Università degli Studi di Padova

Department of Information Engineering

Master Degree in
Control System Engineering

Learning stack of tasks for robotic mobile manipulation

Supervisor:
Prof. Pietro Falco

Co-Supervisor:
Prof. Ruggero Carli

Master Candidate:
Alessandro Adami
2089579

September 3, 2024

Abstract

The aim of this work is to make a robot capable to manage a redundant solution, in order to minimize the cost function associated with a set of tasks. The cost is defined by the user, depending on which aspect is more relevant for the purposes of the final result (e.g. time or precision). Trough reinforcement learning techniques, it will be capable to choose the best solution among the infinite possibilities given by the redundancy of the manipulator. Then the robot will be able to perform different tasks in a three dimensional environment subject to dynamical changes, reacting to different and unpredictable inputs. With the usage of behavior trees and genetic programming techniques, the prioritization of a set of predefined tasks can be learned independently combining them with convex weighted sum and defining the relative parameters. During this learning phase several possible solutions are exploited with operations like crossover and mutation, altering the architecture of the solution. Comparing the result of the cost function, the best resulting algorithm will be chosen. In particular the Baxter robot model was chosen for the problem resolution. The robot is a bi-manual one, with 7 degrees of freedom for each arm, and then a mobile base was integrated in order to let it freely move into the environment. The solution of the tasks, like collision avoidance with obstacles, are separately solved and then combined together with null space projector. The simulations were made using RoboSuite simulation framework and MuJoCo (Multi-Joint dynamics with Contact) engine, through Python programming language.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem description

In the context of robotic systems, those involving manipulators with multiple degrees of freedom, managing redundancy (section 1.3.1) is a critical challenge. Redundancy in a robotic manipulator happens when there are more degrees of freedom than are strictly necessary to perform a given task. While this redundancy offers the potential for greater flexibility and adaptability, it also introduces complexity in decision-making. The robot must select from a possibly infinite set of feasible configurations, each of which can achieve the task, but with varying degrees of efficiency, precision, and other performance metrics. In dynamic and unpredictable environments a robot's ability to adapt its movements is crucial to achieving optimal performance. However, without a robust method for selecting the most appropriate configuration, the benefits of redundancy can be easily lost, leading to sub-optimal outcomes or even failure in task execution. This work focuses on developing a system that enables a robot to manage its redundancy efficiently, minimizing a user-defined cost function associated with a stack of tasks. The cost function is tailored by the user based on the specific priorities of the application, such as minimizing time or maximizing precision. By leveraging reinforcement learning techniques, the robot learns to navigate the space of possible configurations and choose the one that best minimize the cost function.

The challenge of managing goes further simply finding a solution, it involves dynamically adapting to changes in the environment and prioritizing multiple tasks. In this context, the use of genetic programming provides a framework for the robot to learn task prioritization. Through operations like crossover and

mutation, the robot explores various potential solutions, continuously refining its approach to achieve the optimal balance of task execution. In this study, the Baxter robot model serves as the base for implementing and evaluating the proposed methods. With its 7 degrees of freedom per arm and an additional mobile base, Baxter represents a highly redundant system, in order to explore redundancy management. By solving tasks, such as obstacle avoidance or inverse kinematic, independently and then integrating these solutions using a null space projector, this work aims to demonstrate the efficacy of the proposed approach in a simulated 3D environment, using the RoboSuite (section 1.5) framework and MuJoCo (section 1.4) engine.

1.1.1 Objectives

The main objective of this thesis is to advance the state of the art in genetic programming by demonstrating how it is possible to derive an optimized and prioritized order for a set of tasks in a redundant robotic system, guided by a user-defined cost function that reflects the specific performance criteria relevant to the task, such as minimizing execution time. To achieve this, the thesis employs an approach combining reinforcement learning with genetic programming techniques. The reinforcement learning aspect of the work enables the robot to iteratively adjust its parameters and task priorities in response to feedback from the environment, aiming to minimize the cost function. This dynamic adaptation is essential for the robot to effectively manage its redundancy and respond to the complex and variable demands of its operating environment.

The genetic programming techniques, including operations like crossover and mutation, are employed to generate new stacks of tasks. These techniques allow the system to explore a wide variety of potential solutions, creating and evolving population of stacks. By starting with an initial base population, the genetic algorithm evolves this over successive generations, continually refining the task sequences to produce offspring that are increasingly well-suited to the specified cost function.

A critical aspect of this work is the evaluation of the developed task stacks in a simulated environment. This evaluation phase is designed to test the robustness and effectiveness of the solutions in a variety of scenarios, ensuring that the learned task sequences are not only theoretically optimal but also practically viable. By subjecting the robot to different environmental conditions and challenges, the thesis aims to validate the adaptability of the proposed approach.

Beyond these immediate objectives, this research also seeks to contribute to the broader field of robotics by providing insights into how redundancy can be exploited to improve robotic autonomy and efficiency. The methodologies developed here have the potential to be applied across a range of domains, from industrial automation to healthcare, where robots must perform complex tasks in dynamically changing environments.
In summary, the objectives of this thesis are:

- Develop a method for prioritizing tasks in redundant robotic systems using genetic programming.
- Optimize task execution through reinforcement learning, minimizing a user-defined cost function.
- Generate and evolve new task sequences using genetic programming techniques.
- Evaluate the effectiveness of the resulting task sequences in varied simulated environments, even in unpredictable dynamically changing ones.
- Explore the broader implications of redundancy management in advancing robotic autonomy and efficiency.

1.2 Thesis structure

This project is based on two different topics, robotics and learning, which united together give life to the possibility of finding the best possible solution to solve a given task. The two topics are treated in a separate way, in order to emphasize the characteristic aspects of both.
The thesis is divided in the following chapters:

- **Introduction** (chapter 1): a brief introduction about the main topics of the work, redundant robots and genetic programming, and of the used robot and tools.
- **Literature review** (chapter 2): in this chapter the previous works related to this one are introduced from a theoretical point of view.
- **Simulation framework** (chapter 3): an introduction about methodologies adopted from a practical point of view to develop the used algorithm.

- **Robotics** (chapter 4): simulation and results of the robotics features present in this work. The single tasks are showed alone and in combinations trough null space projector.
- **Genetic programming** (chapter 5): all the simulations and the results obtained about genetic programming are presented and showed in this chapter.
- **Test of the learned stack of tasks** (chapter 6): finally, a stack of tasks is tested in two different environments in order to show the goodness of this work.
- **Conclusions** (chapter 7): finally, conclusions and future developments are treated.

1.3 State of the art

In this section the state of the art of the two main topic of this work, namely redundant robots and genetic programming, are reported.

1.3.1 Redundant robots

Redundant robots are characterized by an higher number of degrees of freedom (DOF) with respect to the number which is required in order to solve a task. So, thanks to the surplus of DOF they are allowed to find multiple solutions for the same assignment, tolerating faults or uncertainties, and adapting to changing environments. The additional DOF introduce challenges in control and optimization, but also offer opportunities for improved performance and versatility. Thanks to this characteristics, redundant robot have reached significant attentions in robotics research and industry due to their improved flexibility, dexterity, and robustness. Their control, involve the management of redundancy to optimize performances metrics like manipulability or energy consumption. Progress in sensing technologies, including 3D vision systems, tactile sensors and force or torque sensors, allow redundant robots to sense and interact with the environment more effectively. Learning-based techniques, like reinforcement learning and neural networks, helps adaptive control and trajectory optimization in complex and dynamic environments. Hybrid control strategies combine analytical models with machine learning algorithms to exploit the benefits of both approaches. Learning

algorithms for interpreting sensor data enable autonomous decision making and learning from experience, improving the adaptability and robustness of the robot. Redundant robots find applications in various industries, including manufacturing, healthcare and construction.

1.3.2 Genetic programming

Genetic Programming (GP) is a computational methodology inspired by natural evolution. The aim, is to generate automatically computer programs in order to solve a specific task. Unlike Genetic Algorithms, which evolve with a fixed length strings, GP evolves with variable length programs. In recent years GP has seen significant advancements, driven by improvements in computational power, innovative algorithmic techniques, and the integration of complementary AI methods such as reinforcement learning (RL).

One of the key aspects of GP, is the manipulation and the representation of programs. Traditionally GPs are represent programs as tree-based structures, due to its alignment with the hierarchical nature of many programming languages. However, there exist other representations like linear GP, in which programs are sequences of instructions similar to machine code, and graph-based GP where programs are represented as directed acyclic graphs (DAGs). These representations allow for more efficient execution and the modeling of more complex relationships and program structures.

Fitness evaluation is a crucial component of GP and it has also seen improvements in recent years. Efficient evaluation techniques, such as fitness prediction, surrogate modeling, and fitness approximation help reduce the computational cost associated with evaluating program fitness. Furthermore, multi-objective optimization techniques enable the simultaneous optimization of multiple objectives, such as program accuracy and complexity, leading to more balanced and effective solutions. The search strategies employed in GP have become more sophisticated as well. For example, co-evolution, which involves the simultaneous evolution of multiple interacting populations, promotes diversity and robustness in solutions through competition and cooperation.

Hybrid approaches have also emerged recently, combining GP with other AI techniques to achieve improved results. One notable example is neuroevolution, which integrates neural networks with GP to evolve both network architectures and weights. This approach has proven particularly advantages for tasks such as image recognition and signal processing. Moreover, the integration of GP with RL

techniques has opened new avenues for evolving adaptive policies and strategies in dynamic environments.

The role of RL in this context, is to train agents to make sequences of decisions by rewarding desirable behaviors. The integration of RL in GP has led to significant advancements in the evolution of policies given by RL. GP can be used to evolve policies directly, representing them as programs. Furthermore, GP can evolve policies in an hierarchical way, where high-level strategies invoke lower-level subroutines, aligning well with the hierarchical nature of many tasks. The use of GP to discover and evolve temporally extended actions enhances an agent's ability to learn long-term strategies.

In practical applications, GP has demonstrated its versatility and effectiveness across various domains. For example in robotics, GP is used to evolve control programs that enable robots to exhibit adaptive and resilient behaviors in dynamic environments. In healthcare, GP is employed to create personalized treatment plans based on patient data, while in the field of gaming GP is used to evolve game-playing agents capable of learning and adapting to different environments.

Despite these advancements obtained in GP, several challenges remain. Improving the scalability of GP to handle more complex tasks and larger datasets is an ongoing area of research. Improve the interpretability of evolved programs, making them transparent, is also crucial particularly in domains where understanding the decision-making process is important. Additionally, developing methods for transfer learning, where knowledge gained from one specific domain can be applied to another similar but not exactly equal, can reduce the need for extensive retraining. Lastly, creating interactive systems that facilitate collaboration between humans and GP systems can leverage human intuition and machine precision, leading to more effective problem-solving.

1.4 MuJoCo

MuJoCo [2] (**Mu**lti-**Jo**int dynamics with **Co**ntact) is a physics engine designed for accurate and efficient simulation of rigid body dynamics with contacts. It is widely used to simulate complex environments and physical interactions in fields like robotics, machine learning or biomechanics. MuJoCo provides a simulation of physical systems, modeling the dynamics of rigid and soft bodies and also fluid interactions. The engine is noted in particular for its robust handling of contact dynamics, including friction, which are essential for realistic interaction between bodies. It has optimized performances, making it able of running simulations in real-time or faster, which is crucial for applications in robotics and reinforcement learning, in which the number of simulations is very high.
Users can define models of objects and environments using the XML-based MuJoCo Model (MJCF) format or the URDF format, which is more commonly used in robotics. The engine allows for customization and extension, so that the user can build a simulation tailored on specif needs.

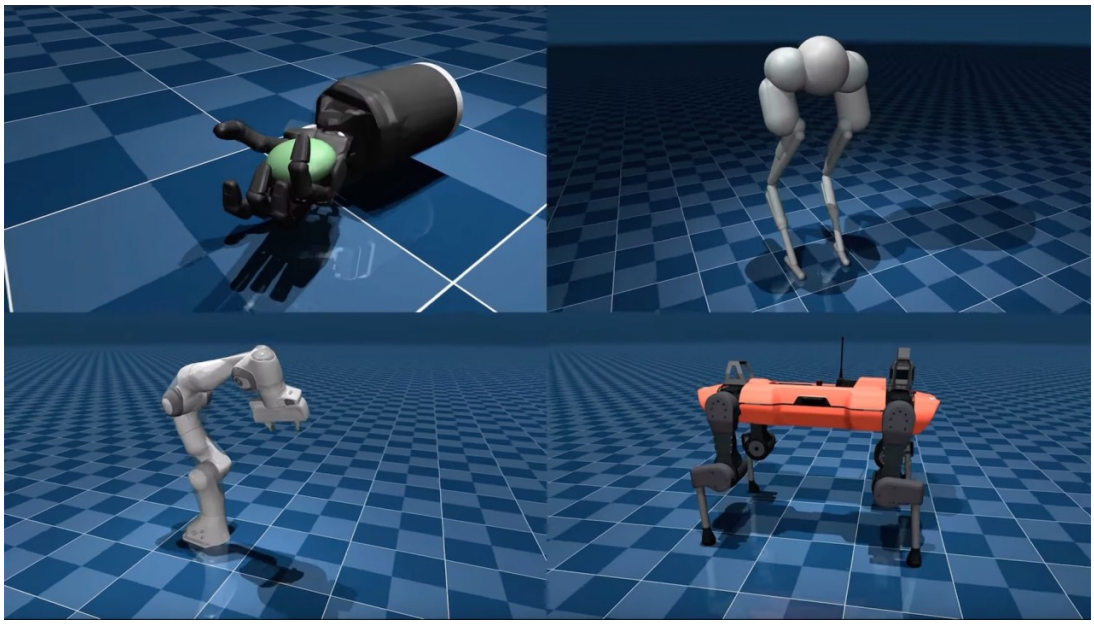

Figure 1.1: Examples of MuJoCo simulations.

MuJoCo provides APIs for several programming languages, including Python which was used for this project, and includes powerful visualization tools to help users debug and analyze their simulations.
It is used in robotics for simulating robotic systems, including manipulation and motion tasks, helping in the development and testing of control algorithms, often

employed in training reinforcement learning agents due to its ability to simulate complex environments and interactions.

1.5 RoboSuite

RoboSuite [1] is a comprehensive software framework designed for simulating robotic manipulation tasks. It builds on the capabilities of MuJoCo (section 1.4), providing the user an environment for developing and testing robotic algorithms. It offers a high-level API, that simplifies the process of setting up and running simulations. The framework is designed with modularity, allowing users to easily swap out components and customize their simulations.
RoboSuite includes a variety of pre-defined robotic manipulation tasks, such as pick and place or open a door. These tasks serve as benchmarks for evaluating robotic algorithms of learning. Users can create custom environments based on their specific research or application needs, exploiting the flexibility of MuJoCo for detailed and accurate simulations. RoboSuite supports the simulation of multiple robots, like Baxter robot, enabling research in collaborative robotics and multi-agent systems. Including support for various sensors, such as cameras and force sensors, it allows for the development of sensor-based control and learning algorithms. Furthermore, RoboSuite provides visualization tools, enabling users to visualize and interact with simulations in real-time. This is essential for debugging and analyzing the behavior of robotic systems using a specific algorithm. In summary, RoboSuite is a powerful tool for researchers and developers in the field of robotics, offering a rich set of features and a flexible interface for simulating robotic manipulation tasks.

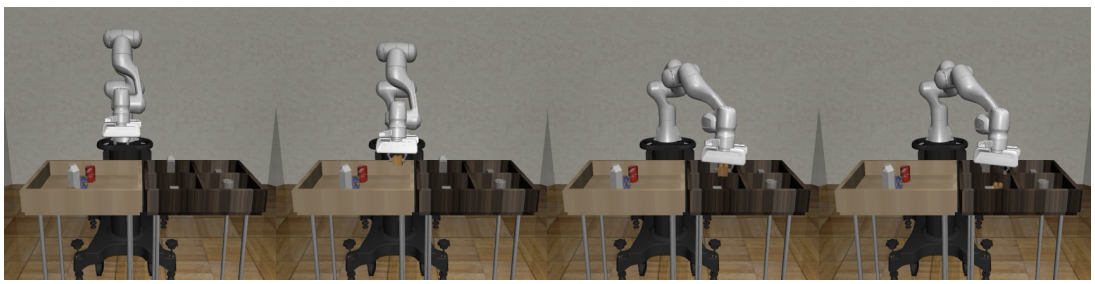

Figure 1.2: Example of RoboSuite single arm pick and place task.

1.6 The Baxter robot

Baxter, a robot developed by Rethink Robotics, is known for its redundancy features, which ensure robustness and safety in various industrial settings. Baxter owns dual-arm *redundancy* (7-DOF, given by 7 revolute joints), equipping it with two identical arms capable of independently performing tasks. This redundancy ensures continuous operation, as if one arm encounters an obstacle, the other can continue performing its task, maximizing the robot productivity.

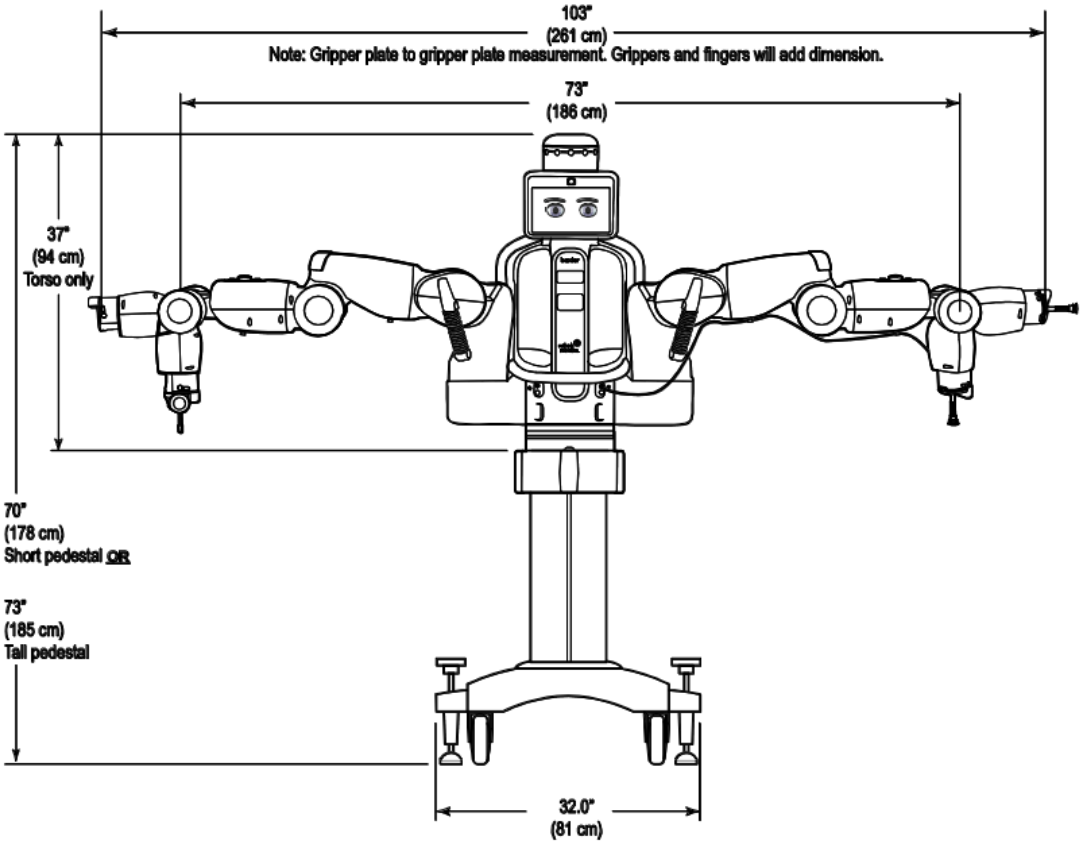

Figure 1.3: Frontal view of the Baxter robot with dimensions.

