussion about the resulting motion

In the following part, we will discuss the motion of the MiR100 model in the following way:

- Comparison of the dataset from Node-red with respect to the reference data registered from the software of the MiR100;
- Comparison of the dataset from the DS as result of trying to follow the same path of the real robot;
- A more exact comparison between the poses acquired with the OptiTrack system and the ideal reference data. As previously mentioned, OptiTrack poses are the real poses of the robot.

We specify that the real MiR100 has a positioning tolerance of $\pm 10.00\ cm$ stated in its software interface and a not specified orientation tolerance, while the DS has been set to $\pm 15.00\ cm$ and $\pm 0.02\ rad$. However, in the following results, it is shown that the positioning is always under this threshold.

	X [m]	Y [m]	ϕ [deg]
Charging station	28.950	21.000	0.000
Assembly station	29.200	22.000	90.000
Unloading station	31.427	18.852	-90.000

Node-red errors	X [m]	Y [m]	ϕ [deg]	ΔX [m]	ΔY [m]	Δϕ [deg]
Assembly Station	29.138	22.115	90.943	0.062	0.115	0.943
Unloading Station	31.389	18.815	-92.003	0.038	0.037	2.003
Charging Station	28.889	20.966	-1.239	0.061	0.034	1.239

Simulation errors	X [m]	Y [m]	ϕ [deg]	ΔX [m]	ΔY [m]	Δϕ [deg]
Assembly Station	29.152	22.033	91.000	0.048	0.033	1.000
Unloading Station	31.404	18.828	-89.000	0.023	0.024	1.000
Charging Station	28.931	21.088	-3.000	0.019	0.088	3.000

Optitrack errors	X [m]	Y [m]	ϕ [deg]	ΔX [m]	ΔY [m]	Δϕ [deg]
Assembly Station	29.240	21.975	88.790	0.040	0.025	1.210
Unloading Station	31.451	18.756	-92.930	0.024	0.096	2.930
Charging Station	28.997	20.898	0.600	0.047	0.102	0.600

Table 4. 1 - Position and orientation errors

In these tables, the charging station values refer to the pose of the MiR100 when it gets back to the starting point after executing its mission.

All the shown values are with respect to the MiR100 map origin on its own software.

To resume the pose error measured for this specific case:

Node-red errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.131	0.943
Unloading Station	0.053	2.003
Charging Station	0.070	1.239

Optitrack errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.047	1.210
Unloading Station	0.099	2.930
Charging Station	0.112	0.600

DS errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.058	1.000
Unloading Station	0.033	1.000
Charging Station	0.091	3.000

Table 4. 2 - Pose errors resume

From these tables, it results that from Node-red point of view, MiR100 was not always inside the positioning tolerance of $\pm 10.00\ cm$. In particular, it misses the assembly station. But for OptiTrack data, the real position acquired reaches values outside the tolerance for the return in the charging station and it is almost outside the tolerance when reaching the unloading one.

For what concerns the simulation, the positioning errors are abundantly under the estimated tolerance of $\pm 15.00\ cm$.

The errors computed with OptiTrack show that even if the robot control box senses the MiR100 in a certain position, the real error might differ, resulting in a worst positioning than the one reached by the simulation system.

Thus, as previously mentioned, the OptiTrack system, helped us to analyze the effective pose of the robot.

Different tasks executed on the simulation might give other results when computing the relative positioning error between the real and virtual system, but for sure, the simulation will always stay inside its predefined tolerance as long as the task is executable.

It is important to remark that this analysis on the real robot concerns just this task execution since different runs on the same task showed a very big positioning error variance, not always inside the tolerance stated in the robot datasheet.

To calculate a precise tolerance to be set as the DS parameter a statistic analysis might be convenient, but since it is not an object of this thesis we will set the worst case error from the data acquisitions of both Node-red and OptiTrack to run further simulations in a coherent way.

4.1.2 – Path optimization

Before discussing UR3 results, we have to explain to what those Simulation inaccuracies of MiR100 DS are due.

Since we are running a path realized by the real robot, the pose errors inside it will increase the flaws implicitly contained in our model generating an even bigger positioning error.

Indeed, as we can see in this zoom-in of *figure 4.2*, the simulated robot follows the Node-red path to reach the stations, but since the real robot might overshoot the desired location or stop before, the DS will follow that path until entering the positioning threshold and then start the orientation phase in wrong position.

This happens since the virtual robot follows the real path from Node-red but knows where the exact position of the station is. If instead of setting the exact location as goal, we put the wrong pose achieved by the real robot the DS would perfectly copy the real robot, but it would be useless in the task execution.

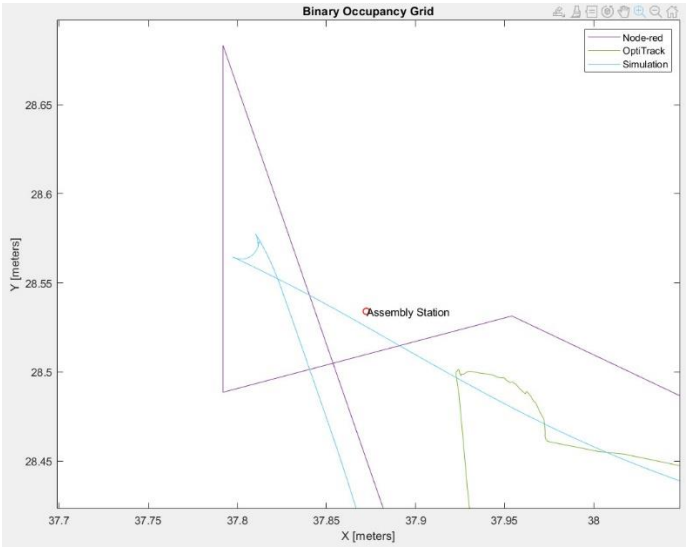

Figure 4. 6 - Zoom in of figure 4.2

Indeed, in this case, we see that the Simulation (light-blue) follows the Node-red path (purple) trying to get to the arrival point, or the perceived station from Node-red point of view, but as soon as the DS enters the positioning tolerance of the assembly station its control algorithm starts the orientation phase, stopping it in the wrong position.

From the collected data in *Table 4.2*, we can see that the error is still inside the tolerance, but we can decrease it even more if instead of running past data collected from Node-red, we design a path free from the errors of the real robot.

The design will be executed on this case study, but in a general approach in which we do not have to perform a run-in of the DS to check its behavior, we can directly design everything in the virtual environment and then perform it on the physical object.

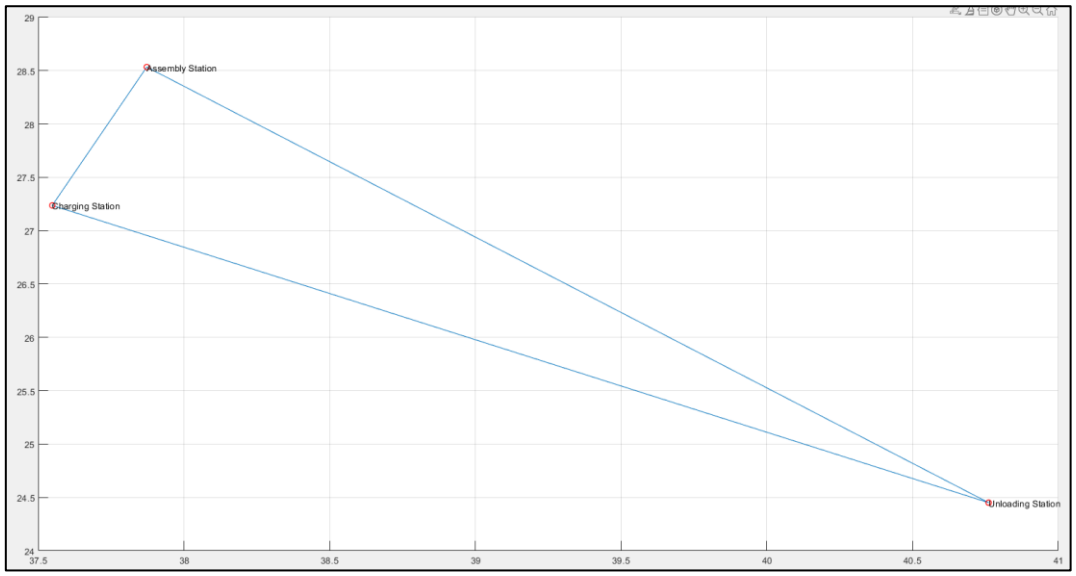

Figure 4. 7 - Optimized path

In *figure 4.7* it is shown a simple path, in which the checkpoints are the same. The main difference with respect to the previously followed one, is that the robot is required to stop at the stations and there is no more the overshoot performed by the real MiR100.

Moreover, since we are designing the path ourselves, the $\phi_{leaving}$ can be set at the optimal values to allow the robot to depart from each station without leaving the designed path. This way, we avoid the time wasting due to the fact that the real robot, even in absence of obstacles on the path, performs curved maneuvers between the stations.

Once the task has been tested in the simulation environment, we proceeded in running it on the real MiR100.

To do so, we wrote the same optimized task on the physical object interface constraining its motion to be equal to the desired route.

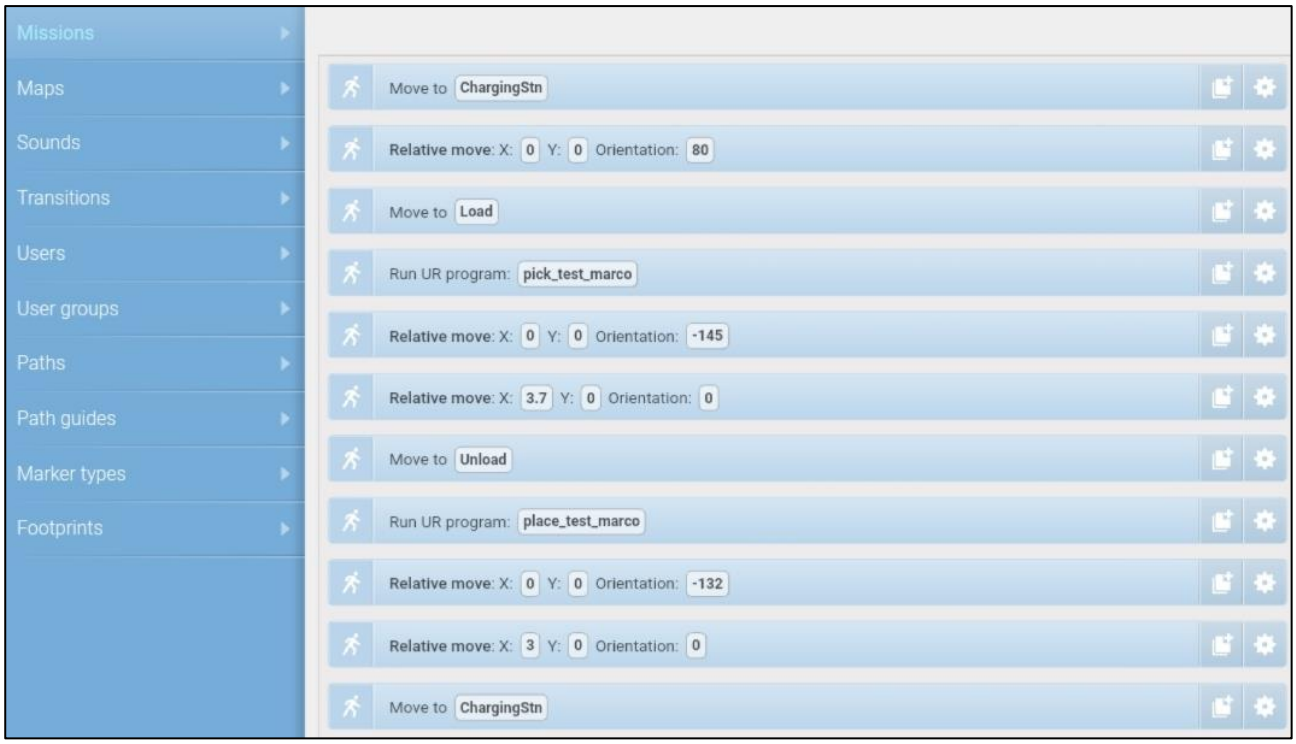

Figure 4. 8 - Modified MiR100 task

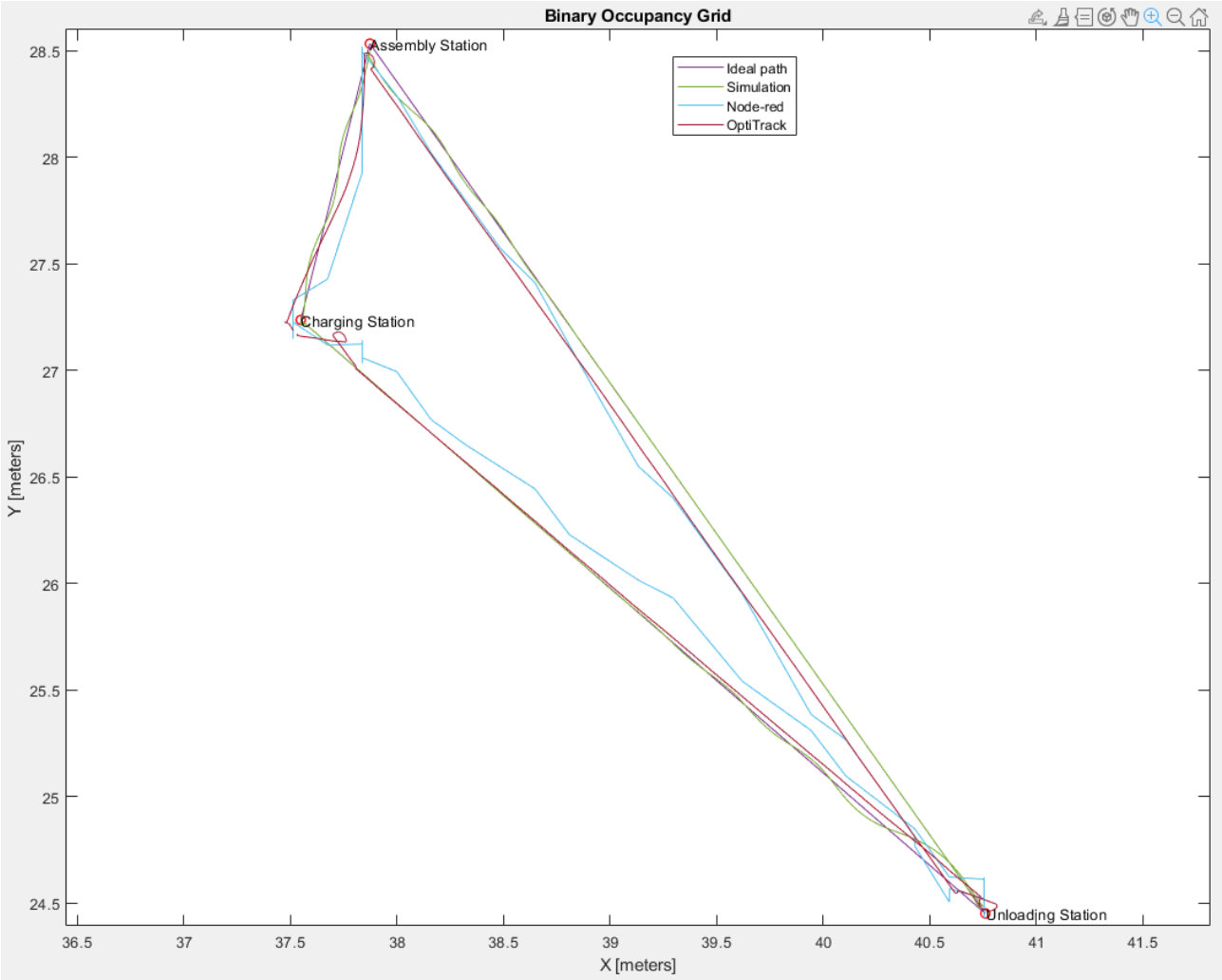

Figure 4. 9 – Run on an ideal path

Node-red errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.030	0.047
Unloading Station	0.017	1.518
Charging Station	0.070	2.184

Optitrack errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.041	0.304
Unloading Station	0.058	0.818
Charging Station	0.053	1.745

Simulation errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.028	0.000
Unloading Station	0.024	1.000
Charging Station	0.018	1.000

Table 4. 3 - Positioning errors

From these tables we can see that on an optimized path in which the DS points toward the right station location and not toward the point in which the physical object stopped, the overall errors are smaller, not only in positioning, but also in orientation. This happens because now the DS is following its own path and not a registered one in which past issues are considered.

Furthermore, it is observable that a constrained straight motion in which we ask the real robot to avoid its useless curvatures when no obstacles are present, implied an improvement on the results in the real world.

Indeed, we can notice how the positioning errors are decreased. Yet, some orientation issues are present but still inside the tolerances.

For what concerns the positioning, the resulting values are more similar in all three acquisition systems than those in the previous run, with an average relative error around $0.024\ m$.

The relative error of the orientation even if we increased the tolerance of the DS is around $0.534°$, showing how unpredictable this error can be.

Now that we know the error of the MiR100, given an optimized path, it is possible to run multiple simulations in different scenarios increasing the virtual robot tolerance to match the precision revealed by OptiTrack to simulate not an ideal behavior, but a realistic one that keeps into account the robot flaws. In this way, we are able to make concrete analysis on simulated tasks.

It is important to specify that even if this modification of the task does not take in consideration the presence of moving obstacles along the path, we can still consider it a general approach since what we are trying to optimize is the arrival and leaving in different stations.

Indeed, we can demonstrate that even if the ideal path is left due to the presence of an object and subsequently reacquired, the DS is able to get to the desired station within the pre-defined tolerance.

To do so, we start from the ideal optimized path and we add some waypoints between the stations to alter the route as if there was some obstacles who required the virtual robot to leave the predefined path and to re-enter it once the obstacle has been passed.