Overall, Baxter's redundancy features make it a robust and reliable robot suitable for a wide range of industrial applications.
It was built to work safely around humans without the need for safety cages. Its joints and integrated sensors allow it to detect and respond to human presence, making it ideal for collaborative tasks. Baxter has been influential in demonstrating the potential of collaborative robots (cobots) in modern industry. Its design philosophy and capabilities have set a precedent for subsequent developments in the field of robotics, emphasizing the importance of safety, ease of use, and versatility.

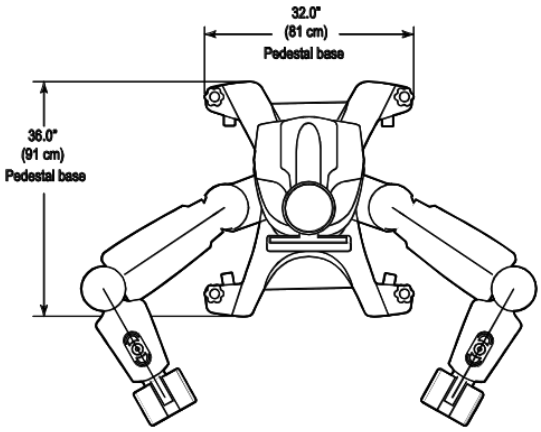

Figure 1.4: Top view of the Baxter robot with pedestal dimensions.

1.6.1 Baxter in RoboSuite

In RoboSuite (section 1.5), Baxter provides a versatile platform for robotic manipulation research, enabling researchers to design, simulate, and evaluate a wide range of robotic tasks and algorithms in a controlled and customizable environment. The model of Baxter in RoboSuite, mirrors the real physical characteristics and capabilities of the real one. This is the only bi-manual robot present by default into the simulation framework and, it serves as a test base to the development of robotic algorithms. Essentially, in RoboSuite, the robot provides a powerful and flexible platform for advancing robot learning research.

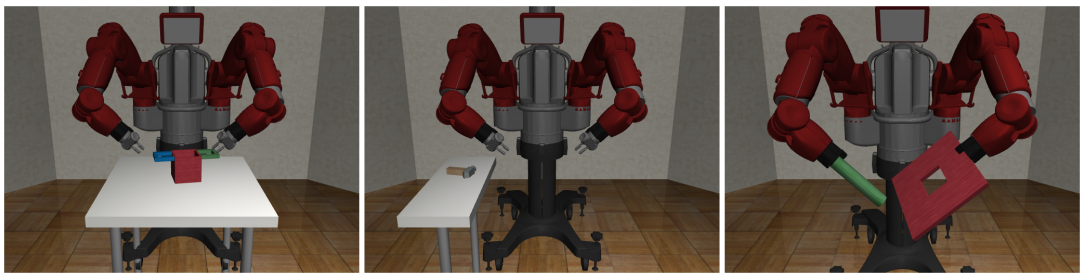

Figure 1.5: Baxter robot simulated into different Robosuite environments.

In general the pedestal does not allow movement, but in the specific case of this work 3-DOF where added simulating a mobile base, bringing the total number to 10-DOF for a kinematic chain, making it high redundant.
The robot is controlled in velocity, meaning that the desired velocity can be applied to each joint as input and the motors are actuated with the proper torque (section 1.6.3).

D-H table

The Denavit–Hartenberg parameters of the Baxter arm are reported in table 1.1:

Link	θ	d	a	α
1	q_1	0.27035	0.069	$-\frac{\pi}{2}$
2	$q_2 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
3	q_3	0.36442	0.069	$-\frac{\pi}{2}$
4	q_4	0	0	$\frac{\pi}{2}$
5	q_5	0.37429	0.01	$-\frac{\pi}{2}$
6	q_6	0	0	$\frac{\pi}{2}$
7	q_7	0.2295	0	$-\frac{\pi}{2}$

Table 1.1: Baxter's arms DH table of parameters. All the joints are revolute, with q_i variable.

For both arms, *left* and *right*, the parameters are the same from the mount of the arm to the end-effector frames. In figure 1.6, all the joints (7 revolute joints) of the two arms are reported with base frame (T_0) and the hand frames at the end of each arm. In frames in figure, the blue axis correspond to the axis z, the red to x and the green to y. Note that all of them are right hand frames. All the joints rotate around their z axis, accordingly to the right hand convention.

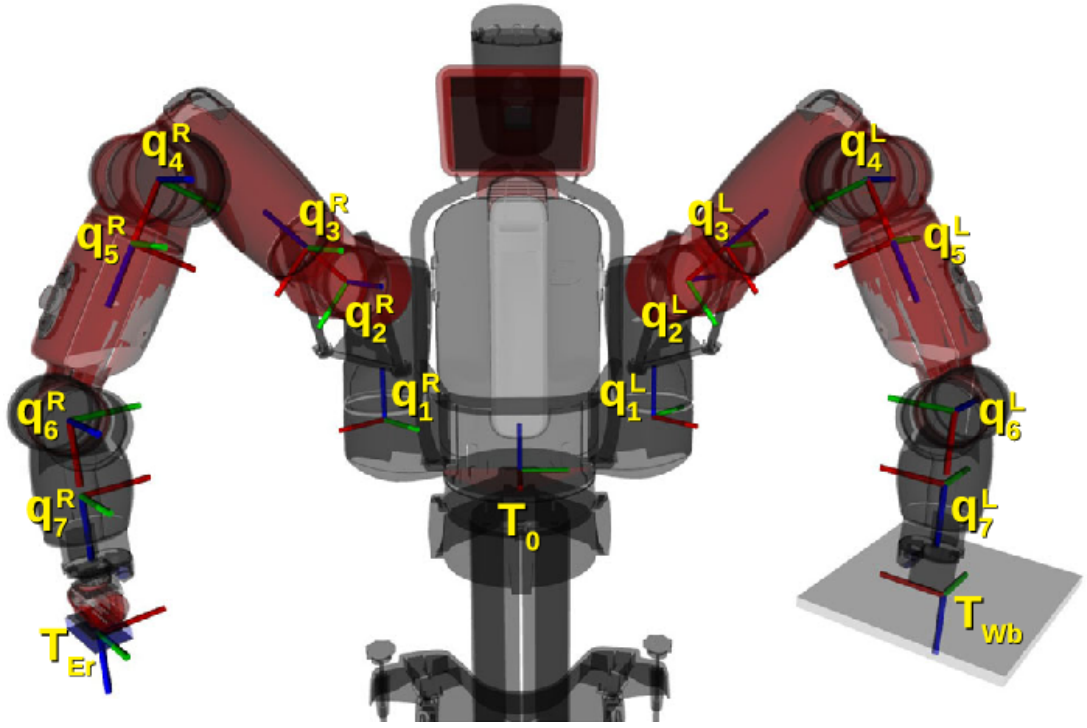

Figure 1.6: Baxter's robot joints with frames.

1.6.2 Baxter arm workspace

Following, the *Baxter workspace* is reported with top and side view of it. All the detailed documents about the robot and its workspace can be find in [7]. Note that the reachable work-space is reported in figure 1.7 and 1.8, and that grippers are not considered.

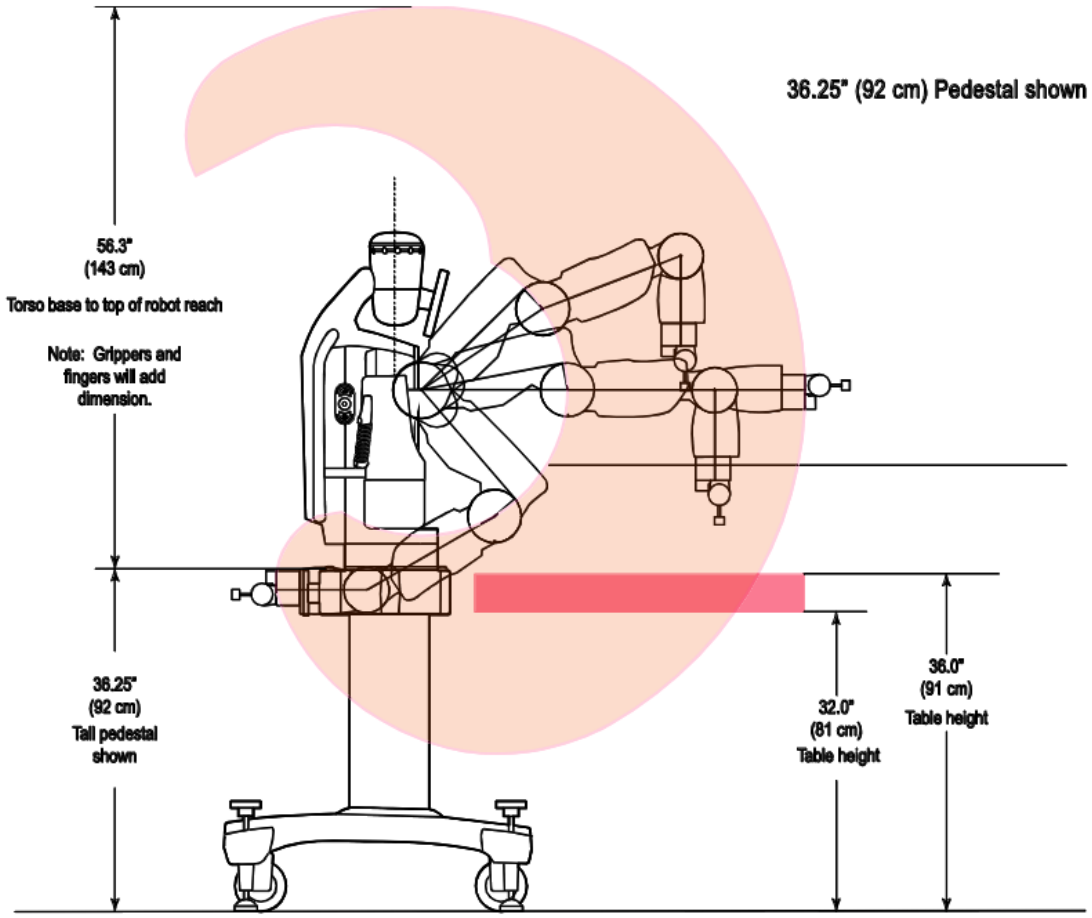

Figure 1.7: Baxter arm work space: side-view.

1.6.3 Baxter joints performances in RoboSuite

The Baxter joints are controlled in velocity, through a proportional control law with the desired value provided by the policy and the current joint velocity of the robot. This control law, parameterized by a proportional constant k_p, generates joint torques to be applied at each simulation step:

$$\tau = k_p(\dot{q}_d - \dot{q}).$$

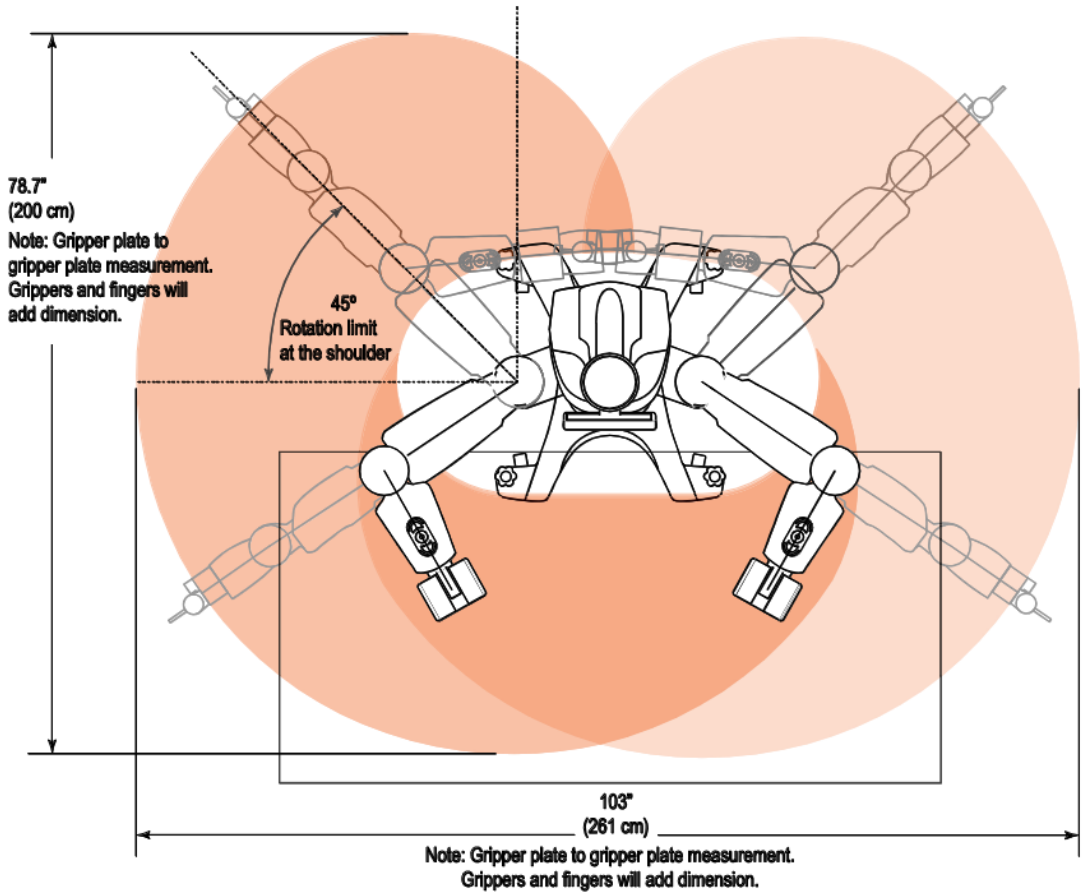

Figure 1.8: Baxter arm work space: top-view.

Each controller of the joints of the arm, has the following parameters in the RoboSuite simulation framework:

Gain k_p	Input range	Output range	Ramp ratio
0.03	[-1, 1] rad/s	[-0.5, 0.5] Nm	0.2 $\frac{Nm}{rad/s}$

Table 1.2: Joint controller parameters.

Following, the *performances* of unitary step response (in radians) in position are reported for each joint. Since the two arms are identical, only performances obtained with the right one are reported in a full dynamics environment.

q_1^R step performances

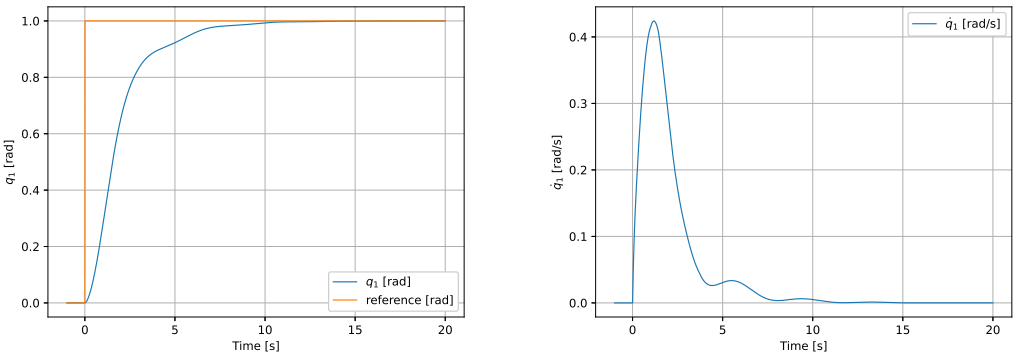

Figure 1.9: q_1^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.189	5.819	0 (0%)	-8.575 e-4 (20s)

Table 1.3: q_1^R joint performances at position step response.

q_2^R step performances

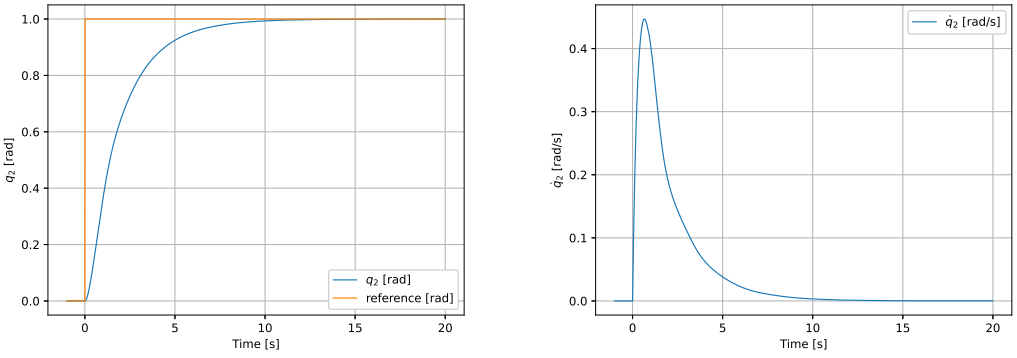

Figure 1.10: q_2^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.442	5.818	0 (1.1 e-2%)	-5.166 e-4 (20s)

Table 1.4: q_2^R joint performances at position step response.

q_3^R step performances

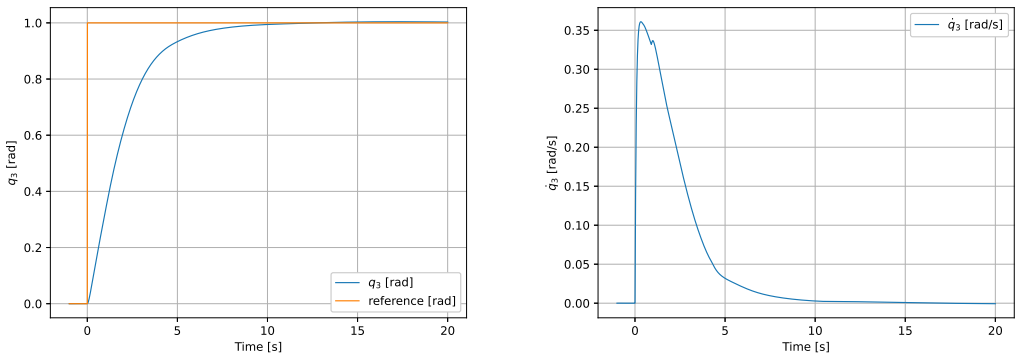

Figure 1.11: q_3^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.196	5.596	0.004 (4.0 e-1%)	3.665 e-3 (20s)

Table 1.5: q_3^R joint performances at position step response.

q_4^R step performances

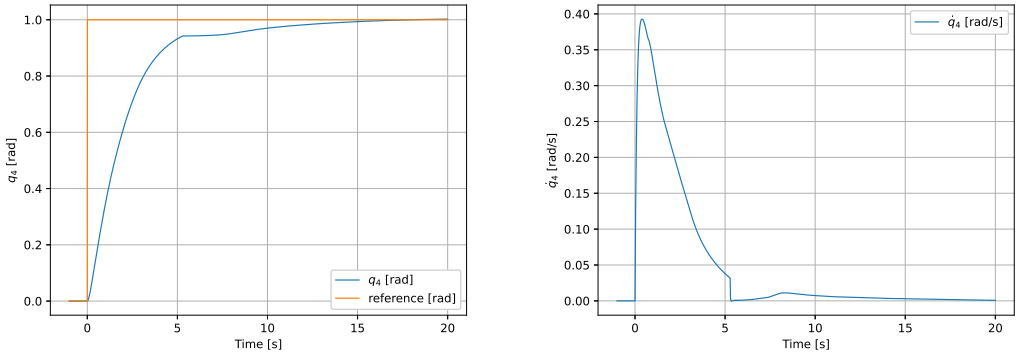

Figure 1.12: q_4^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.314	5.55	0.003 (2.8 e-1%)	7.873 e-4 (20s)