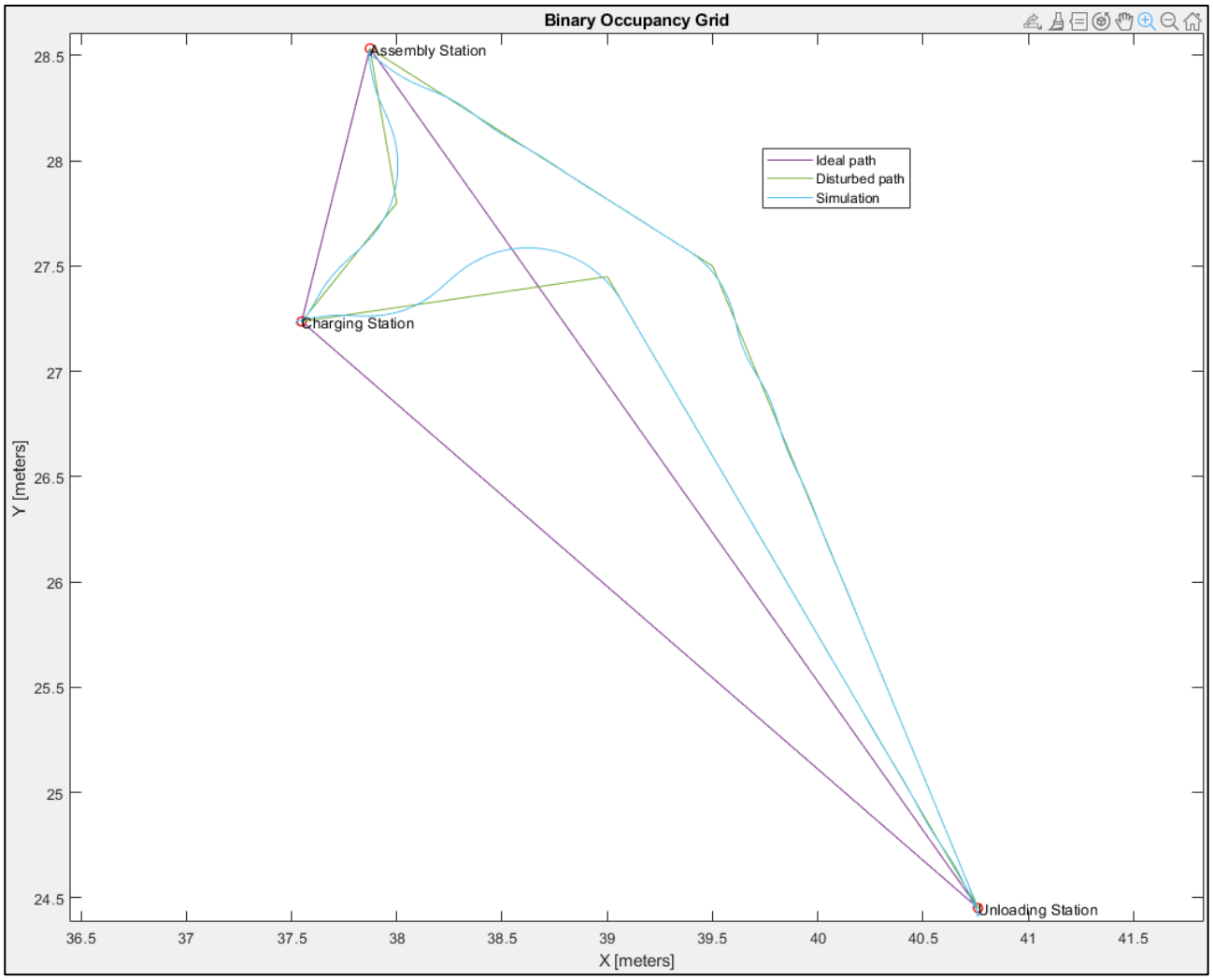

Figure 4. 10 - Simulation result on disturbed path

Simulation errors	Δ(X,Y) [m]	Δϕ [deg]
Assembly Station	0.009	0.000
Unloading Station	0.032	1.000
Charging Station	0.018	0.000

Table 4. 4 - Positioning errors

As a result, the Digital Shadow arrived at the desired stations with the correct pose even after performing a perturbed route.

In conclusion, we verified that the designed Digital Shadow, once calibrated to have the same flaws of the real robot, can be exploited to run different missions, optimize those and once we are sure of the results, run the real robot in the same condition, showing how powerful this model can be.

4.2 – UR3 results

In the following chapter, we will analyze the TCP trajectory with respect to the UR3 base of the Digital Shadow and the joints rotations during the tasks of the case study.

The analysis will start from the 3D positioning of the TCP in the workspace comparing each dataset with respect to the reference values of each pose acquired from the UR3 software. Then, we will conclude with the main analysis on the joints rotations that are the controlled parameter of the model.

4.2.1 – Discussion about the resulting motion in task 1

In the figures below, it is shown the TCP trajectory during the execution of the task 1 from each system point of view.

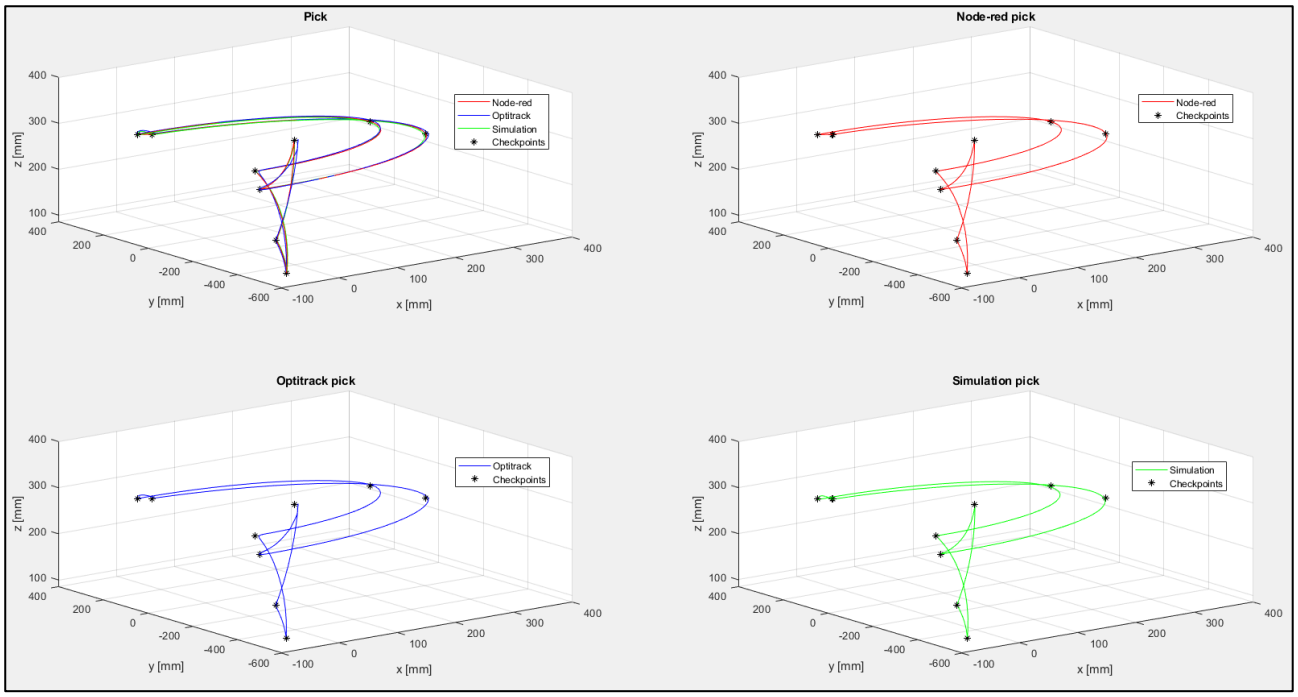

Figure 4. 11 – 3D plot of the TCP positions

From *figure 4.11*, we can see how the 3 trajectories almost overlap and, in particular, the Node-red one seems the most precise in reaching all the TCP reference positions.

Already from this first image, we can notice that the OptiTrack measurement shows a little error in the trajectory with respect to the checkpoints.
This is due to the fact that to measure the TCP trajectory during the task, a tool with a marker on its tip has been mounted on the UR3 (*figure 1.7*). But since the TCP is a virtual point, the marker results shifted with respect to it and even an error of a fraction of a millimeter can generate a wrong line missing the checkpoints of the task.
For this reason, OptiTrack measurements have been included just to verify the motion of the TCP and to underline this limit in the application for the UR3.
Anyhow, this is not a problem since we exploited OptiTrack to analyze the error of MiR100 that is less reliable than the UR3 that we know has been designed to be very precise.
For a better understanding of the outcome, we report the plots in each standard 2D plane:

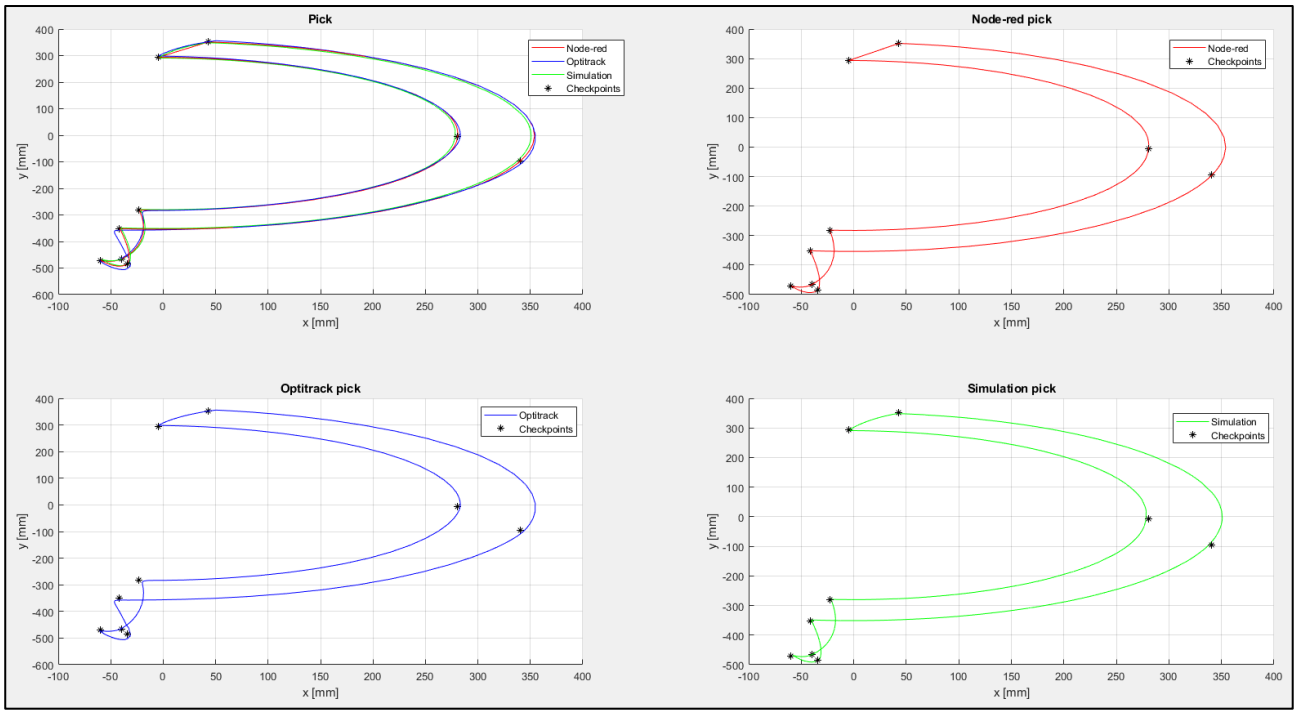

Figure 4. 12 - TCP position on plane XY

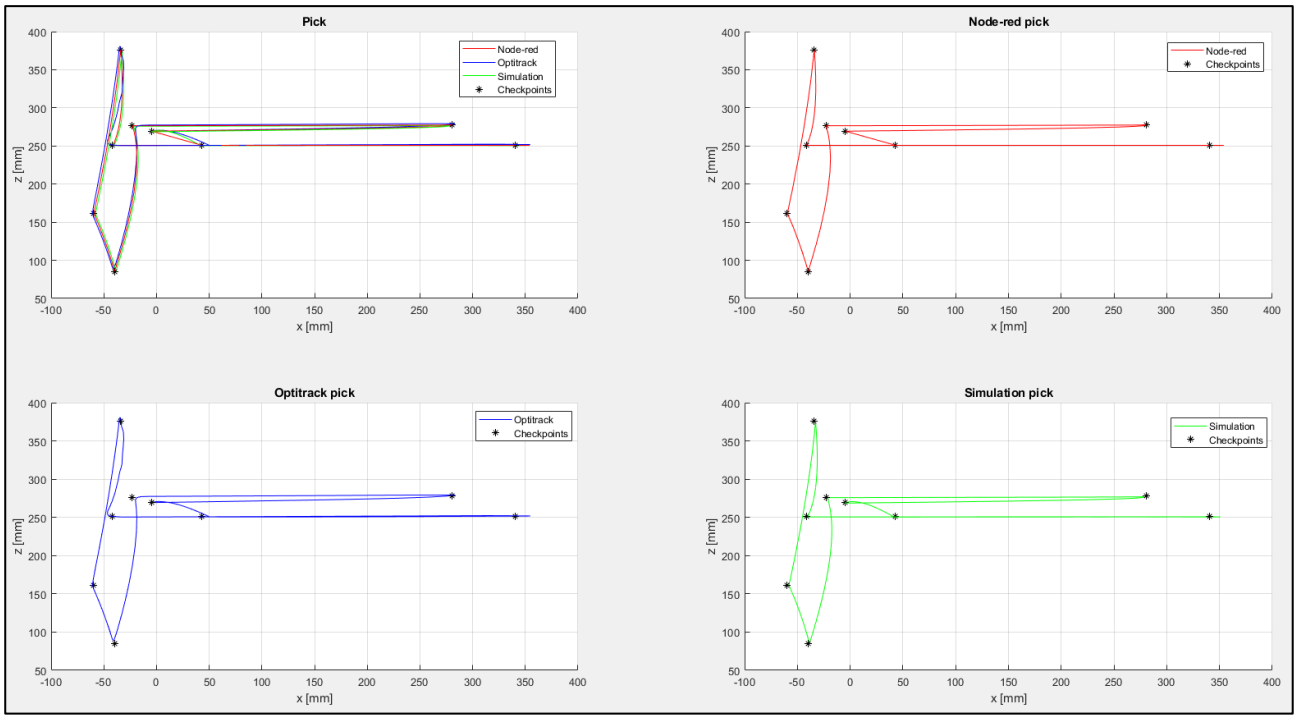

Figure 4. 13 - TCP position on plane XZ

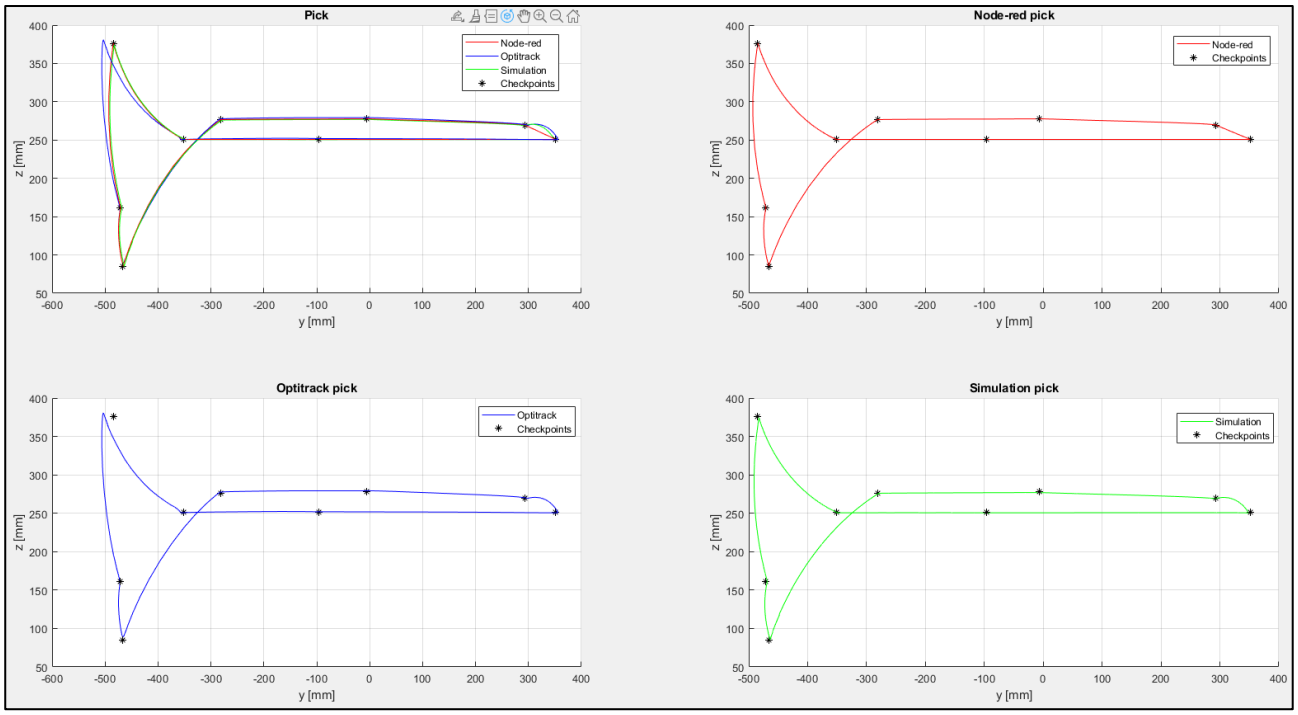

Figure 4. 14 - TCP position on plane YZ

Once again, we specify that this is not a statistic analysis, but the discussion about the results of a restricted number of runs, mainly focused on the behavior seen from three different systems.

In the UR3 datasheet, the only data about the positioning error is related to the repeatability ($\pm 0.1mm$), but since we are now performing a simulated analysis, in which the outcome with same conditions will always be the same, we will limit our study the TCP positioning error and joints angles.

In particular, we remark that the controller of the digital UR3 has been designed in order to follow the same configurations of the real robot keeping the error within an interval of $\pm 0.004\ rad$.

However, the TCP error will be analyzed to understand how good the digital UR3 precision is with a controller designed to replicate the motion of the real robot.