Table 1.6: q_4^R joint performances at position step response.

q_5^R step performances

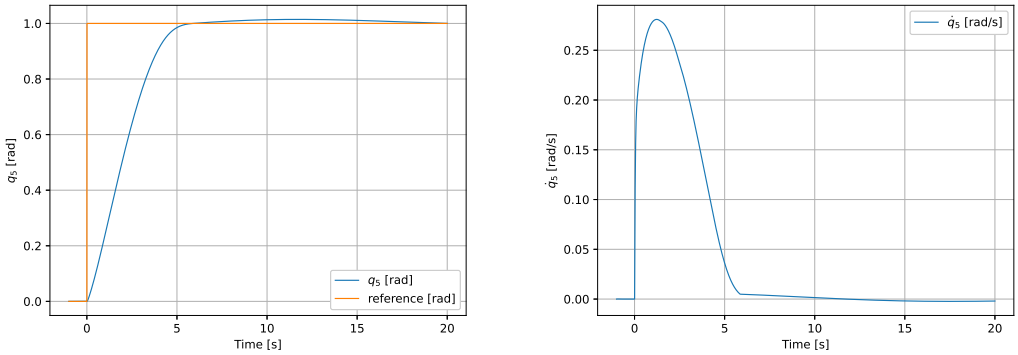

Figure 1.13: q_5^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
3.912	4.39	0.0145 (1.45%)	1.427 e-2 (20s)

Table 1.7: q_5^R joint performances at position step response.

q_6^R step performances

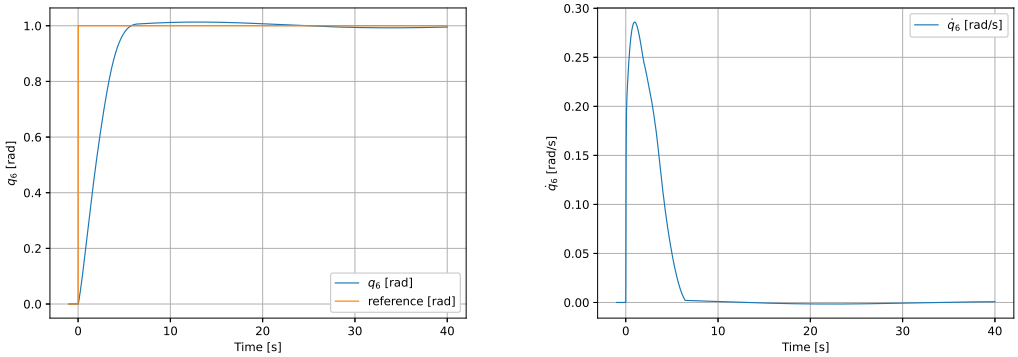

Figure 1.14: q_6^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.029	4.557	0.018 (1.8%)	8.111 e-4 (40s)

Table 1.8: q_6^R joint performances at position step response.

q_7^R step performances

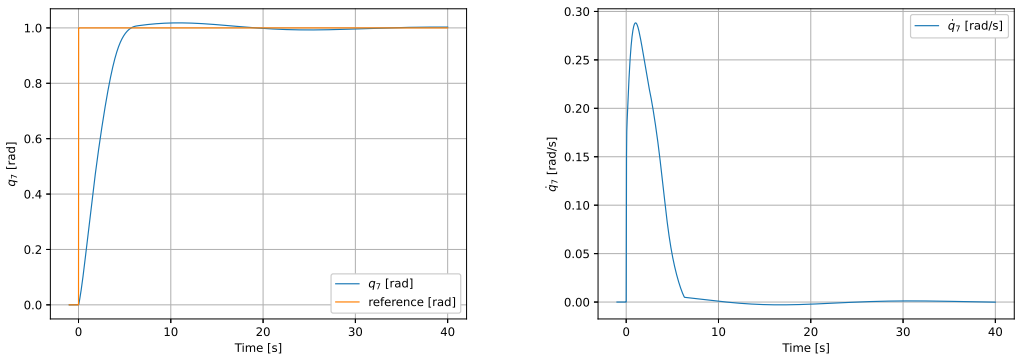

Figure 1.15: q_7^R joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot [rad]	Steady state error [rad]
4.006	5.798	0.037 (3.7%)	1.108 e-4 (40s)

Table 1.9: q_7^R joint performances at position step response.

As it is possible to observe from the performances, the joint response of the Baxter robot in a RoboSuite simulation are not particularly high performing in terms of speed. However, for the aim of this work (section 1.1.1) this does not entail any consequences or complications. On the other hand, the joints are not affected by significant overshoot. This this will help ensure accuracy in task execution.

Chapter 2

Literature review

2.1 Robotics

2.1.1 Fundamentals of Robotics Kinematics

For a detailed view of all the elements introduced in this section and not, refer to Siciliano et. al in book [3].

Cartesian Space

Cartesian coordinates allow to specify the location and orientation of a point in a 3-dimensional space. Position and orientation are both expressed as a triplet of numbers, which express the distance from coordinate axis and orientation with respect to them. The Cartesian coordinate system is based on three mutually perpendicular and oriented coordinate axes: the x-axis, the y-axis, and the z-axis. The intersection point of the three axes is called origin. This space is intuitive for humans to understand and specify robot positions and movements in terms of Cartesian coordinates, as they reflect how we perceive three-dimensional space.
The combination of position and orientation of a point is called pose, and it is represented as:

$$\boldsymbol{p} = [x, y, z, \alpha_x, \alpha_y, \alpha_z] \in \mathbb{R}^6. \tag{2.1}$$

In this work, all the axes are always oriented following the right hand rule, the same for the direction of rotations.
The pose of the end-effector is typically expressed in Cartesian coordinates and it will be expressed in this work as:

$$\boldsymbol{p} = [p_x, p_y, p_z, \theta_x, \theta_y, \theta_z] \in \mathbb{R}^6, \tag{2.2}$$

while the end-effector velocities (linear and angular) will be represented as:

$$\dot{\boldsymbol{p}} = [\dot{p}_x, \dot{p}_y, \dot{p}_z, \dot{\theta}_x, \dot{\theta}_y, \dot{\theta}_z] \in \mathbb{R}^6. \tag{2.3}$$

Joint Space

The coordinates of each link of a robot are represented in *Joint Space* coordinates (or *Configuration Space*). The space is n-dimensional, where n is the number of independent joints of a kinematic chain, which correspond to the number of degrees of freedom (DOF) of the mechanical structure. Controlling the robot's movements in the joint space is often more direct and simplified, since it is possible to act directly on the motors or actuators of each joint.
All the joint variables are represented with the letter q, so that the space is represented with a vector:

$$\boldsymbol{q} = [q_1, ..., q_n] \in \mathbb{R}^n, \tag{2.4}$$

while the space of joint velocities is represented as:

$$\dot{\boldsymbol{q}} = [\dot{q_1}, ..., \dot{q_n}] \in \mathbb{R}^n. \tag{2.5}$$

The joints are numbered in ascending order from base to the end-effector.

Forward Kinematics

Forward Kinematics is the process that maps the position and the orientation of the end-effector in Cartesian space, from the position of the joints into the operational space:

$$f\big([q_1, q_2, ..., q_n]\big) = [p_x, p_y, p_z, \theta_x, \theta_y, \theta_z], \tag{2.6}$$

where f is a map $\mathbb{R}^n \longrightarrow \mathbb{R}^6$.
Considering a n-DOF manipulator, the direct kinematic can be expressed as

$$\boldsymbol{T}_e(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{R}_e(\boldsymbol{q}) & \boldsymbol{p}_e(\boldsymbol{q}) \\ \mathbf{o}^T & 1 \end{bmatrix}, \tag{2.7}$$

where $\boldsymbol{q} = [q_1, q_2, ..., q_n]$ is the vector of joint variables, $\boldsymbol{R}_e(\boldsymbol{q})$ the rotation matrix associated to the end-effector and $\boldsymbol{p}_e(\boldsymbol{q})$ its position. The pose of the end-effector varies as $\boldsymbol{q}$.

Kinematic chain

A manipulator can be schematically represented from a mechanical viewpoint as a *kinematic chain* of links connected by means of joints, revolute or prismatic. One end of the chain is constrained to a base, while to the other end an end-effector is mounted. The resulting motion of the structure is obtained by composition of the elementary motions of each link with respect to the previous one.

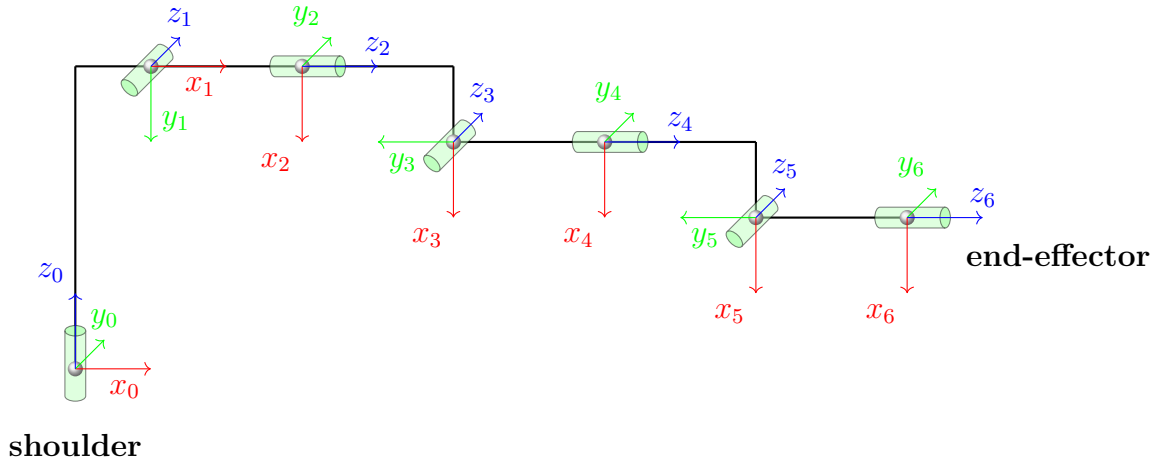

Figure 2.1: Baxter's arm kinematic chain.

Jacobian Matrix

The *Jacobian* constitutes one of the most important tools for manipulator characterization, to analyze and control robot's movement. In the field of robotics, the Jacobian matrix is used to relate the velocities of the robot's joints to the velocity of the end-effector.

Geometric Jacobian The *Geometric Jacobian* is s a matrix that represent the relation between joints velocities in operational space and end-effector velocities in Cartesian space, and depends on robot configuration.
Considering the direct kinematic equation (2.6) for a n-DOF manipulator, it is desired to express the end-effector linear velocity $\dot{\boldsymbol{p}}_e$ and angular velocity $\boldsymbol{\omega}_e$ e as a function of the joint velocities $\dot{\boldsymbol{q}}$. The relations are both linear in joint velocities:

$$\dot{\boldsymbol{p}}_e = \boldsymbol{J}_P(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.8}$$

$$\boldsymbol{\omega}_e = \boldsymbol{J}_O(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.9}$$

where $\boldsymbol{J}_P$ and $\boldsymbol{J}_O$ are both $3 \times n$ dimensional matrices. The first one relates the joint velocities $\boldsymbol{q}$ to the end-effector linear velocities $\dot{\boldsymbol{p}}_e$, while the second one is related to end-effector angular velocities $\boldsymbol{\omega}_e$. (2.8) and (2.9) can be rewritten in a compact form as:

$$\boldsymbol{v}_e = \begin{bmatrix} \dot{\boldsymbol{p}}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_P(\boldsymbol{q}) \\ \boldsymbol{J}_O(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{2.10}$$

which represents the manipulator differential kinematics equation. The $6 \times n$ matrix $\boldsymbol{J}(\boldsymbol{q})$ is the geometric Jacobian matrix of the manipulator, in function of the joint variables.

Analytical Jacobian The *Analytical Jacobian*, is the resulting matrix via differentiation of the direct kinematics function with respect to the joint variables. This is possible if the end-effector pose is expressed with reference to a minimal representation in the operational space.
The linear velocity of the end-effector can be expressed as the derivative of the position vector $\boldsymbol{p_e}$, representing the e-e frame with respect to the base frame, in function of $\boldsymbol{q}$:

$$\dot{\boldsymbol{p}}_e = \frac{\partial \boldsymbol{p_e}}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} = \boldsymbol{J}_P(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.11}$$

In the case of rotational velocity of the end-effector frame, the minimal representation of orientation in terms of three variables $\boldsymbol{\phi}_e(\boldsymbol{q})$ can be considered. Its time derivative $\dot{\boldsymbol{\phi}}_e$, in general, differs from the angular velocity vector defined for geometric Jacobian. Then, the obtained Jacobian is:

$$\dot{\boldsymbol{\phi}}_e = \frac{\partial \boldsymbol{\phi}_e}{\partial \boldsymbol{q}} \dot{q} = \boldsymbol{J}_\phi(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.12}$$

Finally, the direct kinematic equation is:

$$\dot{\boldsymbol{x}}_e = \begin{bmatrix} \dot{\boldsymbol{p}}_e \\ \dot{\boldsymbol{\phi}}_e \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_P(\boldsymbol{q}) \\ \boldsymbol{J}_\phi(\boldsymbol{q}) \end{bmatrix} \dot{\boldsymbol{q}} = \boldsymbol{J}_A(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{2.13}$$

where in general $\boldsymbol{J}_A$ is different from the geometric Jacobian $\boldsymbol{J}$.

Relation between Jacobians The two derived Jacobians are in general different, since the angular velocity of the frames $\boldsymbol{\omega}_e$ is not given by $\dot{\boldsymbol{\phi}}_e$. It is possible to find the relationship between angular velocity $\boldsymbol{\omega}_e$ e and rotational velocity $\dot{\boldsymbol{\phi}}_e$ for

a given set of orientation angles

$$\boldsymbol{\omega}_e = \boldsymbol{T}(\boldsymbol{\phi}_e)\dot{\boldsymbol{\phi}}_e, \tag{2.14}$$

where, in the case of statics XYZ Euler angles

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & -sin(\phi_{e_2}) \\ 0 & cos(\phi_{e_1}) & cos(\phi_{e_2})sin(\phi_{e_1}) \\ 0 & -sin(\phi_{e_1}) & cos(\phi_{e_2})cos(\phi_{e_1}) \end{bmatrix}.$$

Once the relation between $\boldsymbol{\omega}_e$ and $\dot{\boldsymbol{\phi}}_e$ is obtained, it is possible to relate the analytical and geometrical Jacobians as:

$$\boldsymbol{J} = \boldsymbol{T}_A(\boldsymbol{\phi})\boldsymbol{J}_A, \tag{2.15}$$

where

$$\boldsymbol{v}_e = \begin{bmatrix} \boldsymbol{I} & \mathbf{o} \\ \mathbf{o} & \boldsymbol{T}(\boldsymbol{\phi}_e) \end{bmatrix} \dot{\boldsymbol{x}}_e = \boldsymbol{T}_A(\boldsymbol{\phi})\dot{\boldsymbol{x}}_e \tag{2.16}$$

Singularities

The Jacobian typically define a linear mapping between velocities in Cartesian and configuration space

$$\boldsymbol{v}_e = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.17}$$

Jacobian which, in general, is in function of the configuration $\boldsymbol{q}$. Those configurations in which $\boldsymbol{J}$ is rank-deficient, namely it is not full rank, are called *singularities*. Singularities represent configurations at which mobility of the structure is reduced, so it is not possible to impose an arbitrary motion to the end-effector. In this point, infinite solutions to the inverse kinematics problem may exist and, in the neighbourhood of this kind of configuration very small velocities in the operational space may cause large velocities in the joint space, since the determinant of $\boldsymbol{J}$ is close to zero.

Redundant Manipulators

Redundant robots (section 1.3.1) have more degrees of freedom than what is needed in order to complete a task and, the Jacobian matrix has more columns than rows. So that, infinite solution to (2.17) exist. A typical solution in this case is to formulate the problem as a constrained linear optimization problem.

The detailed computation derived by Siciliano et al. [3], can be found in section 3.5.1 of the cited book.
The final obtained result, for desired velocities in configuration space is

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}\boldsymbol{v}_e + (\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}_0, \tag{2.18}$$

where $\boldsymbol{J}^{\dagger}$ represents the Moore–Penrose pseudo-inverse of the Jacobian matrix and $\dot{\boldsymbol{q}}_0$ a vector of arbitrary joint velocities. The matrix $(\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}J)$ is one of those matrices, that allow the projection into the *null space* (section 2.1.6) of $\boldsymbol{J}$ the vector $\dot{\boldsymbol{q}}_0$. This means that in the case in which $\boldsymbol{v}_e = 0$, it is possible to generate internal motions described by $(\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J})\dot{\boldsymbol{q}}_0$, that reconfigure the manipulator structure without changing the end-effector position and orientation.

Denavit–Hartenberg Convention

The *Denavit-Hartenberg Convention*, fixes a standard way to define the relative position and orientation of two consecutive links. In general, frames can be arbitrarily attached to links but, it is convenient to set some rules also for the definition of the link frames. More details can be founded in section 2.8.3 of [3]. In this work, this convention was followed, assign frames with a right hand rule. All the parameters used in order to pass from the base frame to the frame attached to the end-effector, summarized into the DH-table, as did in table 1.1 for the Baxter robot.

2.1.2 Inverse Kinematics

The *inverse kinematic* problem, consist in determining the joint variable values corresponding to a given end-effector configuration in the Cartesian space in position and orientation. This is essential in order to transform end-effector motion specification into the corresponding joint space motion specification, to execute the desired task. The solution is, in general, non linear and it is not always possible to find a close form solution for the equation. Many solutions may exist, potentially infinite, and in some cases might be no admissible solutions. The existence of solutions is guaranteed only if the given end-effector position and orientation belong to the manipulator dexterous workspace (section 2.10.1 [3]).
When, in most of the cases, the computation of a closed form solution is not possible, it is the case to use numerical solution techniques. These have the advantage of being applicable to any kinematic structure.

Inverse Differential Kinematics

Differential kinematics equation represents a linear mapping between the joint velocity space and the operational velocity space. This fact suggests the possibility to utilize the differential kinematics equation to tackle the inverse kinematics problem. Supposing that a given trajectory is assigned at the end-effector (v_e), the scope is to find a feasible trajectory in terms of $\boldsymbol{q}(t)$ and $\dot{\boldsymbol{q}}(t)$, respectively joints position and velocity with respect to time.
Via simple inversion, it can be obtained

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\boldsymbol{v}_e. \tag{2.19}$$

This technique is independent of the solvability of the kinematic structure. Nonetheless, it is necessary that the Jacobian be square and of full rank. In the case of redundant robot and singularities, the solution is exploited in sections *Redundant Manipulators* of section 2.1.1 and *Singularities* of section 2.1.1.
The task can be executed actuating the corresponding velocity in the configuration space,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\Big(\dot{\boldsymbol{p}}_d + \gamma\big(\boldsymbol{p}_d - \boldsymbol{p}(\boldsymbol{q})\big)\Big), \tag{2.20}$$

where $\boldsymbol{p}$ indicates the pose of the end-effector.