In the following tables, we resume the data concerning the TCP position:

Task1	X [mm]	Y [mm]	Z [mm]
1	42.900	351.360	250.850
2	340.780	-95.610	250.840
3	-41.660	-351.460	250.870
4	-34.130	-484.660	376.190
5	-59.580	-470.810	161.460
6	-39.860	-466.170	85.410
7	-23.000	-281.470	276.530
8	280.450	-6.110	277.700
9	-4.490	293.620	269.360
10	42.850	351.360	250.850

Node-red	X [mm]	Y [mm]	Z [mm]	ΔX [mm]	ΔY [mm]	ΔZ [mm]
1	42.800	351.400	250.800	0.100	-0.040	0.050
2	340.800	-95.700	250.800	-0.020	0.090	0.040
3	-41.500	-351.400	250.900	-0.160	-0.060	-0.030
4	-33.900	-484.500	376.200	-0.230	-0.160	-0.010
5	-59.500	-470.700	161.400	-0.080	-0.110	0.060
6	-39.900	-466.100	85.400	0.040	-0.070	0.010
7	-22.400	-281.900	276.500	-0.600	0.430	0.030
8	280.500	-6.100	277.700	-0.050	-0.010	0.000
9	-4.500	293.600	269.400	0.010	0.020	-0.040
10	42.900	351.400	250.800	-0.050	-0.040	0.050

Digital Shadow	X [mm]	Y [mm]	Z [mm]	ΔX [mm]	ΔY [mm]	ΔZ [mm]
1	41.700	348.900	250.500	1.200	2.460	0.350
2	338.500	-94.000	250.500	2.280	-1.610	0.340
3	-40.400	-349.000	250.600	-1.260	-2.460	0.270
4	-32.900	-482.800	374.400	-1.230	-1.860	1.790
5	-58.000	-468.500	161.100	-1.580	-2.310	0.360
6	-38.500	-463.700	85.000	-1.360	-2.470	0.410
7	-21.900	-279.200	275.800	-1.100	-2.270	0.730
8	277.800	-5.200	276.900	2.650	-0.910	0.800
9	-5.000	290.800	268.700	0.510	2.820	0.660
10	41.600	348.800	250.500	1.250	2.560	0.350

Table 4. 5 – TCP positions on task 1

In the following image, we plot the joints angles value during the task execution.

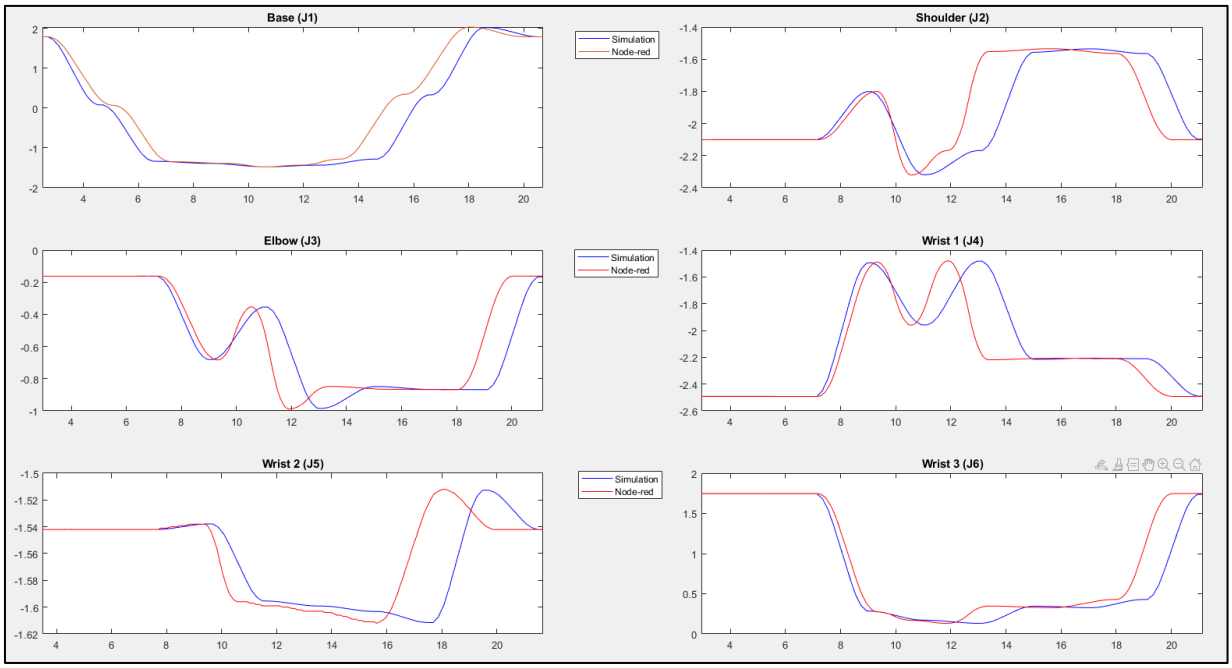

Figure 4. 15 - Joints angles plot

Task 1	Base [rad]	Shoulder [rad]	Elbow [rad]	Wrist1 [rad]	Wrist2 [rad]	Wrist3 [rad]
1	1.794	-2.101	-0.162	-2.492	-1.542	1.747
2	0.071	-2.101	-0.162	-2.492	-1.542	1.747
3	-1.344	-2.101	-0.162	-2.492	-1.542	1.747
4	-1.390	-1.800	-0.684	-1.489	-1.538	0.274
5	-1.474	-2.321	-0.353	-1.961	-1.596	0.169
7	-1.432	-2.168	-0.989	-1.479	-1.599	0.131
8	-1.279	-1.553	-0.848	-2.219	-1.603	0.347
9	0.346	-1.535	-0.867	-2.207	-1.612	0.327
10	2.036	-1.565	-0.868	-2.211	-1.512	0.430

Simulation	Base [rad]	Shoulder [rad]	Elbow [rad]	Wrist1 [rad]	Wrist2 [rad]	Wrist3 [rad]	ΔJ1 [rad]	ΔJ2 [rad]	ΔJ3 [rad]	ΔJ4 [rad]	ΔJ5 [rad]	ΔJ6 [rad]
1	1.794	-2.101	-0.162	-2.492	-1.542	1.747	0.000	0.000	0.000	0.000	0.000	0.000
2	0.076	-2.102	-0.161	-2.496	-1.542	1.747	-0.004	0.001	-0.001	0.004	0.000	0.000
3	-1.340	-2.102	-0.161	-2.496	-1.542	1.747	-0.004	0.001	-0.001	0.004	0.000	0.000
4	-1.390	-1.800	-0.686	-1.491	-1.538	0.278	0.000	0.000	0.002	0.003	0.000	-0.004
5	-1.473	-2.318	-0.357	-1.958	-1.595	0.173	0.000	-0.004	0.004	-0.003	-0.001	-0.004
7	-1.432	-2.169	-0.987	-1.483	-1.599	0.132	0.000	0.001	-0.001	0.004	0.000	-0.001
8	-1.280	-1.555	-0.852	-2.215	-1.603	0.343	0.001	0.003	0.003	-0.004	0.000	0.004
9	0.341	-1.534	-0.870	-2.209	-1.611	0.327	0.004	-0.001	0.003	0.001	0.000	0.000
10	2.032	-1.564	-0.871	-2.212	-1.513	0.429	0.004	-0.001	0.003	0.001	0.001	0.001
11	1.794	-2.102	-0.162	-2.496	-1.542	1.743	0.000	0.001	0.000	0.004	0.000	0.004

Table 4. 6 - Joints angles values

From these tables, we can notice how the simulation is able to follow all the main poses of the motion with a precision of $\pm 0.004\ rad$.

However, from the plots, we can see that the timing with which these poses are reached is slightly different. Aside from this aspect that has already been discussed in *paragraph 3.5.2* , we can state that the model is able to replicate the poses sequence of the real robot with good precision.

For what concerns the TCP positioning with respect to the base from the data contained in the tables above:

- Node-red average position error: $\Delta_{xyz} = 0.105\ mm$;
- Simulation average position error: $\Delta_{xyz} = 2.677\ mm$;

As previously mentioned, the OptiTrack results, aside from the motion study, cannot be used from a numerical point of view.

From these values we can notice how the limitations abundantly discussed result in a big imprecision of the TCP virtual model, being about $2.677\ mm$.

This relevant error can be justified with the following reasons:

1) The parameters of the internal mechanics of the joints have been supposed and are not the exact ones of the real robot.
   We underline that the parameters have been calibrated with multiple tests until finding a compromise between a smooth motion and a stable one, but even the slightest difference with respect to the real values, multiplied for 6 the joints of the kinematic chain, can affect the outcome, both in position and in execution speed;
2) As we showed in *chapter 3.5.2*, there is a discrepancy between the real robot measures and the official .step file available on the Universal Robots site. For this reason, while the joints rotations are really accurate, the TCP position in the 3D space is not. The different measures of the joints of the 3D model are of the order of the millimeters, but while these differences do not affect the angles, they have a big impact on the TCP position.

In conclusion, we can state that the controller designed to simulate the robot is accurate and is able to make the DS execute the various poses as the real robot, but cannot be used to analyze the TCP position for this specific UR3 model.

However, if in a future development we wanted to design our own robot and use this controller knowing all the needed data and the exact measurements, this model would be even more precise.