Inverse Kinematic trajectory

Instead of simply solving the problem (2.19) to reach the desired pose of the end-effector, it i possible to assign a *trajectory* to it. All the details of the computation can be find in section 4.1 of [3].
Given the initial position of the end-effector $\boldsymbol{p}_i$ and the target position $\boldsymbol{p}_f$, it is possible to define a trajectory in term of position and velocity

$$\boldsymbol{p}_d = \boldsymbol{p}_i + s_t\left(\frac{\boldsymbol{p}_f - \boldsymbol{p}_i}{||\boldsymbol{p}_f - \boldsymbol{p}_i||}\right) \tag{2.21}$$

$$\dot{\boldsymbol{p}}_d = (3a_3t^2 + 2a_2t)\left(\frac{\boldsymbol{p}_f - \boldsymbol{p}_i}{||\boldsymbol{p}_f - \boldsymbol{p}_i||}\right) \tag{2.22}$$

where t is the actual time from the initial instant of time, $s_t = a_3t^3 + a_2t^2$, $a_3 = \frac{-2}{3}\frac{a_2}{t_f}$, $a_2 = \frac{3s_f}{t_f^2}$ where $s_f = ||\boldsymbol{p}_f - \boldsymbol{p}_i||$ and t_f is the desired time (in seconds) in which the trajectory should be completed.
Similarly, it's possible to derive a trajectory in Euler Angles representation of the

orientation of the end-effector. However, an *Angle and Axis* representation was preferred. Given two coordinate frames in the Cartesian space with the same origin and different orientation, it is always possible to determine a unit vector so that the second frame can be obtained from the first frame by a rotation of a proper angle about the axis of such unit vector. Being $\boldsymbol{R}_i$ and $\boldsymbol{R}_f$ the rotation matrices of the initial and final configuration, the rotation matrix between the two frames is described as

$$\boldsymbol{R}_f^i = \boldsymbol{R}_i^T \boldsymbol{R}_f = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \tag{2.23}$$

If the matrix $\boldsymbol{R}_i(t)$ is defined to describe the transition from $\boldsymbol{R}_i$ to $\boldsymbol{R}_f$, $\boldsymbol{R}^i(0) = \boldsymbol{I}$ and $\boldsymbol{R}^i(t_f) = \boldsymbol{R}_f^i$. Hence, the matrix $\boldsymbol{R}_f^i$ can be expressed as the rotation matrix about a fixed axis in space; the unit vector r_i of the axis and the angle of rotation θ_f can be computed by using

$$\theta_f = cos^{-1}\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right) \tag{2.24}$$

$$r = \frac{1}{2sin\theta_f} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \tag{2.25}$$

if $sin\theta_f \neq 0$. $\boldsymbol{R}_i(t)$ can also be expressed in function of $\theta(t)$, imposing a timing law from θ_i to θ_f. Then, since r_i is constant, the resulting velocity and acceleration can be expressed as

$$\omega_i = \dot{\theta} r_i \tag{2.26}$$

$$\dot{\omega}_i = \ddot{\theta} r_i. \tag{2.27}$$

Finally in order to characterize the end-effector orientation trajectory with respect to the base frame, the following transformations are needed

$$\boldsymbol{R}_e(t) = \boldsymbol{R}_i \boldsymbol{R}^(t) \tag{2.28}$$

$$\omega_e(t) = \boldsymbol{R}_i \omega^i(t) \tag{2.29}$$

$$\dot{\omega}_e(t) = \boldsymbol{R}_i \dot{\omega}^i(t). \tag{2.30}$$

After that, he desired trajectory can be imposed trough $\boldsymbol{R}_e(t)$ and $\omega_e(t)$. In particular $\boldsymbol{R}_e(t)$ is transformed in the corresponding quaternion $\boldsymbol{q}_e = \{\eta_e,\ \boldsymbol{\epsilon}_e\}$ and the $\boldsymbol{R}_f$ quaternion $\boldsymbol{q}_f = \{\eta_f,\ \boldsymbol{\epsilon}_f\}$. The error can be derived as

$$\boldsymbol{e}_o = \eta_e(\boldsymbol{q})\boldsymbol{\epsilon_f} - \eta_f\boldsymbol{\epsilon}_e(\boldsymbol{q}) - \boldsymbol{S}(\boldsymbol{\epsilon}_f)\boldsymbol{\epsilon}_e(\boldsymbol{q}), \tag{2.31}$$

where the skew-symmetric operator $\boldsymbol{S}(\cdot)$ has been used. At this point the equation 2.20 can be computed.

Inverse kinematic error

The *error in inverse kinematic* at time t $\boldsymbol{e}_{pose}\big(\boldsymbol{q}(t)\big)$ is represented as the difference element by element between the desired pose $\boldsymbol{p}_d$ and the actual one at time t $\boldsymbol{p}\big(\boldsymbol{q}(t)\big)$:

$$\boldsymbol{e}_{pose}\big(\boldsymbol{q}(t)\big) = \boldsymbol{p}_d - \boldsymbol{p}\big(\boldsymbol{q}(t)\big), \tag{2.32}$$

and it describes how far is the robot from reaching the desire pose with the end-effector. The error can also be divided in position and orientation error:

$$\boldsymbol{e}_{position}\big(\boldsymbol{q}(t)\big) = \begin{bmatrix} p_{x_d} \\ p_{y_d} \\ p_{z_d} \end{bmatrix} - \begin{bmatrix} p_x\big(\boldsymbol{q}(t)\big) \\ p_y\big(\boldsymbol{q}(t)\big) \\ p_z\big(\boldsymbol{q}(t)\big) \end{bmatrix}, \tag{2.33}$$

$$\boldsymbol{e}_{orientation}\big(\boldsymbol{q}(t)\big) = \begin{bmatrix} \theta_{x_d} \\ \theta_{y_d} \\ \theta_{z_d} \end{bmatrix} - \begin{bmatrix} \theta_x\big(\boldsymbol{q}(t)\big) \\ \theta_y\big(\boldsymbol{q}(t)\big) \\ \theta_z\big(\boldsymbol{q}(t)\big) \end{bmatrix}. \tag{2.34}$$

2.1.3 Obstacle Avoidance

The robot is considered as protected by N fictitious springs, with rest length r_k. Each one is considered as attached to distance sensor. When an *obstacle* is sensed and the distance is smaller than r_k, the total elastic energy associated to the springs increase, as if they are touched and compressed by the obstacle. The objective of the robot, for this task, is to minimize the total energy, correcting its configuration.
The total pseudo-energy of the k^{th} spring is defined as:

$$\epsilon_k(\boldsymbol{q}) = \begin{cases} \frac{1}{2}(d_k - r_k)^2, & \text{if } d_k \leq r_k \\ 0, & \text{otherwise} \end{cases} \tag{2.35}$$

where $d_k(\boldsymbol{q}) = ||\boldsymbol{p}_{o_k} - \boldsymbol{p}_{s_k}(\boldsymbol{q})||$ is the information provided by the k^{th} distance sensor. $\boldsymbol{p}_{o_k}$ is the position vector of the obstacle point at minimum distance from the k^{th} sensor and $\boldsymbol{p}_{s_k}(\boldsymbol{q})$ is the position vector of the k^{th} sensor from the base of the kinematic chain. Note that the value of the pseudo-energy depends on the distance of the obstacle and the robot configuration.
The total energy of all the springs is given by

$$\sigma(\boldsymbol{q}) = \sum_{k=1}^{N} \epsilon_k(\boldsymbol{q}), \tag{2.36}$$

while, the velocity in the configuration space is given by

$$\dot{\boldsymbol{q}}_o = \boldsymbol{J}_o^{\dagger}(\boldsymbol{q})(\dot{\boldsymbol{\sigma}}_d + \gamma_o(\boldsymbol{\sigma}_d - \boldsymbol{\sigma}(\boldsymbol{q}))) \tag{2.37}$$

where $\boldsymbol{J}_o(\boldsymbol{q}) = \frac{\partial \boldsymbol{\sigma}(\boldsymbol{q})}{\partial \boldsymbol{q}}$, γ_o is the gain which influence the close loop kinematics, $\boldsymbol{\sigma}_d$ is the desired pseudo energy and $\dot{\boldsymbol{\sigma}}_d$ its derivative. The definition of energy in (2.35) and (2.36) allows the computation of the gradient in closed for as:

$$\boldsymbol{J}_o(\boldsymbol{q}) = \sum_{k=1}^{N} \frac{\partial \epsilon_k(\boldsymbol{q})}{\partial \boldsymbol{q}} \tag{2.38}$$

where

$$\frac{\partial \epsilon_k(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{cases} -(d_k - r_k) v_{d_k}^T \frac{\partial \boldsymbol{p}_{s_k}(\boldsymbol{q})}{\partial \boldsymbol{q}}, & \text{if } d_k \leq r_k \\ 0, & \text{otherwise} \end{cases} \tag{2.39}$$

and $\boldsymbol{v}_{d_k}(\boldsymbol{q}) = \frac{\boldsymbol{p}_{o_k} - \boldsymbol{p}_{s_k}(\boldsymbol{q})}{||\boldsymbol{p}_{o_k} - \boldsymbol{p}_{s_k}(\boldsymbol{q})||}$ direction of the sensed obstacle.
In order to compute the matrix $\boldsymbol{J}_o(\boldsymbol{q})$ all the derivatives $\frac{\partial \boldsymbol{p}_{s_k}(\boldsymbol{q})}{\partial \boldsymbol{q}}$ are analytically computed.

Obstacle Avoidance trajectory

In order to obtain a *trajectory* for the robot, while avoiding an obstacle, $\boldsymbol{\sigma}_d(t, \boldsymbol{\sigma}_i)$ and $\dot{\boldsymbol{\sigma}}_o(t, \boldsymbol{\sigma}_i)$ are dependent from time and initial pseudo-energy, for the function (2.37). The initial pseudo-energy ϵ_{i_k}, is sampled at the first instant of time, in which the k^{th} sensor is activated by an obstacle. Then

$$\boldsymbol{\sigma}_d = \boldsymbol{\sigma}_i + s_t \left(\frac{\boldsymbol{\sigma}_{d_0} - \boldsymbol{\sigma}_i}{||\boldsymbol{\sigma}_{d_0} - \boldsymbol{\sigma}_i||} \right) \tag{2.40}$$

$$\dot{\boldsymbol{\sigma}}_o = (3a_3t^2 + 2a_2t)\left(\frac{\boldsymbol{\sigma}_{d_0} - \boldsymbol{\sigma}_i}{||\boldsymbol{\sigma}_{d_0} - \boldsymbol{\sigma}_i||}\right) \tag{2.41}$$

where t is the actual time of the simulation from the moment in which the sensor is activated, $s_t = a_3t^3 + a_2t^2$, $a_3 = \frac{-2}{3}\frac{a_2}{t_f}$, $a_2 = \frac{3s_f}{t_f^2}$ where $s_f = ||\boldsymbol{\sigma}_{d_0} - \boldsymbol{\sigma}_i||$ and t_f is the desired time (in seconds) in which the trajectory should end. $\boldsymbol{\sigma}_{d_0}$ is the desired final pseudo-energy, which typically is equal to zero.

2.1.4 Manipulability Ellipsoids and Measure

Manipulability represents the attitude of a manipulator to arbitrarily change end-effector position and orientation, and in particular it is describe by the velocity manipulability ellipsoid. Considering the set of joint velocities of constant unit norm $\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} = 1$, this equation describes the points on a sphere surface in the joint velocity space. Using equation 2.19 it is possible to obtain

$$\boldsymbol{v}_e^T\big(\boldsymbol{J}^{\dagger T}(\boldsymbol{q})\boldsymbol{J}^{\dagger}(\boldsymbol{q})\big)\boldsymbol{v}_e = \boldsymbol{v}_e^T\big(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q})\big)\boldsymbol{v}_e = 1, \tag{2.42}$$

which is the equation of the points on the surface of an ellipsoid in the end-effector velocity space. Along the direction of the major axis of the ellipsoid, the end-effector can move at large velocity, while along the direction of the minor axis small velocities are obtained for it. The principal axes of the ellipsoid is determined by the eigenvectors $\boldsymbol{u}_i$ for the matrix $\boldsymbol{J}\boldsymbol{J}^T$, while the dimensions of the axes are given by $\sigma_i = \sqrt{\lambda_i \boldsymbol{J}\boldsymbol{J}^T}$. A global representative measure of manipulation ability can be obtained by computing the volume of the ellipsoid proportional to

$$w(\boldsymbol{q}) = \sqrt{det(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q}))}. \tag{2.43}$$

Maximization of the manipulability measure

Defined as equation 2.43 the manipulability measure vanish in a singular configuration. So, maximizing this measure, means to move away from singularities. A typical choice for the vector $\dot{q}_0$ in equation (2.18) is:

$$\dot{\boldsymbol{q}} = k_0\left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T \tag{2.44}$$

where $k > 0$. The solution moves along the direction of the gradient of the objective function $w(\boldsymbol{q})$.

Derivative

In order to maximize the measure, it is fundamental to obtain the derivative of the measure $w(\boldsymbol{q})$. The derivative can be find in analytically as

$$\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} = w \cdot tr\left((\boldsymbol{J}\boldsymbol{J}^T)^{-1} \frac{\partial \boldsymbol{J}}{\partial q_i} \boldsymbol{J}^T \right) \tag{2.45}$$

for each of the i joints. The gradient of the Jacobian, is founded analytically.

2.1.5 Distance from mechanical joint limits

The *distance from mechanical joint limits* is defined as

$$w(\boldsymbol{q}) = -\frac{1}{2n} \sum_{i=1}^{n} \left(\frac{q_i - \bar{q}_i}{q_{i_{max}} - q_{i_{min}}} \right)^2 \tag{2.46}$$

where $q_{i_{max}}$ and $q_{i_{min}}$ denotes the maximum and the minimum values that the i^{th} joint variable can assume, while $\bar{q}_i$ is the mean value of the joint range. This helps to keep as close as possible the to the center of each range the joint variables, avoiding limits of the joints range and mechanical issues.
The maximization of the measure is made in the same way of (2.44).

Derivative

In order to maximize the measure, it is fundamental to obtain the derivative of the measure $w(\boldsymbol{q})$. The derivative can be find in analytically as

$$\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} = -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{q_i - \bar{q}_i}{q_{i_{max}} - q_{i_{min}}} \right) \tag{2.47}$$

for each of the i joints.

2.1.6 Null Space Projection

The *null space* of a Jacobian matrix $\boldsymbol{J}$ is the subspace $N(\boldsymbol{J})$ in $\mathbb{R}^{n-r}$ of joint velocities, that do not produce any effect on the end-effector velocities. On the contrary the *range space* $R(\boldsymbol{J})$ in $\mathbb{R}^r$, is the space of the end-effector velocities generated by joint velocities. If the Jacobian is full rank:

$$dim\big(R(\boldsymbol{J})\big) = r \qquad dim\big(N(\boldsymbol{J})\big) = n - r \ ,$$

holds

$$dim\big(R(\boldsymbol{J})\big) + dim\big(N(\boldsymbol{J})\big) = n.$$

where $\boldsymbol{q} \in \mathbb{R}^n$.

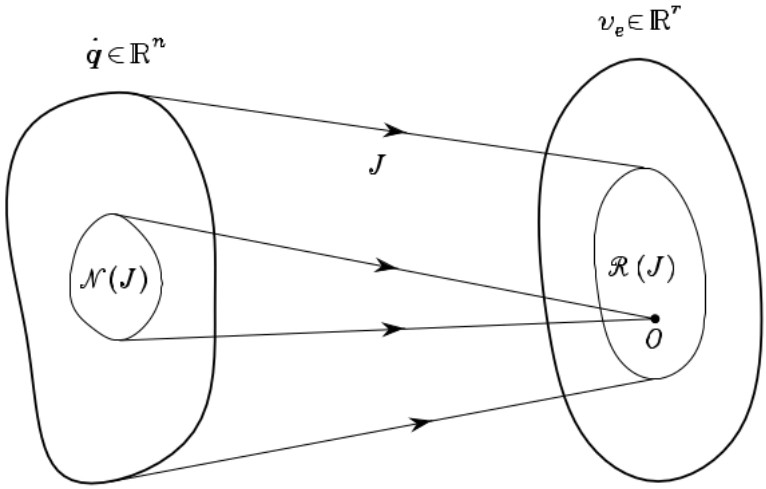

Figure 2.2: Mapping between configuration velocity space and Cartesian velocity space.

Instead, if the robot is in a singular configuration, the dimension of the range space decreases while the dimension of the null space increases.
For redundant robots, the subspace $N(\boldsymbol{J}) \neq \emptyset$, and then it is possible to manage redundant DOFs. Being $\dot{\boldsymbol{q}}_a$ the solution of (2.17) and $\boldsymbol{P}$ an $n \times n$ matrix such that

$$R(\boldsymbol{P}) \equiv N(\boldsymbol{J})$$

the joint velocity vector $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_a + \boldsymbol{P}\dot{\boldsymbol{q}}_b$, with arbitrary $\dot{\boldsymbol{q}}_b$, is still a solution of (2.17). In fact

$$\boldsymbol{J}\dot{\boldsymbol{q}} = \boldsymbol{J}(\dot{\boldsymbol{q}}_a + \boldsymbol{P}\dot{\boldsymbol{q}}_b) = \boldsymbol{J}\dot{\boldsymbol{q}}_a + \boldsymbol{J}\boldsymbol{P}\dot{\boldsymbol{q}}_b = \boldsymbol{J}\dot{\boldsymbol{q}}_a = \boldsymbol{v}_e \tag{2.48}$$

since $\boldsymbol{J}\boldsymbol{P}\dot{\boldsymbol{q}}_b = 0$ for any $\dot{\boldsymbol{q}}_b$. This result is fundamental, in order to exploit and take advantage of the redundant DOFs of a manipulator, offering the opportunity of choosing arbitrarily the vector $\dot{\boldsymbol{q}}_b$ of joint velocities. In fact, the effect of $\dot{\boldsymbol{q}}_b$, does not change the end-effector pose, but allow to change the robot configuration. A good choice for $\boldsymbol{P}$ in redundant manipulators is, as did in equation 2.18,

$$\boldsymbol{R} = \boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J}. \tag{2.49}$$

Prioritized order by projecting task velocities through the null spaces of higher-priority tasks

Through null space projector, it is possible to derive a *prioritized order* for the tasks that have to be executed by a robot. The higher priority are tasks that must be accomplished first and their execution must be guaranteed, while lower priority are less critical and should be performed only if they do not interfere with higher-priority tasks. Since with null space projection the first task is guaranteed and the second one acts without changing the end-effector position, the higher priority task come first into the null space projection. Then, following the priority order the other tasks are recursively projected into the null space of higher priority task with respect to them.
Then, multiple tasks can be arranged in priority. Consider, as an example, three possible task that have to be executed with a priority order in an high redundant system (n DOF). The velocities are denoted as $\dot{\boldsymbol{q}}_1$, $\dot{\boldsymbol{q}}_2$ and $\dot{\boldsymbol{q}}_3$ and their Jacobians as $\boldsymbol{J}_1$, $\boldsymbol{J}_2$ and $\boldsymbol{J}_3$. The corresponding null-space projector of the first task is defined as

$$\boldsymbol{N}_1 = \boldsymbol{I}_n - \boldsymbol{J}_1^{\dagger}\boldsymbol{J}_1, \tag{2.50}$$

while the null-space projector for tasks 1 and 2 is then defined as

$$\boldsymbol{N}_{12}^{A} = \boldsymbol{I}_n - \boldsymbol{J}_{12}^{A\dagger}\boldsymbol{J}_{12}^{A}, \tag{2.51}$$

where $\boldsymbol{J}_{12}^{A}$ is the augmented Jacobian of tasks 1 and 2 is given by stacking the two independent task Jacobians:

$$\boldsymbol{J}_{12}^{A} = \begin{bmatrix} \boldsymbol{J}_1 \\ \boldsymbol{J}_2 \end{bmatrix}. \tag{2.52}$$

Then, the desired velocity, of the prioritized order combination, can be found as

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_1 + \boldsymbol{N}_1\dot{\boldsymbol{q}}_2 + \boldsymbol{N}_{12}\dot{\boldsymbol{q}}_3. \tag{2.53}$$

This rule can be recursively applied to general cases with an higher number of tasks.