4.2.2 – Discussion about the resulting motion in task 2

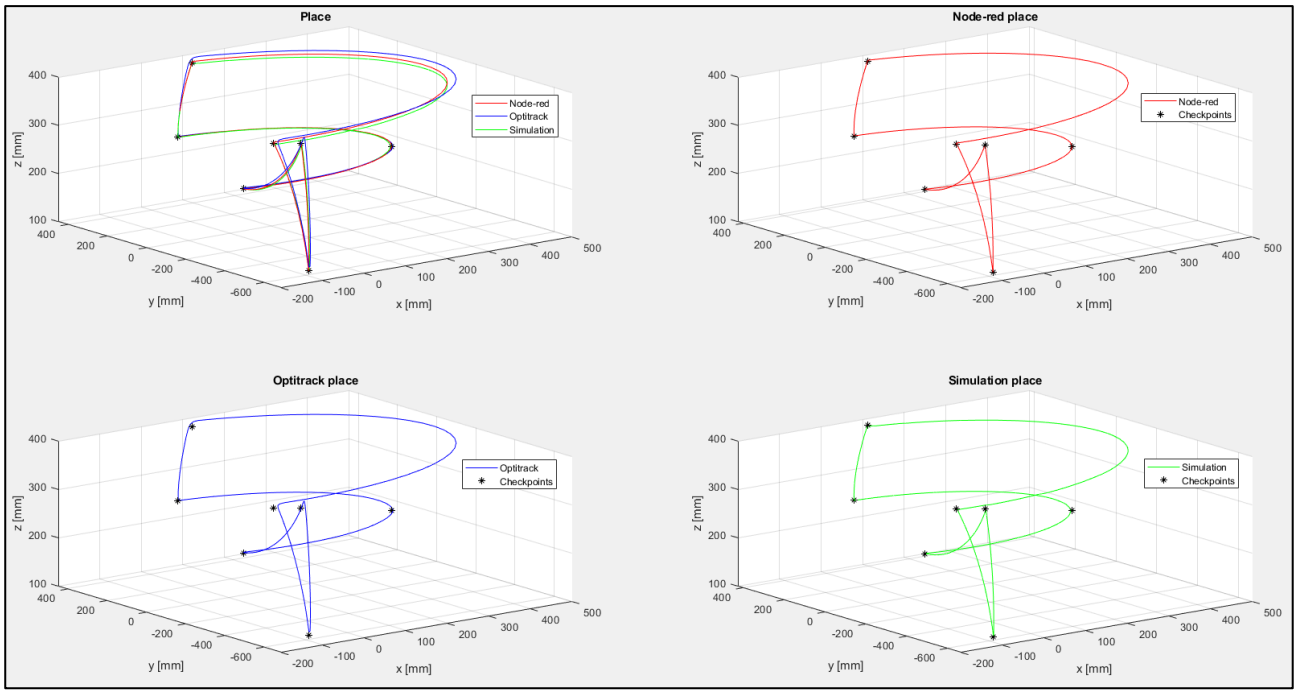

Figure 4. 16 - 3D plot of the TCP positions

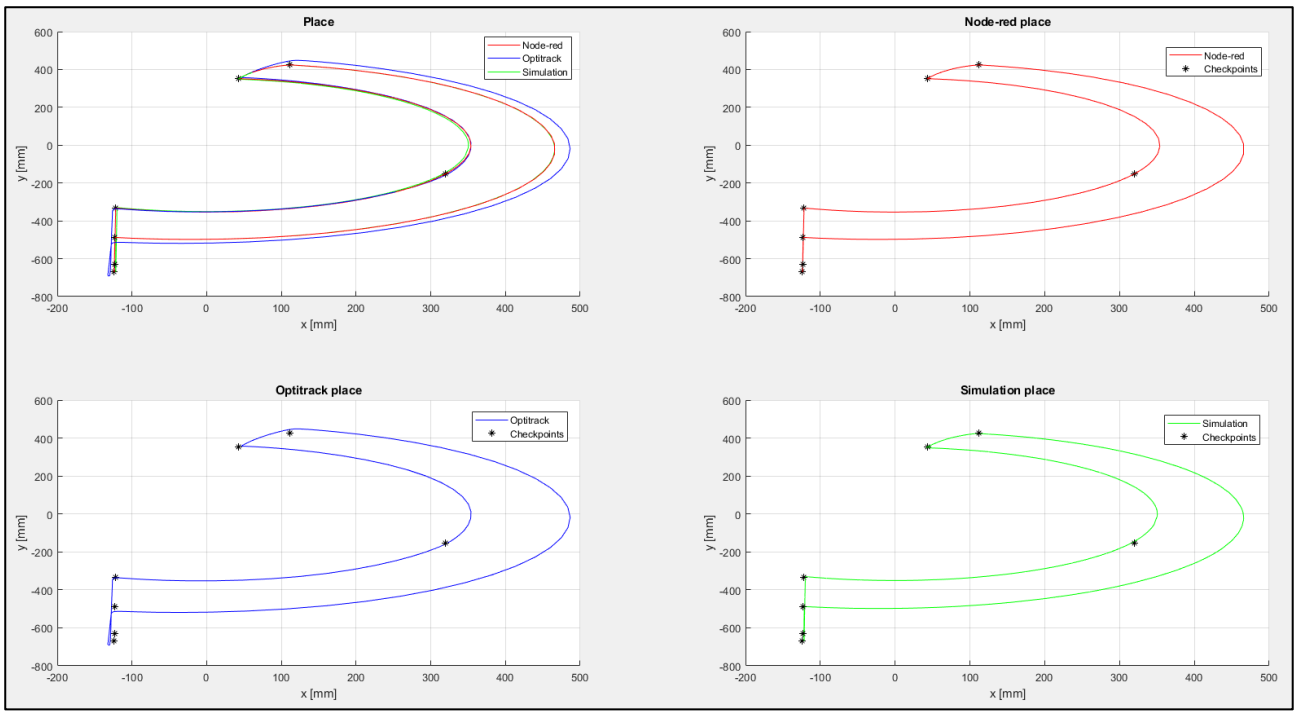

Figure 4. 17 - TCP position on plane XY

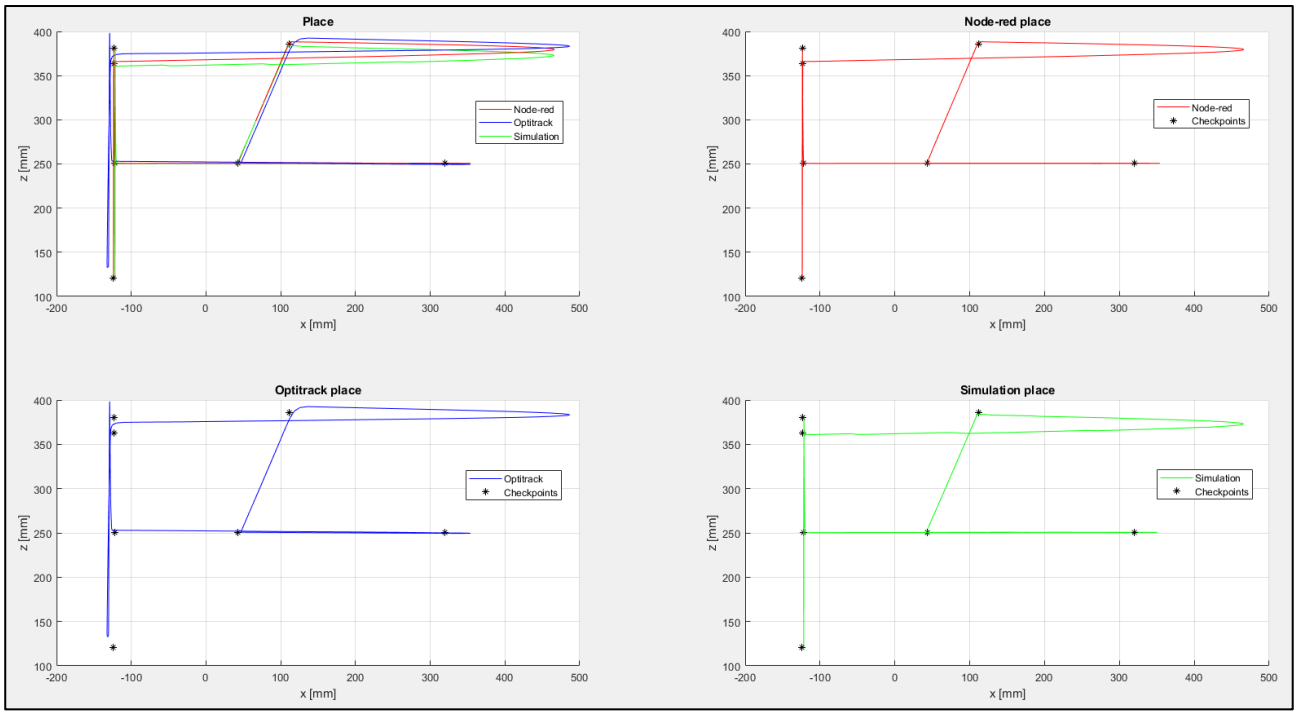

Figure 4. 18 - TCP position on plane XZ

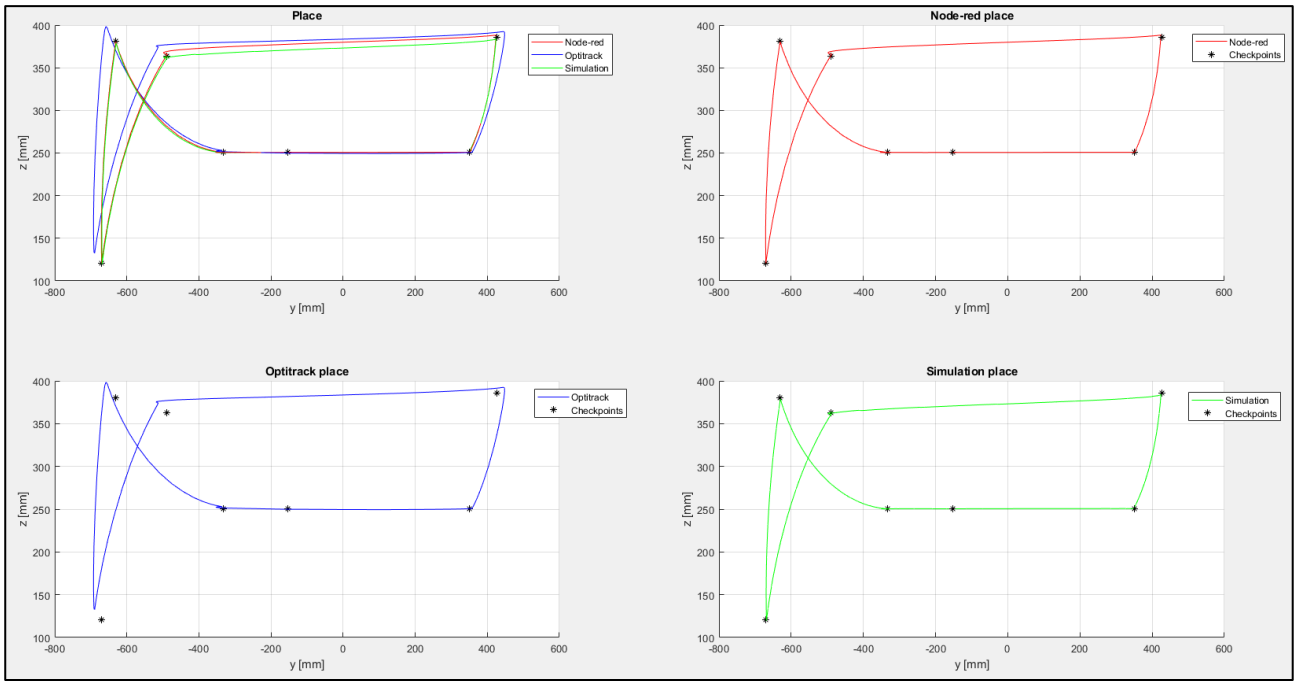

Figure 4. 19 - TCP position on plane YZ

Task2	X [mm]	Y [mm]	Z [mm]
1	42.900	351.370	250.830
2	319.340	-152.550	250.680
3	-121.840	-332.270	250.650
4	-123.550	-630.780	380.690
5	-123.860	-670.060	121.090
6	-123.860	-670.060	121.070
7	-123.090	-489.620	363.150
8	111.720	425.970	385.520
9	42.900	351.380	250.830

Node-red	X [mm]	Y [mm]	Z [mm]	ΔX [mm]	ΔY [mm]	ΔZ [mm]
1	42.900	351.400	250.800	0.000	-0.030	0.030
2	319.330	-152.520	250.700	0.010	-0.030	-0.020
3	-121.800	-332.500	250.700	-0.040	0.230	-0.050
4	-123.500	-630.500	380.300	-0.050	-0.280	0.390
5	-123.800	-669.900	120.800	-0.060	-0.160	0.290
6	-123.800	-669.900	120.800	-0.060	-0.160	0.270
7	-122.100	-487.600	365.900	-0.990	-2.020	-2.750
8	111.900	424.000	388.300	-0.180	1.970	-2.780
9	42.900	351.300	250.800	0.000	0.080	0.030

Digital shadow	X [mm]	Y [mm]	Z [mm]	ΔX [mm]	ΔY [mm]	ΔZ [mm]
1	41.700	348.900	250.500	1.200	2.470	0.330
2	317.400	-150.600	250.400	1.940	-1.950	0.280
3	-120.000	-330.100	250.300	-1.840	-2.170	0.350
4	-121.600	-629.800	378.900	-1.950	-0.980	1.790
5	-121.700	-668.500	119.900	-2.160	-1.560	1.190
6	-121.700	-668.500	119.800	-2.160	-1.560	1.270
7	-121.300	-488.100	361.600	-1.790	-1.520	1.550
8	110.100	424.200	384.100	1.620	1.770	1.420
9	41.700	348.900	250.500	1.200	2.480	0.330