2.2 Reinforcement learning

Reinforcement Learning (RL) is a branch of machine learning where agents learn to make decisions by interacting with an environment to maximize cumulative rewards.

2.2.1 Policy

A *policy* is a crucial concept in reinforcement learning, in order to describe the behaviour of an agent. Denoted as π, it is a mapping from states of the environment to actions. In this work the policy is always deterministic once parameters are defined, meaning that at a certain state corresponds a specific action,

$$a \in A = \pi(s \in S), \tag{2.54}$$

where S is the set of all the states and A the set of all the possible actions.
The objective in reinforcement learning is often to find an optimal policy π^* that maximizes the expected cumulative reward, or minimizes costs.

2.2.2 Genetic Programming

The *Genetic programming* is a domain-independent method that breeds a population of computer programs to solve a problem. It is a systematic method for getting computers to automatically solve a problem, starting from an high-level of what needs to be done.
Genetic programming acts iteratively on the population, transforming this in a new generation of programs. This happens applying operations similar to natural genetic operations. The process can be repeated until a candidate solution for to the problem is found.
Programs are in general expressed as threes and can have subroutines and, each of them is a candidate solution to the problem. The human user typically specify the set of terminals, the set of primitive functions, the fitness measure, the parameters to control the run and termination criterion.
Once the preparatory steps are concluded the first run is launched. Each run of Genetic Programming can be seen as a competition for survival among different programs. At the end of it, only a part of the population is allowed to survive, basing on fitness function. Then new elements are introduced into the new population with genetic operations analogs of naturally ones. Iteratively, a population

of computer programs is transformed into a new generation population. From the last population, the solution of the problem can be selected, for example choosing the program with the best fitness.

Stack of tasks structure

In genetic programming, the organization of tasks can be conceptualized as a *stack of tasks* to represent how the process of solving a problem is structured and executed, instead of sub-tree (which, however, remains the best representation in order to understand the behaviours of genetic programming). This approach leverages the concept of breaking down the main task into smaller, and manageable sub-tasks, each contributing to the overall solution. Each sub-task represents a specific aspect of the problem that needs to be solved. This hierarchical organization allows for a modular approach to problem-solving.
Tasks are organized in a stack where the execution follows a last-in, first-out order of priority and they are all executed together and combined through null space projector in the case of this work (section 2.1.6). The fitness function is evaluated basing on the result of all the combined tasks.
Each task in the stack, has its own parameters that can be changed through the genetic process. Also the order of the priority can be changed through evolution.

1^{st} task
2^{nd} task
3^{rd} task
4^{th} task

Figure 2.3: Example of stack of tasks.

Fitness Function and Selection (Genetic Competition)

In genetic programming, the *fitness function* is a critical component that measures how well a given program or solution performs with respect to the problem being solved. Essentially, it evaluates the quality of individual solutions within the population. The goal of the GP algorithm is to evolve solutions with increasingly better fitness values over successive generations. The final result of a genetic programming algorithm will be heavily influenced by the fitness function since, typically, the aim is to minimize or maximize this value.
Fitness function plays a crucial role in the *selection* of algorithms among the population, guiding the evolutionary process by evaluating and selecting individuals

based on their performances. As an example, in this work, the fitness function should be minimized as possible to increase the performances of what is required by the user. After the running of a generation of algorithms, they are randomly paired and only the one with the lowest fitness function survives and can generate an offspring or being part of the new generation, being a candidate for the final solution. The other one, with the highest value, will be lost forever.

Crossover

Crossover creates a new offspring program for the new population, by recombining randomly chosen parts from two select parent programs.

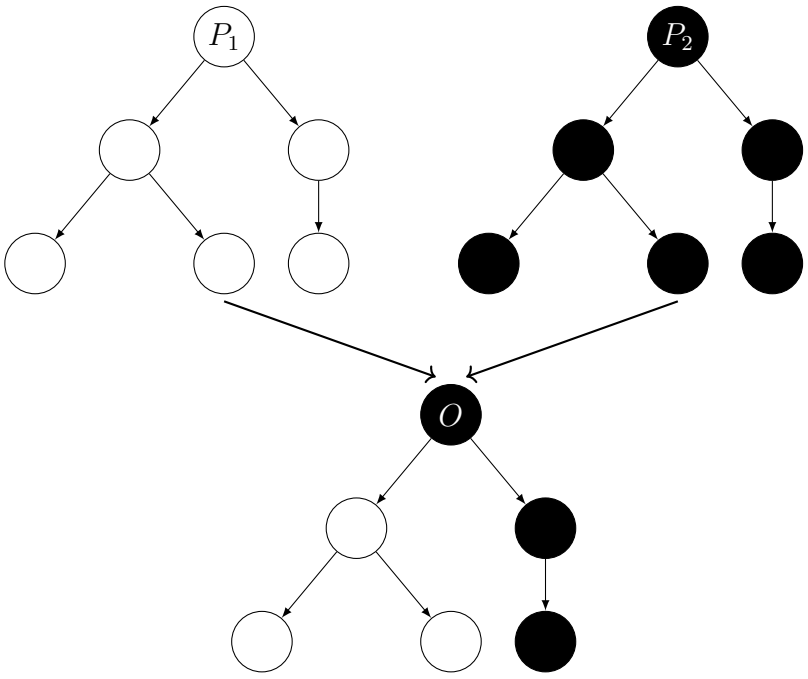

Figure 2.4: Example of crossover.

In figure 2.4, P_1 and P_2 represent the two parents. The elements of each are represented with the same color. O represent a case of the offspring generated by the two parents. The elements which compose O, preserve the same color of the corresponding parent.

Mutation

Mutation creates one new offspring program for the new population by randomly mutating a randomly chosen part of a parent program.

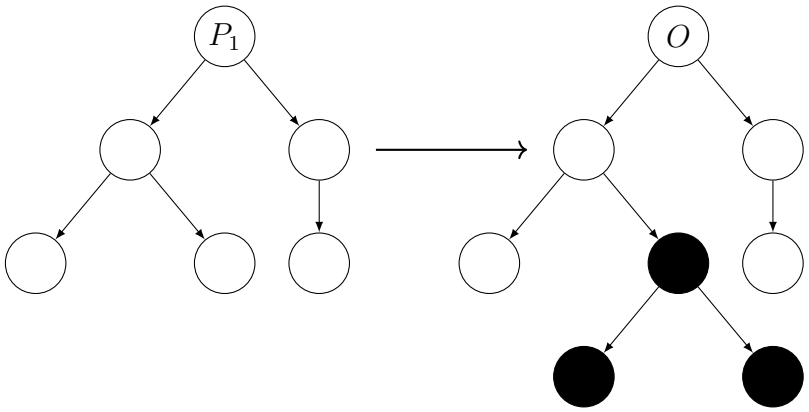

Figure 2.5: Example of mutation.

In figure 2.6, P is the parent while O is the offspring generated through mutation. The parts represented in black are the mutated parts.

Reproduction

With *reproduction*, the algorithms are simply copied into the new population. In this way if it has good performances, it will be maintained also in the next generation of algorithms.

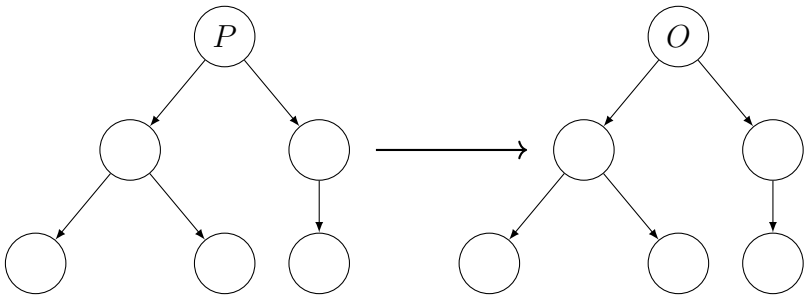

Figure 2.6: Example of reproduction.

Chapter 3

Simulation framework

3.1 Work environment

An *environment* is created using Robosuite's functions at the beginning of each simulation. This will initialize a MuJoCo environment selecting its name, load the model of the robots and activate the selected options. Then the velocities needed to solve tasks can be computed and an action, as vector of desired velocities for the joints, can be submitted to the robot at each time step. Each time step of the simulation lasts 0.002 s, as default option.
It is possible, for the training part in particular, to let the robot work using only kinematics in the simulation, without useless dynamics. This will make learning faster, reducing computational time. However, since MuJoCo libraries can not be modified, this imply the deactivation of useful features too, like detection of geometries collision. Since this, during the learning phase, it is possible to relay only on sensors (section 3.4) for the collision detection. At the beginning of each learning simulation, the position of the arm joints are set randomly in their range, in order to derive a solution which is not configuration-dependent.
Different type of environment (called arena) where implemented, starting from the creation of the *.xml* file of the arena tailored on the specific needs of each situation. All the environments can be easily modified accessing the related files. The default environments were not used, since they do not foresee the motion of the robot with its base in the environment, but only for manipulation tasks.
RoboSuite uses a $r - zxy$ Euler angle representation, where r indicates that the frame is a rotating one. However, all the results reported in this thesis are expressed in the order $[\theta_x, \theta_y, \theta_z]$ for a better readability.

3.1.1 Empty environment

This environment is made only by a floor on which the Baxter robot can move, without predefined obstacles. In this environment it was possible to develop and test the the algorithms of the individual tasks and their combination in absence of obstacles. For the obstacle avoidance task, they can be added through the *.xml* file of the arena.

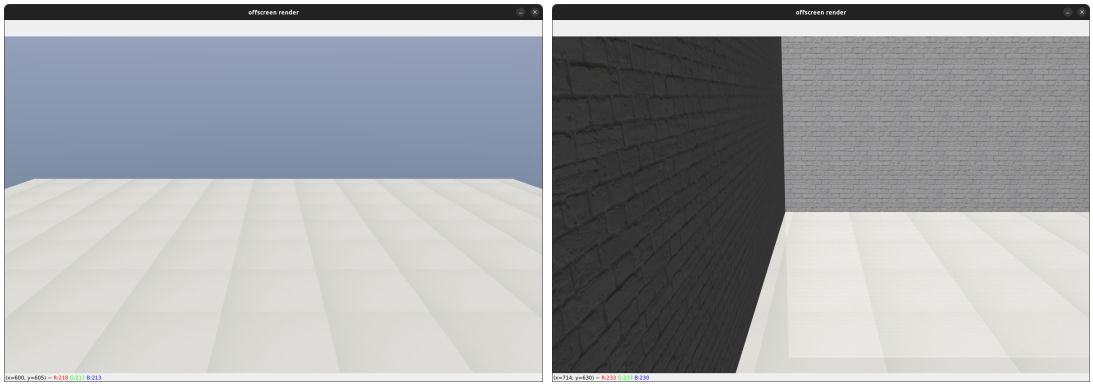

Figure 3.1: Examples of empty arena (left) and empty arena with some walls as obstacles (right).

3.1.2 Static obstacles environment

This arena was implemented to train the robot in an environment full of obstacles. Different type of geometries can be added as obstacles like spheres, cylinders or boxes.

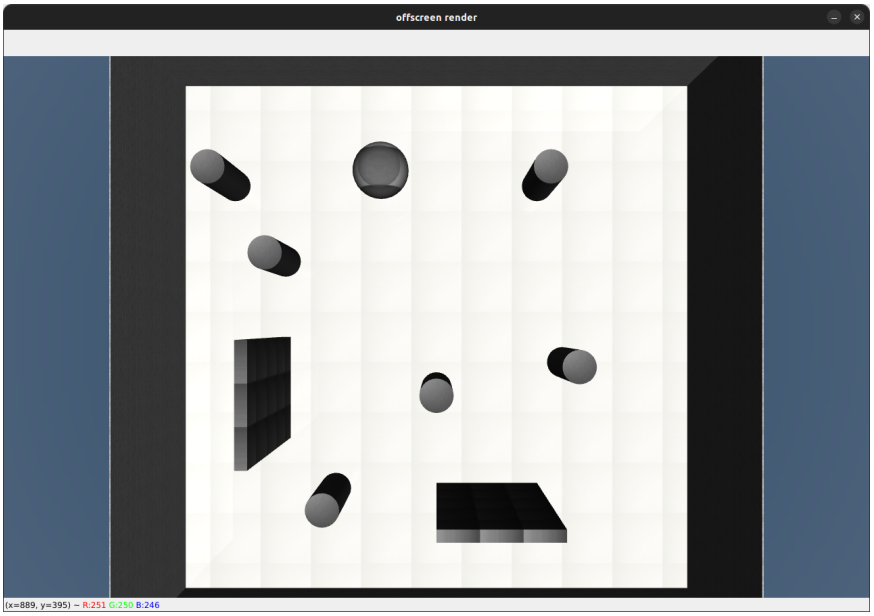

Figure 3.2: Examples of an aerial view of the obstacles arena.

Figure 3.2 shows an aerial view of the obstacle arena. It is possible to note different type of geometries as obstacles. The environment can also be modified accessing the relative *.xml* file following the necessities of the user, adapting it to different kind of situations.

3.1.3 Dynamic obstacles environment

This arena was implemented to train and test the robot with dynamics obstacles, which change their position during simulation time.

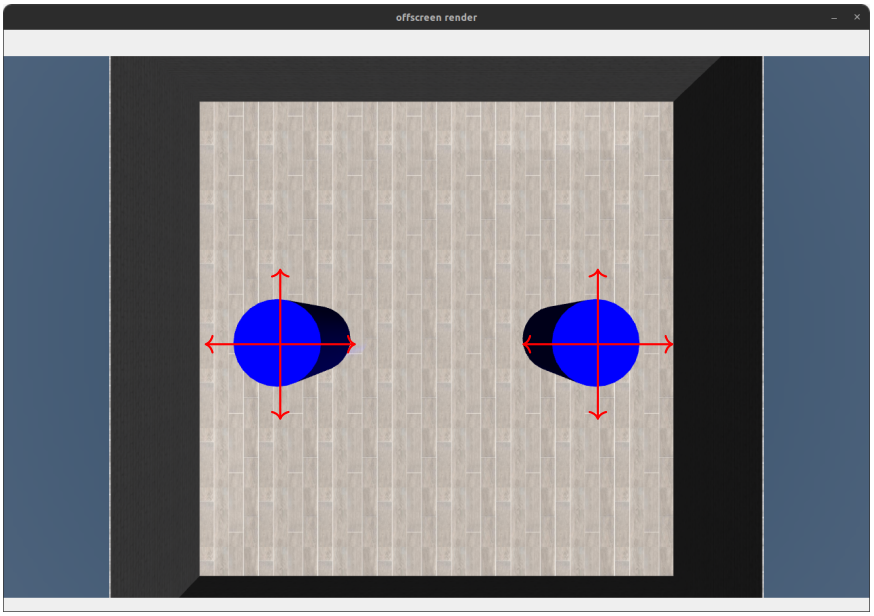

Figure 3.3: Aerial view of the dynamic obstacles arena.

An example of the environment is reported in figure 3.3.
It is possible to see how the movement of the two dynamic obstacles was implemented. They can freely move, changing position on the floor plane, thanks to the presence of two prismatic joints. One allows movement along the x axis of the world reference frame, the other one along the y axis. The possible directions of movement are showed by the red rows.
Accessing the *.xml* and *.py* of the environment, it is possible to add more static or dynamic obstacles, tailoring it on the specific needs of the user.

3.1.4 Pick & place environment

This environment was built in order to test a stack of tasks with a general duty. In fact, the robot has to pick an object and place it in another position.

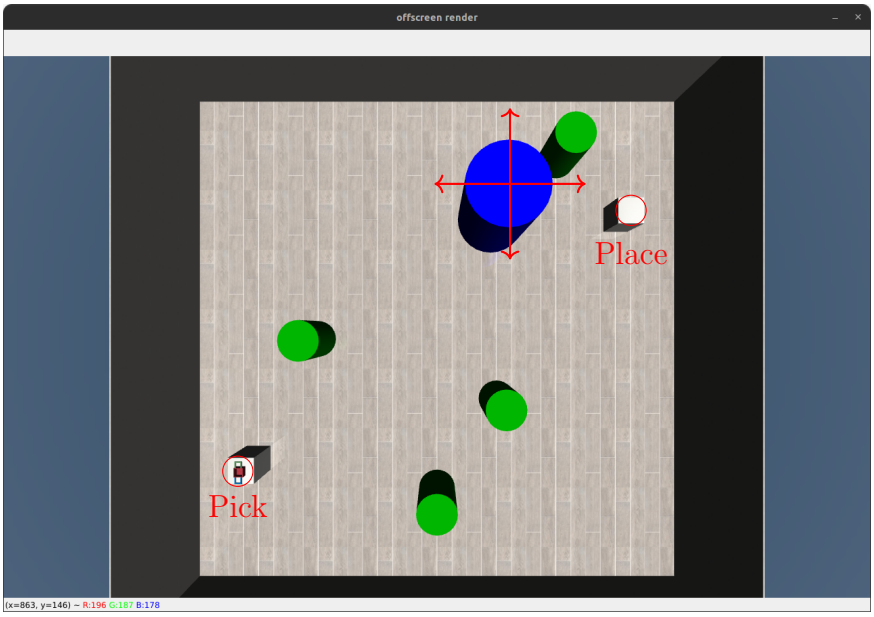

Figure 3.4: Aerial view of the pick and place arena.

In figure 3.4 it is possible to observe an aerial view of the built arena. The obstacles in green are all fixed and they are positioned in such a way the robot can move between them without being stacked but, it is disturbed by them along its path. The blue obstacle, instead, can be moved during the simulation thanks to the presence of two prismatic joints which move along the x and y axes of the reference frame of the room, as showed by red rows in to the picture.

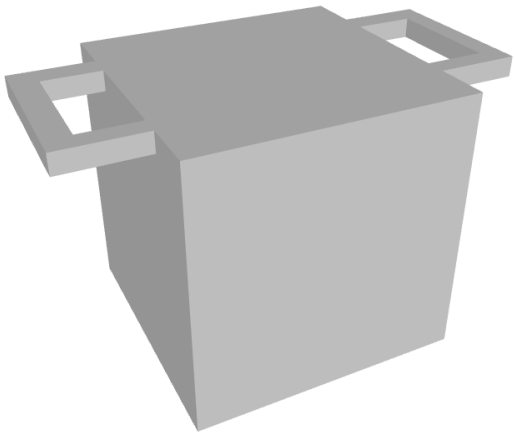

Figure 3.5: Mesh of the pot with two handles.

Then, depending on the specific situation, the obstacle can be moved to simulate, for example, an operator present in the working area of the robot. In figure, also the *pick* and the *place* location are shown. In particular, the robot has to pick with two arms a pot with two handles, one for each arm, or a cylinder with one arm. Also in this case, the environment can be modified accessing the *.xml* file of it, in order to change obstacles, pick and place position or object.