Table 4. 7 – TCP positions on task 2

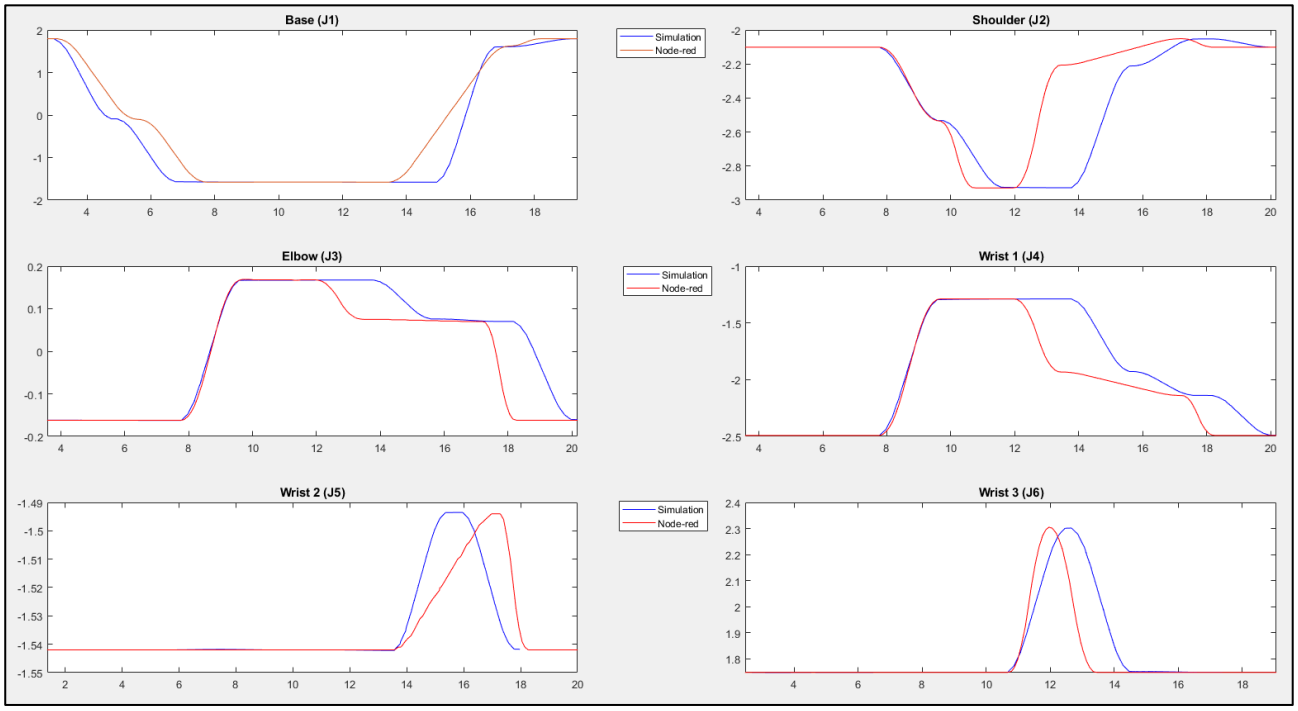

Figure 4. 20 - Joints angles plot

Task 2	Base [rad]	Shoulder [rad]	Elbow [rad]	Wrist1 [rad]	Wrist2 [rad]	Wrist3 [rad]
1	1.794	-2.101	-0.162	-2.492	-1.542	1.747
2	-0.101	-2.101	-0.162	-2.492	-1.542	1.747
3	-1.577	-2.101	-0.162	-2.492	-1.542	1.747
4	-1.577	-2.535	0.169	-1.288	-1.542	1.747
5	-1.577	-2.928	0.168	-1.289	-1.542	1.747
6	-1.577	-2.928	0.168	-1.289	-1.542	2.305
7	-1.578	-2.207	0.075	-1.933	-1.542	1.747
8	1.621	-2.051	0.070	-2.140	-1.493	1.747
9	1.794	-2.101	-0.162	-2.492	-1.542	1.747

Simulation	Base [rad]	Shoulder [rad]	Elbow [rad]	Wrist1 [rad]	Wrist2 [rad]	Wrist3 [rad]	ΔJ1 [rad]	ΔJ2 [rad]	ΔJ3 [rad]	ΔJ4 [rad]	ΔJ5 [rad]	ΔJ6 [rad]
1	1.794	-2.101	-0.162	-2.492	-1.542	1.747	0.000	0.000	0.000	0.000	0.000	0.000
2	-0.097	-2.102	-0.162	-2.495	-1.542	1.747	-0.004	0.001	-0.001	0.003	0.000	0.000
3	-1.573	-2.102	-0.162	-2.495	-1.542	1.747	-0.005	0.001	-0.001	0.003	0.000	0.000
4	-1.577	-2.537	0.171	-1.293	-1.542	1.747	0.000	0.001	-0.002	0.004	0.000	0.000
5	-1.577	-2.927	0.170	-1.292	-1.542	1.747	0.000	-0.001	-0.002	0.004	0.000	0.000
6	-1.577	-2.930	0.170	-1.292	-1.542	2.301	0.000	0.002	-0.003	0.004	0.000	0.004
7	-1.578	-2.210	0.078	-1.935	-1.542	1.751	0.000	0.003	-0.002	0.002	0.000	-0.004
8	1.617	-2.053	0.072	-2.144	-1.494	1.747	0.004	0.001	-0.002	0.004	0.000	-0.001
9	1.792	-2.102	-0.159	-2.492	-1.542	1.747	0.002	0.001	-0.003	0.000	0.000	0.000

Table 4. 8 - Joints angles values

In these last plots, we can see that the digital UR3 is able to achieve all the poses of the task, but it does not reach them with the same timing of the real UR3.