3.2 Motion of the base

The Baxter robot is not equipped with a *mobile base* in the RoboSuite simulation framework, but only with a static pedestal like all the other present robots.
In order to provide the robot the capability to move freely in the space, three more joints are added at the base of the mount of the robot itself. Two prismatic joints give to the robot the mobility on the floor along x and y coordinate axes. A third revolute joint, provides the robot's base with the possibility to rotate along the z axis. So that, the robot's base gains three additional degrees of freedom, which allow the base to reach any pose in a 2-dimensional environment (i.e. the floor plane). Additionally, thanks to the addition of more degrees of freedom, it is possible to use an high redundant kinematic chain of 10 degrees of freedom.
The mount's frame at instant of time zero, $t = 0\ s$, will be the world frame for the entire duration of the simulation. Namely all the positions and orientation of a simulation are expressed with respect to this frame.
Since the motion of the base is not included in RoboSuite, it was necessary to define the parameters in order to simulate the motion on a floor with friction and real joints with damping in a full dynamics case. For the 3 additive joints, the following parameters where added:

Joint	Type	Damping	Friction loss	K_p
q_{1b}	prismatic	0.7	0.9	0.2
q_{2b}	prismatic	0.7	0.9	0.2
q_{3b}	revolute	0.5	0	0.03

Table 3.1: Chosen parameters of the mobile base.

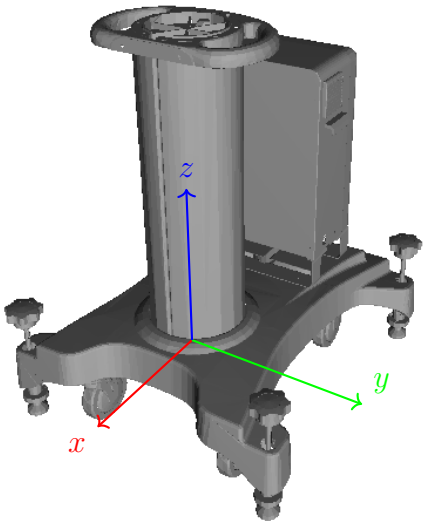

Figure 3.6: Pedestal of the robot with frame.

Note that the proportional gain K_p for the prismatic joints is almost 10 times bigger with respect to the arm's gain. This happens because the previous gain was too low and the robot was poorly reactive. This will prevent oscillatory behaviours of the base around the desired position with high overshoots and without convergence. Following, the parameters of the actuators controller for prismatic joints are reported in table 3.2.

Gain k_p	Input range	Output range	Ramp ratio
0.2	[-1, 1] m/s	[-0.5, 0.5] N	0.2 $\frac{N}{m/s}$

Table 3.2: Prismatic joint controller parameters.

Base joints performances

In this subsection, the *performances* of the new implemented joints, that simulate the movement of the base, are reported. This performances are not a characteristic of the robot, but are the result of the parameters chosen and reported in table 3.1 for the implementations of this 3 additive joints.

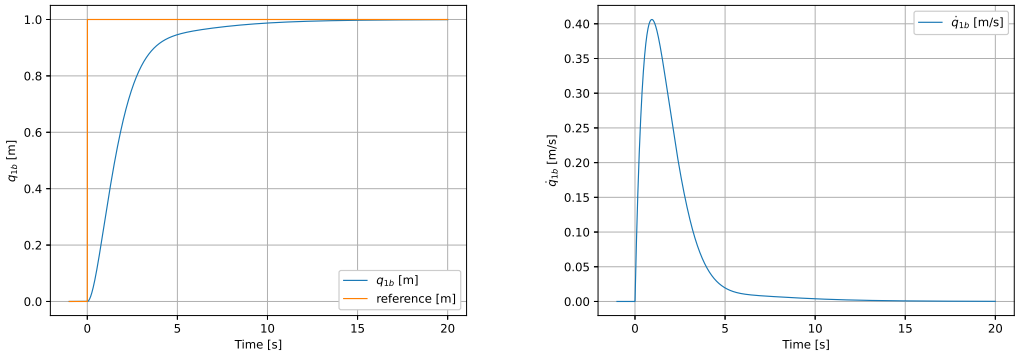

Figure 3.7: q_1^b joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot	Steady state error
3.726	5.208	0.999 [m] (0%)	1.536 e-3 [m]

Table 3.3: q_1^b joint performances at position step response.

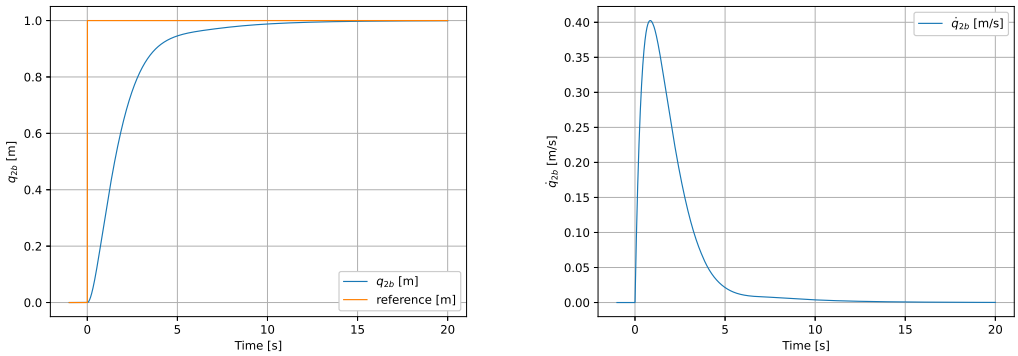

Figure 3.8: q_2^b joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot	Steady state error
3.802	5.223	0.999 [m] (0%)	1.503 e-3 [m]

Table 3.4: q_2^b joint performances at position step response.

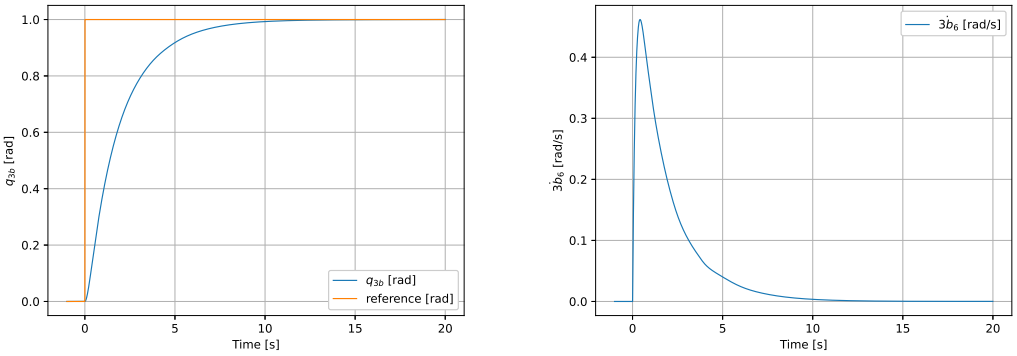

Figure 3.9: q_3^b joint performances. Position step response (left) and velocity (right).

Rise time [s]	Settling time [s]	Overshoot	Steady state error
4.587	5.996	0.999 [m] (0%)	5.731 e-6 [m]

Table 3.5: q_3^b joint performances at position step response.

3.3 Master and Slave arm

One arm at a time can take control of the robot's mobile base (the *Master* arm), increasing the number of degrees of freedom of the kinematic chain to 10, making the master system high redundant. The pose of this arm is expressed with respect to the world frame. The other arm (the *Slave* arm) can not control the base, and its pose is expressed with respect to the base frame (T_0), the same showed in figure 1.6. In this way the base can move accordingly with one of the two arms in order to solve a task, while the other arm moves with the base.
In addition, the base can move as master alone (3 DOF kinematic chain) and both the arms are considered as slaves. However, the base alone can only satisfy a pose in a 2-dimensional environment (3 DOF, namely a pose on a plane) without redundancy for this particular case.

Singularities

It may happen that the robot, while solving a task, reaches a *singular configuration* (section 2.1.1). This situation implies that the robot becomes uncontrollable, loosing degrees of freedom. Furthermore, in a neighborhood of the singular configuration, the velocities imposed to the robot are really high and they may cause a collision, and then a failure.

Such that, the rank of the Jacobian matrix of the current kinematic chain is checked at each iteration. If, the matrix is not full rank, the simulation is stopped and cost 10^{10} is assigned to the stack. This implies that during the genetic selection it is discarded if compared with a stack which did not produce a failure.

3.4 Range-Finder Sensors

In order to avoid obstacles (section 2.1.3), *range finder sensors* where attached to the robot in different points.

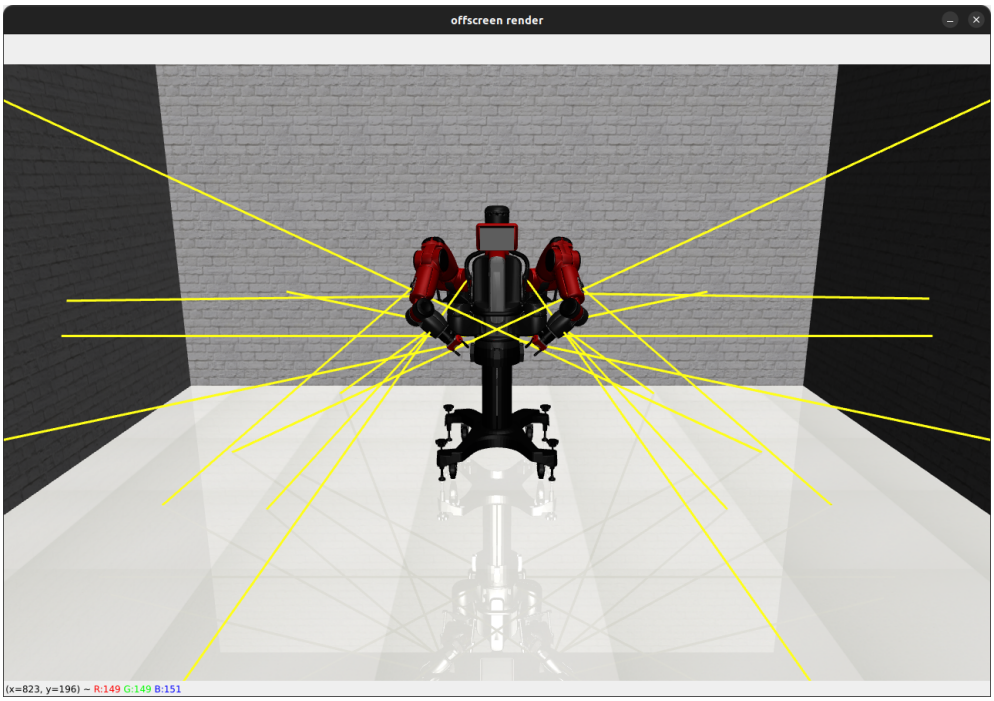

Figure 3.10: RoboSuite rendering of the Baxter robot with range-finder sensor ranges in yellow.

These sensors are active when there is an obstacle in their range, along the given direction. In that case the k^{th} sensor returns the distance d_k from them and the sensed object, otherwise they return -1. However, only distances smaller or equal to the desired range r_k activate the obstacle avoidance task, and if the distance is such that $d_k > r_k$ it is set to -1 by the algorithm.
Since, during learning, in an only kinematic environment it is not possible to detect collisions between geometries the collision checking must relies only on range finder sensors. If a sensor reports a distance smaller than $d = 0.01\ m$, the simulation is stopped and, it is considered like a collision. Cost 10^{10} is assigned at the stack implying that during the genetic selection it is discarded if compared

with a stack which did not produce a failure.

3.4.1 Map of the sensors

8 depth sensors were used to cover the motion of the base while 12 for each arm were used, bringing the total to 32 sensors for the whole Baxter robot. Following, a description of their position is reported.

Mobile base

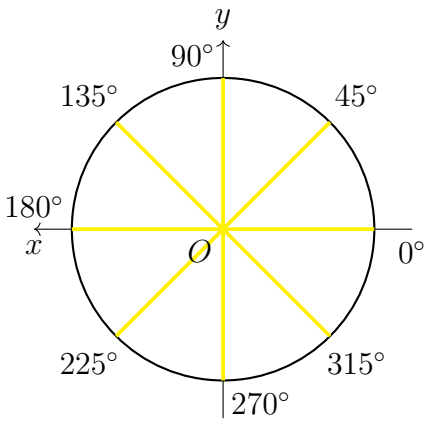

Figure 3.11: Sensors map of the mobile base.

The mobile base of the robot is covered by 8 sensors, at the height of 0.924 m from the mobile base frame, on the top of the base itself. They are positioned with a radius of 0.21 m from the center of the pedestal mesh, in order to avoid to sens the mesh itself instead of an obstacle, and they point toward the environment with direction perpendicular to the tangent of the pedestal mesh in that point. The exact position and the direction of the sensor are computed at each 45°, as showed in picture 3.11 where the axes are the same of the base frame.

Arm

The arms own 12 sensors each. This allows to define a map of sensors divided in 3 possible sites. For each site, a different number of sensors is present. This consents a better sensing of obstacles, avoiding useless sensors.
Following, the image 3.12 shows where the different sites are implemented on the left arm, for the right it is the same. The first site is centered at 0.1 m from the joint q_3 along its z axis, the second is centered at 0.1 m from the joint q_5 along its z axis and the third site is centered at 0.0 m from the joint q_7 along its z axis.

Figure 3.12: Sensor sites of the left arm, with pedestal reference frame.

Then, as did for the pedestal, each site can be think as a circumference of radius 0.075, 0.05 and 0.075 m respectively, which lies on a plane parallel to the one generated by x and y axes of the cited joints, with z axis oriented in the same way. The direction of the sensors are reported in figure 3.13, in which the sites of left arm can be seen looking at the straight arm from the end-effector to the arm mount. The maps of the right arm are mirrored with respect to the y axis.

3.4.2 Position of the sensed obstacle p_k

The distance d_k of the k^{th} sensor is not the only information needed from sensors to avoid obstacles, in spite of it is the only one given by the sensors. In order to find the gradient of the pseudo-energy (2.39) and solve this task, also the vector joining the origin of the sensor and the obstacle $\boldsymbol{v}_{d_k}$ is needed. This information is not provided by the sensor, but it is derived from the geometry of the robot, knowing the exact position of the sensors.

$$d_k \boldsymbol{v}_{o_k} = \boldsymbol{v}_{d_k}, \tag{3.1}$$

where $\boldsymbol{v}_{o_k}$ is the versor associated to the k^{th} sensor frame with respect to the robot's base frame such that $||\boldsymbol{v}_{o_k}|| = 1$, from which follows that

$$\boldsymbol{p}_k = \boldsymbol{p}_{s_K} + d_k \boldsymbol{v}_{o_k} = \boldsymbol{p}_{s_K} + \boldsymbol{v}_{d_k} \tag{3.2}$$

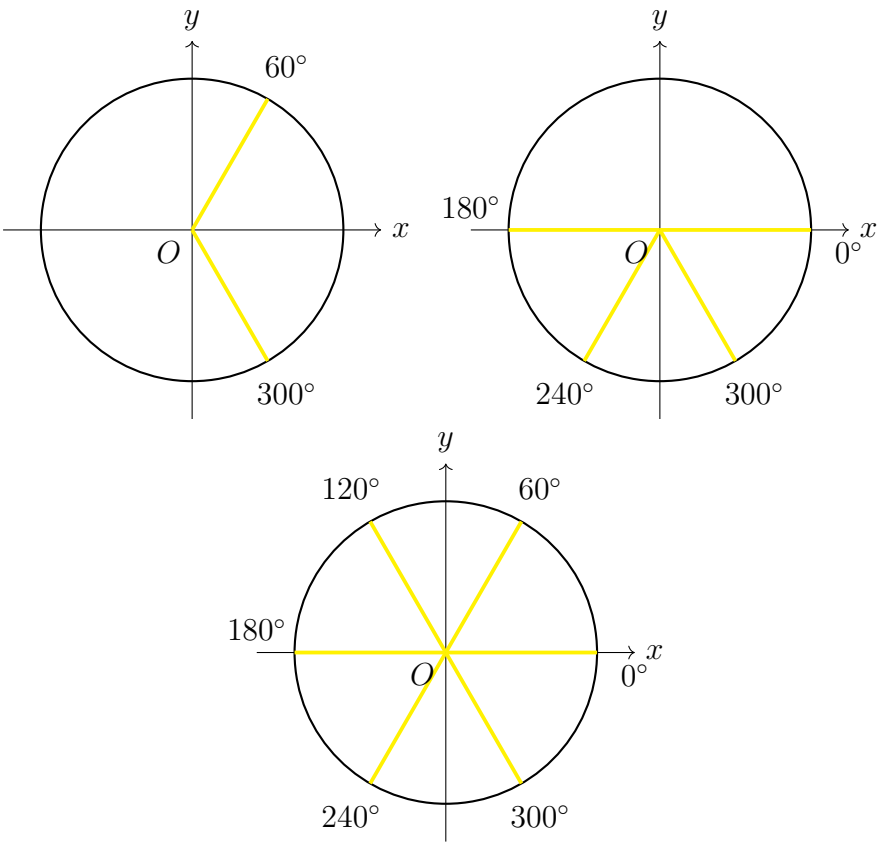

Figure 3.13: 1^{st}, 2^{nd} and 3^{rd} site map of the left arm.

where $\boldsymbol{p}_{s_K}$ is the position of the sensor with respect to the base frame.
Furthermore, knowing the position of the contact is fundamental in order to understand if the sensor was activated by an obstacle or by the robot itself, allowing to differentiate the two cases (section 3.4.3).

3.4.3 Self sensing for obstacle avoidance

The robot works differently if a sensor sens an obstacle or a part of the robot itself. In fact, if an obstacle is sensed, the robot avoid it if it is in the range of the distance r_k. But, if the sensor is activated by the robot, the range of action is reduced at s_k, with $s_k \leq r_k$. In this way, the robot is allowed to differentiate it's action, avoiding itself only in the case in which a collision is imminent. If this differentiation had not been implemented, the robot would have tried to escape from itself if r_k is big, preventing the achievement of positions close to the robot. On the other hand if r_k is small, the robot reacts to obstacles too late and a collision has high probability to happens.
Since this solution is applied, it is needed to identify the nature of the activation

of a sensor. So, to achieve this feature, the robot is completely mapped with respect to world reference frame, with simple fictitious geometries, which cover the entire structure. This is possible knowing the position of all the geometries of the robot with respect to the world frame.
Then, it is checked if the point in which the obstacle is sensed $\boldsymbol{p}_{s_K}$ (3.2) is included in one of these geometries. If it is, the obstacle is considered as the robot itself and s_k is used to solve the task for that sensor in place of r_k.

Spherical Geometry

Spherical Geometries are defined with a center $\boldsymbol{C}_s$, attached to a geometry of the robot, and a radius r_s. Then, if

$$\sum_{i=1}^{3}(\boldsymbol{p}_{s_K i} - \boldsymbol{C}_{si})^2 \leq r_s^2 \tag{3.3}$$

it is considered as a self sensing.

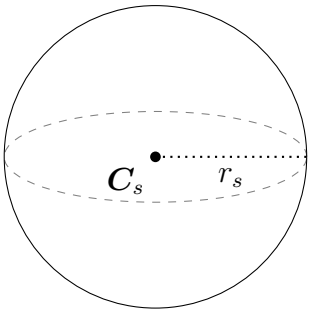

Figure 3.14: Spherical geometries with center reference and radius.

Cylindrical Geometry

Cylindrical Geometries are defined as a center $\boldsymbol{C}_c$, attached to a geometry of the robot, and half height $\frac{h_c}{2}$ and a radius r_c. The center is located in the middle of the main axis of the cylinder. Then, given $\boldsymbol{p_1} = \boldsymbol{C}_c - \frac{h_c}{2}\boldsymbol{v_c}$ and $\boldsymbol{p_2} = \boldsymbol{C}_c + \frac{h_c}{2}\boldsymbol{v_c}$ as the centers of the base circumferences where $\boldsymbol{v_c}$ is the unit vector describing the orientation of the cylinder with respect to the of the base world frame, if

$$(\boldsymbol{p}_{s_K i} - \boldsymbol{p_1}) \cdot (\boldsymbol{p_2} - \boldsymbol{p_1}) \geq 0 \tag{3.4}$$

$$(\boldsymbol{p}_{s_K i} - \boldsymbol{p_2}) \cdot (\boldsymbol{p_2} - \boldsymbol{p_1}) \leq 0 \tag{3.5}$$

$$\frac{(\boldsymbol{p}_{s_K i} - \boldsymbol{p_1}) \cdot (\boldsymbol{p_2} - \boldsymbol{p_1})}{|\boldsymbol{p_2} - \boldsymbol{p_1}|} \leq r_c \tag{3.6}$$

it is considered as a self sensing.

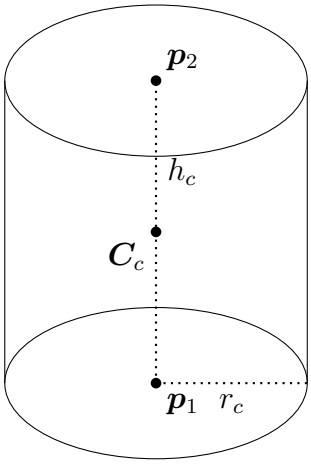

Figure 3.15: Cylindrical geometries with references and parameters.

Used geometries

Here, a list of the used fictitious geometries with parameters and the respective geometry of the robot are reported with the center.

Geometry name	parameters	Robot geom.	Center [x, y, z]
right_mount_low	[0.08, 0.18]	coll_base	[0.09 -0.26 0.211]
right_s0_collision	[0.08, 0.08]	right_upper_shoulder	[0, 0, 0.1361]
right_s1_collision	[0.11, 0.1]	right_lower_shoulder	[0, 0, 0]
right_e0_collision	[0.096, 0.17]	right_upper_elbow	[0, 0, 0.15]
right_e1_collision	[0.0695, 0.096]	right_lower_elbow	[0, 0, 0]
right_w0_collision	[0.07, 0.044]	right_upper_forearm	[0, 0, -0.044]
right_upper_forearm_col	[0.076, 0.12]	right_upper_forearm	[0, 0, 0.11]
right_w1_collision	[0.06, 0.083]	right_lower_forearm	[0, 0, 0]
right_w2_collision	[0.07, 0.0825]	right_wrist	[0, 0, 0]

Table 3.6: Right arm geometries, in this case only cylindrical geometries were used.

The geometries attached to the left arm, are the same of the right one. So that, for the arms only right geometries are reported.

Geometry name	Type	Parameters	Robot geom.	Center [x, y, z]
ped_col	cylinder	[0.4, 0.2]	pedestal	[0, 0, 0.1]
ped_col_1	cylinder	[0.17, 0.25]	pedestal	[0.33, 0.3, 0.11]
ped_col_2	cylinder	[0.17, 0.25]	pedestal	[0.33, -0.3, 0.11]
ped_col_3	cylinder	[0.17, 0.25]	pedestal	[-0.33, 0.3, 0.11]
ped_col_4	cylinder	[0.17, 0.25]	pedestal	[-0.33, -0.3, 0.11]
ped_col_5	cylinder	[0.25, 0.6]	pedestal	[0, 0, 0.25]

Table 3.7: Pedestal geometries.

Geometry name	Parameters	Robot geom.	Center [x, y, z]
collision_head_link_1_col	[0.22]	collision_head_link_1	[-0.07, -0.04, 0]
collision_head_link_2_col	[0.22]	collision_head_link_2	[-0.07, 0.04, 0]

Table 3.8: Head geometries, in this case only cylindrical geometries were used.

Geometry name	Type	Parameters	Robot geom.	Center [x, y, z]
body_low_col	sphere	[0.2]	coll_base	[0, 0, 0]
body_mid_col	cylinder	[0.22, 0.1365]	coll_base	[0, 0, 0.273]
body_high_col	sphere	[0.19]	coll_base	[0, 0, 0.443]
base_high_col	cylinder	[0.2, 0.1]	coll_base	[0, 0, -0.1]
base_mid_col	cylinder	[0.11, 0.2]	coll_base	[0, 0, -0.4]
left_back	sphere	[0.13]	coll_base	[-0.14, 0.12, 0.48]
right_back	sphere	[0.13]	coll_base	[-0.14, -0.12, 0.48]
right_mount	sphere	[0.19]	coll_base	[0.09, -0.26, 0.385]
left_mount	sphere	[0.19]	coll_base	[0.09, 0.26, 0.385]

Table 3.9: Body geometries.

Sensing in working area of the robot

The robot was made able to handle with objects considering, for example, the table on which the object is placed in the same way as a wall in terms of collision avoidance. In order to obtain a differentiation between objects, the working area of the robot is subject to the same rules of collision that governs the self collision avoidance, with a smaller radius $s_k \leq r_k$.
The working area is considered to be a sphere of radius 0.4 m, positioned in $\boldsymbol{p}^T = [0.2, 0, 1.2]$ $[m]$, with respect to the robot base frame.

So that the robot can work with objects in front of it in a finer way, without reducing r_k.

Base and Arm r_k

Since the pedestal have base with dimensions reported in figure 1.4, a small value of r_k would not avoid collisions between this part of the robot and the environment. On the other hand, a bigger r_k would prevent the arms to work close to some obstacles and reach the desired precision. So that, two different r_k were implemented, one for the base sensors and the other one for the arm sensors, so that

$$s_k \leq r_{k_{arm}} \leq r_{k_{base}}. \tag{3.7}$$

This helps the arms to reach higher precision close to obstacles (which are outside the working area), and the base to prevent collisions during navigation.

3.5 Respect of the joint limits

Since, during train and validation of the robot, the dynamics are no longer present in the simulation in order to save computational time, the *joint position limits* could be overcome. So that, the velocity of each joint i is constrained to be

$$\varepsilon(q_{min_i} - q_i) \leq \dot{q}_i \leq \varepsilon(q_{max_i} - q_i) \tag{3.8}$$

where ε is a positive damping parameter, q_{min_i} is the minimum value that q_i could reach and q_{max_i} is the maximum. This imply that if the joint is in the position of maximum (minimum) the velocity can not be bigger (lower) than zero, and then limits can not be overcome. Also, in a neighborhood of the limit, the joint is slowed towards it.

$$q_{max} = [1.70168,\ 1.047,\ 3.05418,\ 2.618,\ 3.059,\ 2.094,\ 3.059] \tag{3.9}$$

$$q_{min} = [-1.70168,\ -2.147,\ -3.05418,\ -0.05,\ -3.059,\ -1.57,\ -3.059] \tag{3.10}$$

Moreover the *velocity joint limits* can be overcome and then, being $\dot{q}_{lim}$ the absolute value of that limit, the velocity is also constrained to be

$$-\dot{q}_{lim} \leq \dot{q}_i \leq \dot{q}_{lim}. \tag{3.11}$$

Since the limit of the velocity that the real controllers of the Baxter arm can apply is $[-1,\ 1]\ m/s$ (table 1.2), $\dot{q}_{lim}$ was chosen to replicate this range.

3.6 Chattering avoidance

The presence of an obstacle along the Inverse Kinematic trajectory, or really close to it, may cause *chattering*. This happens due to the fact that the Jacobian matrix of the obstacle avoidance task (equation 2.38), loose rank when the obstacle is no longer sensed, and then a secondary task in the null space projection is allowed to have more freedom of movement. At this point, the secondary task may bring back the arm to the previous position and the obstacle is sensed again, augmenting again the rank of the Jacobian matrix of the obstacle avoidance task. This process may continue changing the rank of $\boldsymbol{J}_o(\boldsymbol{q})$ at each time-step causing chattering, whose result is instantaneous high velocities in the joint space and a stall position. In order to avoid this type of occurrence, the combination of the tasks is given (as did in [5] by P. Falco et al) by a convex combination of tasks:

$$\dot{\boldsymbol{q}} = (1 - \lambda(d))\dot{\boldsymbol{q}}_g + \lambda(d)(\dot{\boldsymbol{q}}_o + (\boldsymbol{I} - \boldsymbol{J}_o^{\dagger}(\boldsymbol{q})\boldsymbol{J}_o(\boldsymbol{q}))\dot{\boldsymbol{q}}_g), \tag{3.12}$$

where $\dot{\boldsymbol{q}}_o$ and $\boldsymbol{J}_o(\boldsymbol{q})$ are the velocity and the Jacobian given by the solution of the obstacle avoidance task, and $\dot{\boldsymbol{q}}_g$ the velocity given by the combination of the remaining tasks. d is the minimum distance sensed by the sensors and $\lambda(d)$ is a weighting function

$$\lambda(d) = \begin{cases} 1 & d \leq f \\ 0 & d > f \end{cases} \tag{3.13}$$

where f is an activation threshold to be selected. To avoid undesired effect that can generate vibrations of the mechanical structure of the robot, $\lambda(d)$ can be chosen as a smooth sigmoidal function:

$$\lambda(d) = \frac{1}{\pi} arctan(-K(d - f)) + \frac{1}{2} \tag{3.14}$$

or a piece-wise linear function, i.e.

$$\lambda(d) = \begin{cases} 1 & \text{if } d \leq f - \Delta/2 \\ 0 & \text{if } d \geq f + \Delta/2\ . \\ \frac{1}{2} - \frac{(d-f)}{\Delta} & \text{otherwise} \end{cases} \tag{3.15}$$

In this way, it is defined a zone in which the priority is not define between the two tasks, outside of it the priority is established. $K > 0$ and Δ vary the slope of the function and the width of the middle zone. This last option was the chosen one for the implementation in this work.
The chattering, however, can not be completely avoided but only reduced in some cases.

3.7 Stack of Tasks

The execution of the algorithm is governed by the structure given by a *stack of tasks* (section 2.2.2), which contains the four different tasks with their parameters and the cost associated to it. The order of the tasks defines the priority in the combination via null space projection, namely the first task in the stack have higher priority with respect to the second one and so on. In first place in the stack the cost of the algorithm is reported.

$cost$	n_a	n_b	n_c	n_d

Figure 3.16: Stack of tasks.

Then, each single task $n_\#$ is made by:

$n_\#$	$active$	θ

Figure 3.17: Single task composition.

where $n_\#$ is a label of the task that has to be executed, the boolean value *active* indicates if the task is present or not in the computation of the final velocity and θ is the vector of parameters of each task. If a task is not active, the computation of the velocities and of the Jacobian will not be done for that specific task returning $\dot{\boldsymbol{q}} = \mathbf{o}_n$ and $\boldsymbol{J} = \mathbf{o}_{m \times n}$. This will help in saving computational time, avoiding useless computations.
In this work, the tasks are labeled as:

- n_1: Obstacle Avoidance task 2.1.3
- n_2: Inverse Kinematic task 2.1.2
- n_3: Manipulability Measure Maximization task 2.1.4

- n_4: Distance from Mechanical Joint Limits task 2.1.5.

All the required parameters for each task are defined in 3.7.1.
With each stack, it is possible to run an algorithm and store the information related to the cost function. Namely, a stack carries all the useful information in order to run and evaluate it.

3.7.1 Tasks parameters

Tasks have different *parameters* and they can be select among a certain range of values. This prevents issues due to instability, unfeasible combinations or useless parameters.
Obstacle avoidance parameters:

- rest length of the base springs, $r_{k_{base}} \in [0.4\,m,\ 2\,m]$
- rest length of the arm springs, $r_{k_{arm}} \in [0.1\,m,\ r_{k_{base}}]$
- rest length of the working area springs, $s_k \in [0.1\,m,\ r_{k_{arm}}]$
- function gain, $\gamma_o \in [0,\ 2]$
- trajectory time, $t_o \in [0\,s,\ 10\,s]$

Inverse kinematic parameters:

- function gain, $\gamma \in [0,\ 2]$
- trajectory time, $t_f \in [0\,s,\ 10\,s]$

Maximization of the manipulability parameters:

- function gain, $k_0 \in [1,\ 100]$

Maximization of the distance from mechanical joint limits parameters:

- function gain, $k_0 \in [1,\ 100]$

3.7.2 Distracting useless task

An extra task can be implemented during the learning phase and it is indicated as n_5. The aim of this task is to *divert the attention* of the robot from the main tasks required by the user. It simply rotates the head of the robot moving an

extra revolute joint attached to the pan of the head. This task can be inserted in to the stack to prove that the algorithm deactivate tasks that are useless for the minimization of the cost function. When it is active, this task returns zero velocities for the useful joints of the robot and a random matrix, with the proper dimensions, as Jacobian. In this way the direct effect of the tasks velocities is null, but it prevents the null space projection from reaching the correct combination. The only parameter is a gain, which defines the velocity of the turning head:

$$v = \gamma \cdot 1\, rad/s,$$

where $\gamma \in [-1, 1]$.

3.8 Cost function (fitness measure)

In order to evaluate the performances of the stack of task with it's parameters, a global *cost function* $J(\theta)$ is introduced, where θ is the parameters vector of a stack. The main focus of this work is to make the robot able to autonomously minimize this fitness measure.
A cost function can be composed by multiple terms, which refers to different parameters of the simulation:

$$J(\theta) = \alpha_1 c_1 + \alpha_2 c_2 + ... + \alpha_m c_m, \tag{3.16}$$

where $c_1, c_2, ..., c_m$ are the costs referred to the single parameter and $\alpha_1, \alpha_2, ..., \alpha_m$ the respective weights, such that $\sum_{i=1}^{m} \alpha_i = 1$.
The cost function is an user choice (in terms of weights and which terms include in a simulation), and can be expressed in function of different parameters. This function will be truly significant in defining the parameters through the learning phase, and the result will heavily depend on it (chapter 5).

Precision

The *precision* cost can be expressed in three different types.
The first refers to the precision in terms of pose, and the error of the actual pose is used:

$$J(\theta) = ||error_{pose}||^2. \tag{3.17}$$

On the other hand, position and orientation can have different weight into the desired task, so that they can be divided in two different costs, as:

$$J(\theta) = ||error_{position}||^2 \tag{3.18}$$

$$J(\theta) = ||error_{orientation}||^2 \tag{3.19}$$

respectively for position and orientation. A lower precision implies an higher error, and then an higher cost. Clearly, precision is related to the *inverse kinematic* task (section 2.1.2) and, can be selected if the user wants to obtain an higher precision.

Distances from objects

The cost with respect to *distances* can be expressed in terms of pseudo-energy (2.35) for each active sensor ($r_k \geq d_k$):

$$J(\theta) = \frac{1}{2}\sum_{k=1}^{N}(r_k - d_k)^2 \tag{3.20}$$

where d_k and r_k are respectively the sensed distance and the rest length of the spring for the k^{th} sensor. The cost increases as the number of active sensors and decreases as their distances from the obstacles are bigger. This cost is clearly related to the *Obstacle Avoidance* task (section 2.1.3) and, if chosen by the user, tends to augment the distance of the robot from obstacles and then safety.

Manipulability

The cost expressed with respect to the *manipulability* of an arm is given by:

$$J(\theta) = \frac{1}{w} \tag{3.21}$$

where w (2.43) is the manipulability measure of the considered arm. Note that w is always positive. An higher manipulability implies a lower cost. This cost is clearly related to the *maximization of the manipulability measure* task (section 2.1.4), since it tends to increase it.

Distance from Mechanical Joint Limits

The cost expressed with respect to the *distance from mechanical joint limits* of an arm is given by:

$$J(\theta) = w^2. \tag{3.22}$$

Since $w \leq 0$ (2.46), a lower value of it implies an higher cost. This cost is clearly related to the *maximization of the Distances from Mechanical Joint Limits* task (section 2.1.5).

Time

The *time* cost is given as:

$$J(\theta, t) = t^2 \tag{3.23}$$

where t is the actual time of simulation. Clearly, an higher time implies an higher cost. If this is chosen by the user, the robot tends to perform its tasks faster.

3.9 Genetic Programming Pipeline

In this section, the pipeline adopted for the *Genetic Programming* process is outlined, once the simulation settings are decided.
First of all, the original generation is initialized. This is made by a number of elements n_{pop}, decided a priori by the user. The number of elements of each generation will be the same of the first one. In the initialization, the stack can be formed by a certain order of the four tasks, or it can be randomized for each element. All the parameters are randomly initialized for each stack in the proper ranges of values or can be fixed at a certain value.
After the initialization of a base population, all the iterations (n_{it}, which is an user choice) follow the same structure:

- All the stacks that have not an assigned a cost yet (namely the cost is still -1) are runned in a selected environment and, the final cost is assigned to each one. Then they are printed in ascending order of cost.

- The stack are randomly paired, the one of the two with the lowest cost is copied in to the new generation (*reproduction* 2.2.2). The other one will be permanently lost.

- For each survivor, a random genetic operation is performed (*crossover* 2.2.2 or *mutation* 2.2.2), and the offspring is added in the new population. After that the cycle restarts.

Initialization

At the beginning of each simulation, a random population is created selecting the active tasks and the number of individuals. The priority order of tasks in the stack can be changed randomly or, once the best order for the combination is found, it can be decided a priori. The tasks order is represented as an array, in which each element is a label which refers to a single task and, the priority is decreasing from first element to the last one. In the beginning phase of the research, some task may also not be included into the stack (which is different from being deactivated, in this case it will never be never present) so the stack can be shorter.
Each stack is represented as a list in the code and, the initialization follows the steps:

- Access the tasks array and recognize the desired task to be solved, then for each element in that array:

  1. Create an empty list, and append the label of the task in first position
  2. The activation of the task in the specific stack is decided randomly with a certain probability. In this specific case the probability to be active is $p[active] = 0.8$. Then, if the task is active in second place at the list will be appended *True*, *False* instead.
  3. Lastly, the list of the parameters is randomly created choosing random elements in a pre-defined range 3.7.1. If parameters are fixed, the list is created with the decided elements. Then this list will be appended in third position.

- When all the task are added to the stack list, the cost -1 is inserted in first place.

All the previous steps are repeated m times, where m is the number of stacks in the first population. Once the initial population is created, all the stacks are simulated with an user-defined cost function in a chosen environment.

Iteration

When all the new stacks have being computed, and the cost functions are obtained, they are randomly paired each other and the one with lowest cost survives and it is copied into the new population (**reproduction**). The other stack will be lost forever. If the number of element in the population is odd, one element is randomly selected and copied into the new population before the creation of the pairs, letting an even number of stacks in the old population.
For each survived element, a genetic operation between **crossover** and **mutation** is selected and applied in order to obtain a new offspring. In this way the number of elements in the population is always the same at each generation.

Crossover

Starting from an element which survived at the genetic competition, a new offspring is generated selecting a second stack randomly between the other survivors. The offspring stack inherit the parameters of each task from one parent which is randomly selected between the two. A counter avoids the duplication of a program which may happen if all the task acquire the parameters from the same parent. The offspring can inherit a maximum of $k-1$ times from a parent, where k is the number of tasks in each stack. In this work, the first parent have an higher priority of being selected to give parameters to offspring. So a new list is created and one by one the selected elements are appended in it.
In the first exploration cases, when the best priority order of the stacks is not already defined, this can be inherit from a parent randomly selected or randomly created from scratch.
The cost -1 is added in first position of the stack. After that, the new element is introduced in the new population.