The TCP error is of the same order of Task1 for the same reasons already explained.

4.3 – MiR100 positioning effect UR3 motion

The TCP results described in the previous sections are all referred to the base joint in order to evaluate the fidelity of the DS for the UR3 alone. This analysis is useful in case we might want to apply the UR3 model on other platforms aside from the MiR100.

On the other hand, since we are currently speaking about a MoMa, we must discuss how the mobile robot precision might affect the manipulator actions.

What we could witness when moving the real MoMa is that not so rarely, the MiR100 arrived at the station so imprecisely that it resulted in the failure of the UR3 task. This issue was more evident when, in a picking task, the manipulator could not grasp the object.

This problem can be solved by applying a camera on the UR3 wrist and implementing a behavior able to compensate the MiR100 pose.

But since we focused on realizing a DS with what we had and based on the available MoMa, the previous analysis we made are more than enough to evaluate the fidelity of a DS that as a matter of fact is the assembly of two separate Digital Shadows.

We can state this because in a future development, the addition of a camera would make the MiR100 effect on UR3 less relevant. But since at the time it is not available, this topic could not be part of our study and we focused more on the DS fidelity for all the important features of the real MoMa.

Anyway, if we still want a graphic vision of how MiR100 uncompensated error affects the UR3 motion we can report the case study results showing the TCP locations with respect to the map origin reference frame instead of the base.

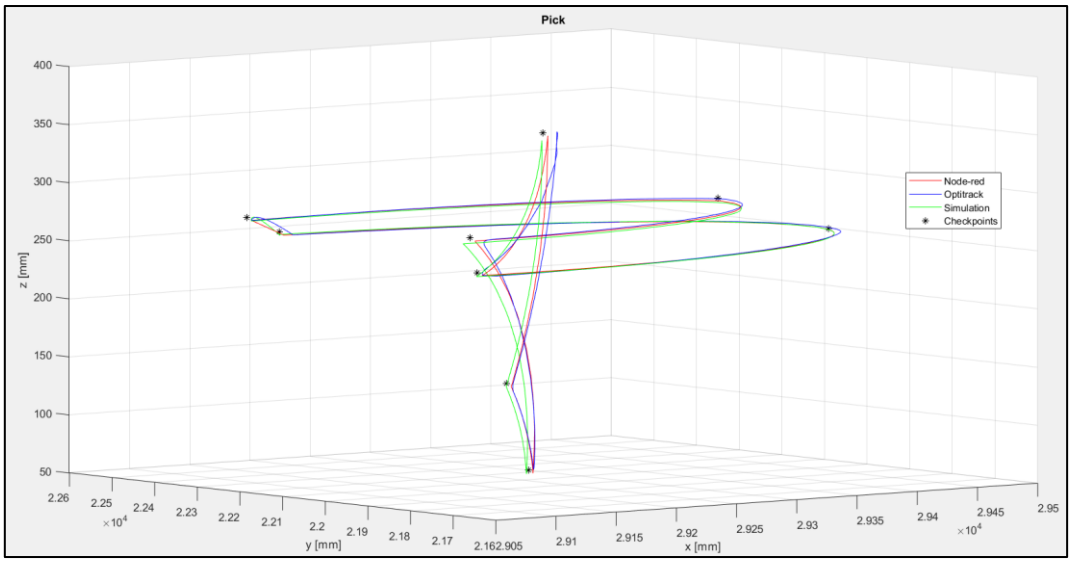

Figure 4. 21 - Task1 failure

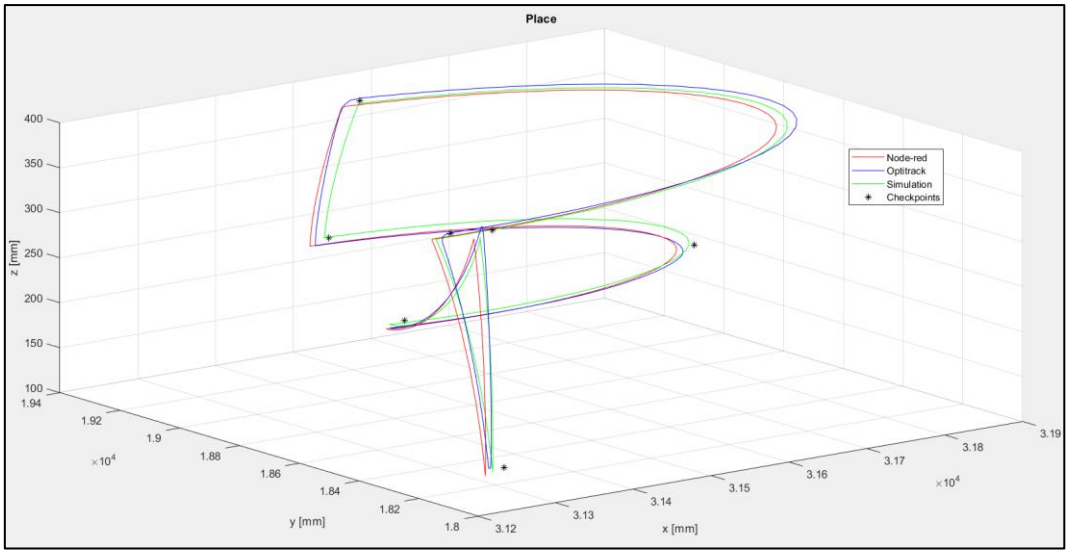

Figure 4. 22 – Task2 failure

In these two figures, to give a better idea about the effect of a MiR100 uncompensated error we report the case of a failed run of the case study we described earlier.

Here we can see how the MiR100 position error, that is of the order of the centimetres, might lead to a total failure in gripping and releasing the object in the desired position if the component to be manipulated has restricted dimensions, not to mention the possible damages to the surrounding areas in case the UR3 starts to work in a crowded environment in the wrong position.

Of course the plots are not overlapping because the Digital Shadow and the real robot are running with same position threshold but behave both in a random way as they are expected to.

Indeed, since in the previous chapters we demonstrated the DS fidelity, this model can now be used to study all the scenarios analysing only the virtual robot results, confident that its range of error will always be the same of the real one, avoiding possible catastrophic consequences in case of failure. Then, after the task to be run has been confirmed to be safe, we can run the real robot and analyse the outcome.

5 – Conclusions

From the results obtained we can confirm that the Digital Shadow has been calibrated to behave as the real robot at the best of the possibilities keeping in mind all the limitations and issues stated in the previous chapters. Further optimizations of the model can certainly be achieved in future developments of the project.

Consequently, the model is good enough to firstly simulate MiR100 trajectories and UR3 tasks in the virtual environment and, only after those have been optimized, execute them on the real MoMa.

This way, it is possible to exploit the digital MiR100 as a powerful tool in case of a point-to-point trajectory optimization in absence of a real robot or in an attempt of avoid using it when not necessary.

The same potential use, focused on the optimization of a sequence of poses to execute various tasks can be thought for the UR3, especially when it comes to possible self-harming motions.

In conclusion, we can state that the realization process for a Digital Twin shown in this Thesis can be used for different systems, especially in case of open-system ones in which we can develop more than a Digital Shadow as seen.

Future works on the realized DS could be, for example:

- Refinement of the error threshold and analysis of the events and maneuvers that lead the MiR100 to reach the goal positions with a certain error through an accurate statistical analysis;
- Improve the path generation and following code in order to better approximate the mobile robot, especially in how it chooses a maneuver instead of another;
- The study of a case in which mobile obstacles, not present on the map when the mobile robot designed its path, appear in the trajectory. This study would require the implementation of all the proximity sensors for the Simscape multibody model and the related algorithms;
- As we already wrote, the addition of a camera to the arm and so to the DS with an algorithm needed to recalibrate the UR3 motion to compensate the MiR100 position error;
- The addition of other Digital Shadows in the virtual laboratory to generate more complex tasks virtualizing a whole production process in which more robots are involved;
- Apply this design process to a open-system robot to generate a DT.

These could be some ideas, but the use of a Digital Shadow and even more of a Digital Twin have infinite possibilities.

Bibliography

- ‘A virtual commissioning based methodology to integrate digital twins into manufacturing systems’ - G.Barbieri, A.Bertuzzi, A.Capriotti, L.Ragazzini, D.Gutierrez, E.Negri, L.Fumagalli;
- ‘Digital Twin: Generalization, characterization and implementation’ - Eric VanDerHorn, Sankaran Mahadevan, Decision Support Systems (2021);
- ‘Execute Tasks for a Warehouse Robot’ – The MathWorks, Inc. ;
- ‘Pure Pursuit Controller’ - The MathWorks, Inc. ;
- ‘Model and Control a Manipulator Arm with Robotics and Simscape’ - The MathWorks, Inc. ;
- MiR100 User guide – Mobile Industrial Robots;
- MiR100 datasheet – Mobile Industrial Robots;
- UR3/CB3 User Manual – Universal Robots;
- OptiTrack Support Documentation – NaturalPoint Corporation;