Mutation

Starting from an element which survived at the genetic competition, a new offspring is generated changing a parameter of a task which is randomly selected with equal probabilities in the stack. The parameter can be both the presence marker (*True* or *False*), or a parameter that control the execution of the task with equal probability. All the other elements of the program are simply copied from the parent. So a new list is created and one by one the selected elements are appended in it. In the first exploration phase, while deriving the best order

for the stack, a mutation can also change the priority order of the tasks.
The cost -1 is added in first position of the stack. After that, the new element is introduced in the new population.

3.10 Graphical User Interface (G.U.I.)

A *Graphical User Interface (G.U.I.)*, was introduced to let an user decide the settings of a simulation and try the developed code. At the beginning the user can chose from 4 different types of functionalities: learning the best stack of tasks for a specific cost function, try a stack choosing an environment with dynamics or with kinematics only or, lastly, completing a duty specifying the stack of tasks in a full dynamics environment.

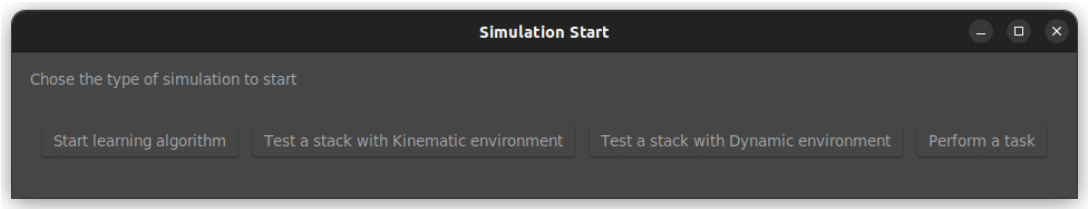

Figure 3.18: Start of the simulation.

Learning

In this phase it is possible for the user to *train* the robot in a specific environment and with a specified cost function.

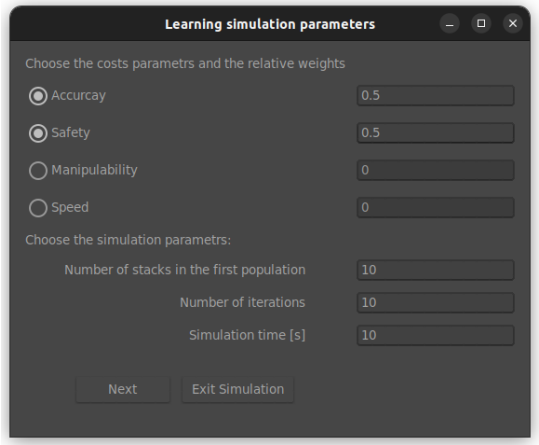

Figure 3.19: 1^{st} learning parameters window.

First of all, the user can decide the cost function that has to be minimized. The number of possible choices was reduced, taking into account:

- Accuracy: as did during the learning phase of this work this cost introduce in the weighted combination the *error of the pose* of the kinematic chain with respect to the desired objective (3.8).
- Safety: this is an equivalent of the previously defined *distances from objects* (3.8), but in this case is defined as the safety that the user wants to have while working with the robot.
- Manipulability: the same of the *manipulability* used in this work (3.8).
- Speed: meaning speed of execution, it represents the *time* cost (3.8).

Note that the user can select the weight of the specific cost item only once the check button is selected. The weights are normalized before they are used in the simulation, in order to maintain a convex combination of the chosen elements. After that, the user can specify the number of stacks in the original population (which will be the same for all the subsequent generations), the number of iterations (namely, the number of generations) and the simulation time to complete a duty. All this values are constrained to be positive and in the case the user insert a float number only the integer part will be considered, except for the time. Once the selection process is completed from the user, the initialization of the simulation continues.

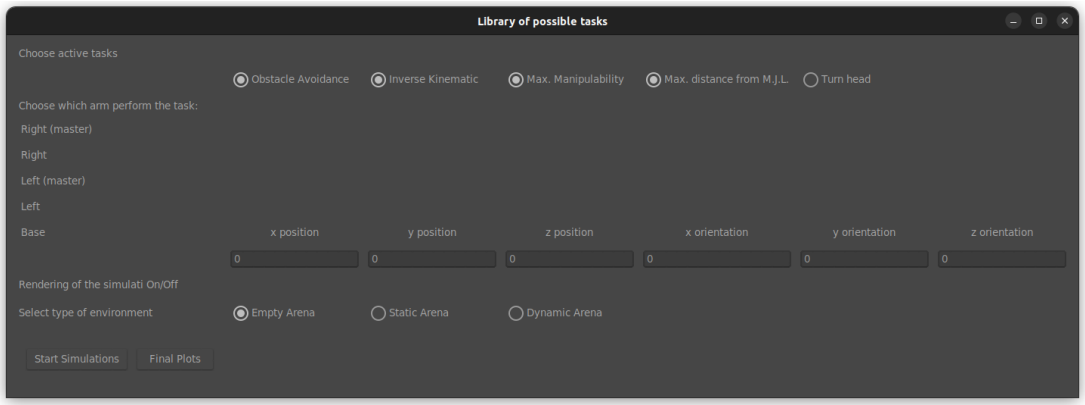

Figure 3.20: 2^{nd} learning parameters window.

In this second window, the user can select the other learning parameters. First of all, it is possible to chose the task that are present or not in the stack. If the check button is active, they are present. Note that tasks can be present but deactivated, if they are not present the stack will never include them for this simulation. Then, the user can decide with buttons which arm, or eventually the base, solves that

stack and if it is master or slave (section 3.3). Only one arm in one configuration can be chosen for a simulation, so their activations are mutually exclusive. Then, the desired pose can be chosen only if the *inverse kinematic* task is present in the stack. If the base is selected, since the pose is expressed as $\boldsymbol{p} = [p_x,\ p_y,\ \theta_z]$, only this three parameters can be inserted by the user. The remaining entries (namely $[p_z,\ \theta_x,\ \theta_y]$) are blocked. It is possible to activate or deactivate rendering with a button, however during learning phase the activation is not suggested due to slower computational times. Then, the environment (section 3.1) and the type of output can be decided. As output it is possible to simply obtain the results of each run (cost function) of a stack, suggested during learning, or obtain all the plots of the measures of interest. If the option with plots is selected, at the end of each run the algorithm stops and wait for the user to close the plots after the visualization. This prevent to execute the following run while plots are not closed yet.

Test

As it is possible to see in figure 3.18, the user can *test* a stack in a full dynamics environment, or in one with kinematics only. In both cases, the selection window is the same, which is reported in figure 3.21.

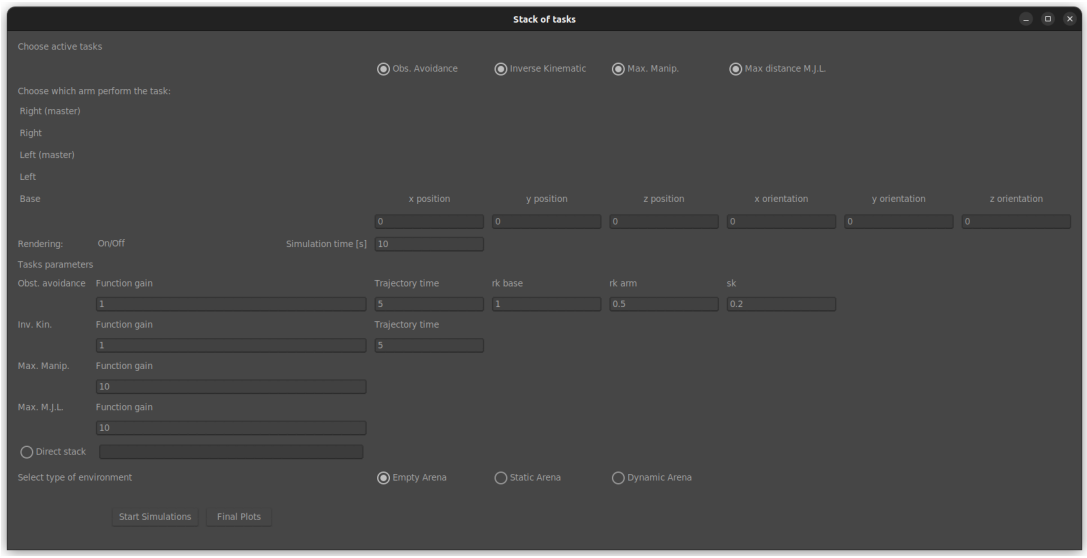

Figure 3.21: Test parameters window.

As for the learning phase, the user can decide the environment, the simulation time, the kinematic chain, the target pose and if the rendering is showed.
In this case, the stack is only one and it is inserted by the user. In fact, the aim

of this section of the simulation is to test a stack with different environment, dynamics and obtain plots of the performances since they are not typically obtained during training. The stack can be manually inserted in two ways:

- 1^{st}: selecting the parameters in the proper entries. If the check button of a task is turned of, its parameters can not be modified.
- 2^{nd}: directly inserting the task as a list in the proper entry. Since during learning phase the stacks are printed in terminal as lists, this is useful in order to copy and paste them to test the performances. If this option is selected, parameters can not be modified as in the 1^{st} case and vice versa.

Then, the stack can be tested with plots at the end of the run. In this case, the initial configuration of the robot can be modified accessing the *.py* files of the specific environments. Otherwise, the robot will always start with $\boldsymbol{q}_{base} = [0, 0, 0]$ and default arms configuration $\boldsymbol{q}_{arm} = [\pm 0.403, -0.636, \pm 0.114, 1.432, \pm 0.735, 1.205, \mp 0.269]$ for right and left arm.

Complete a Duty (pick&place)

Finally, it is possible to perform a *pick and place* task in a predefined environment (section 3.1.4).

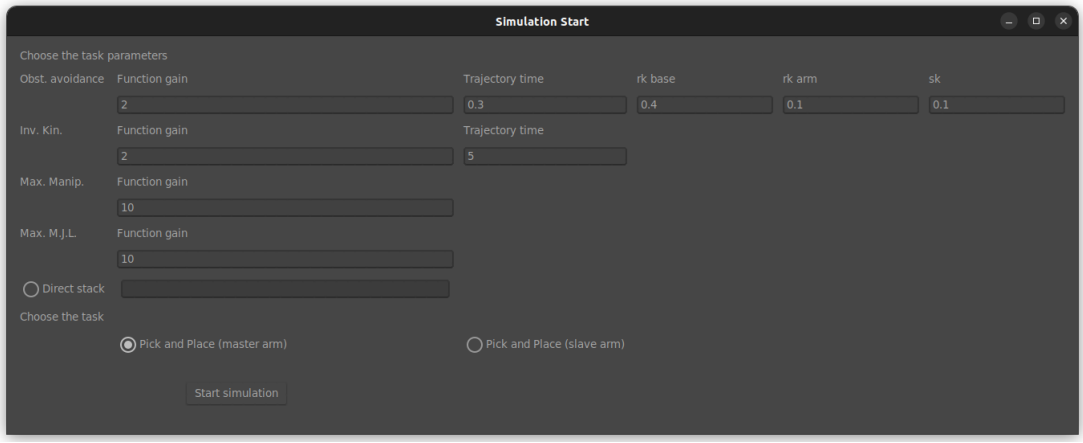

Figure 3.22: Pick & place parameters window.

As seen in section 3.10, it is possible to insert a stack of tasks. However in this case all the tasks have to be included in the stack. Before starting, it is possible to chose if one of the two arm is the master, or both slaves and the base moves alone with its own target pose.

Chapter 4

Single task resolution and tasks combination: simulations and results

4.1 Single task resolution

In this section the results obtained in the solution of single tasks are reported with plots. In this chapter, the parameters are selected in order to show the performances of the robot and, they could not be the best possible parameters for that task in a specific situation.

4.1.1 Inverse Kinematic

In this section the results obtained with the *Inverse Kinematic* (2.1.2) are showed for the mobile base, for the master arm and for the slave arm. In all the cases the result will be shown both with a full dynamics environment and in the case in which only kinematic is present in the simulation. In the case of the arm, the results reported are obtained with the right arm, left arm would have given identical results.
All the poses refer to the pose of the end-effector attached at the end of the kinematic chain and, no obstacles are present during this simulations.

Mobile Base

First, the result for the movement of the mobile base are reported with chosen gain of $\gamma = 1$. Trajectories are not implemented for the base. So that, the

computation of the velocities is not affected by derivatives of the desired pose but, only by the error of the pose $\boldsymbol{e}_{pose}$, namely the difference between desired pose and actual pose. In fact, in equation (2.20), $\dot{\boldsymbol{p}}_d = 0$ and $\boldsymbol{p}_d$ is not given by a trajectory path but it is always fixed at the desired final pose.
The initial pose of the base. for this simulation, is

$$\boldsymbol{p} = [p_x,\ p_y,\ \theta_z] = [0,\ 0,\ 0]$$

which is given by the initial configuration of the robot's joints

$$\boldsymbol{q} = [q_{1b},\ q_{2b},\ q_{3b}] = [0,\ 0,\ 0].$$

The target pose $\boldsymbol{p}_d = [p_{x_d},\ p_{y_d},\ \theta_{z_d}]$ is given by

$$\boldsymbol{p}_d = [1,\ -1,\ \frac{\pi}{2}].$$

Subsequently, the results with an only kinematics environment are reported for a simulation of 20 s. The simulation is not stopped once the objective is reached with a certain accuracy, or equivalently with a small error, to show the stationary behaviour of the robot once the task is completed.

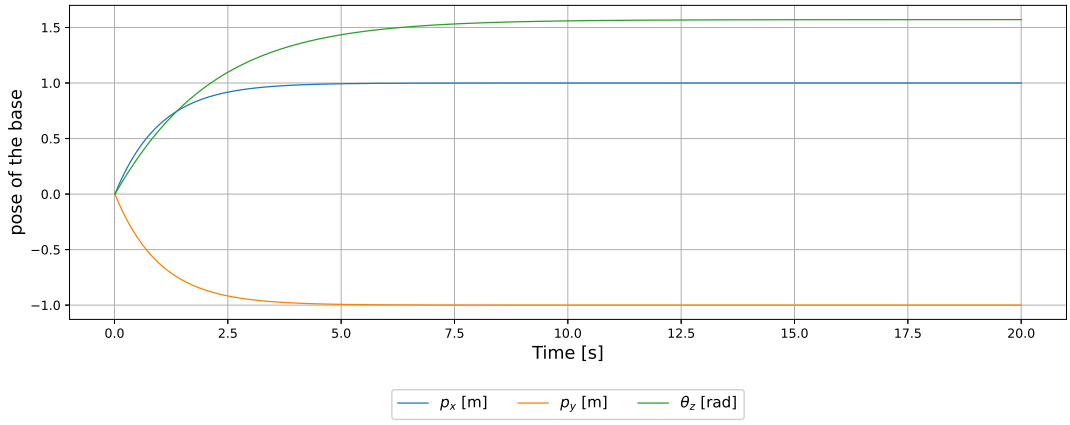

Figure 4.1: Pose of the base.

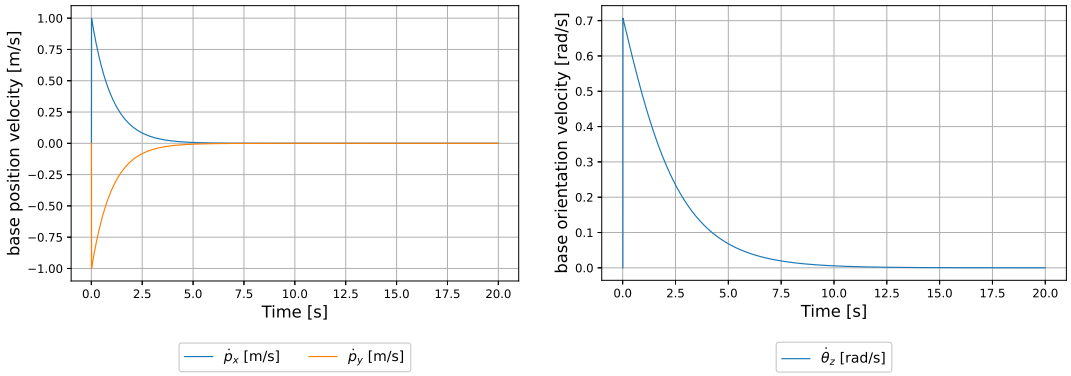

Figure 4.2: Base linear velocities (left) and base angular velocity (right).

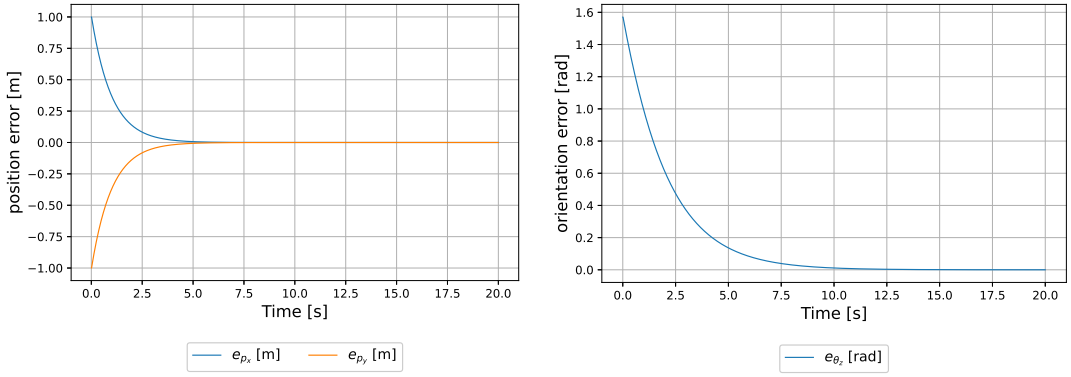

Figure 4.3: Base position errors (left) and base orientation error (right).

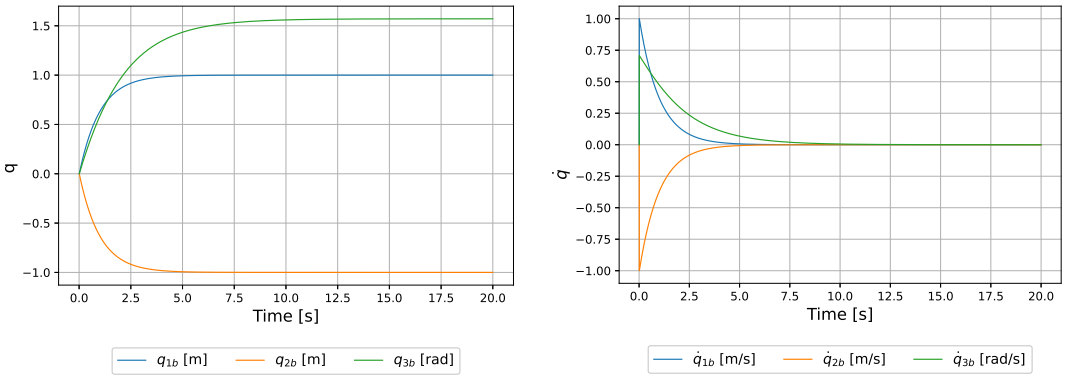

Figure 4.4: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot works as expected. It reaches the final desired pose with its configuration. Note that the velocities go to zero as the error goes to zero. Since the movement of the base is modelled by two prismatic joints and one revolute, the values of the pose in Cartesian space are the same of the configuration in joint's

space.
It is possible to notice also that, in this case, the convergence in position is faster with respect to the convergence in orientation.
Subsequently, the results obtained for the mobile base in a full dynamics environment are reported. It is emphasized that the results are dependent from the choices reported in table 3.1, during the implementation of the mobile base.

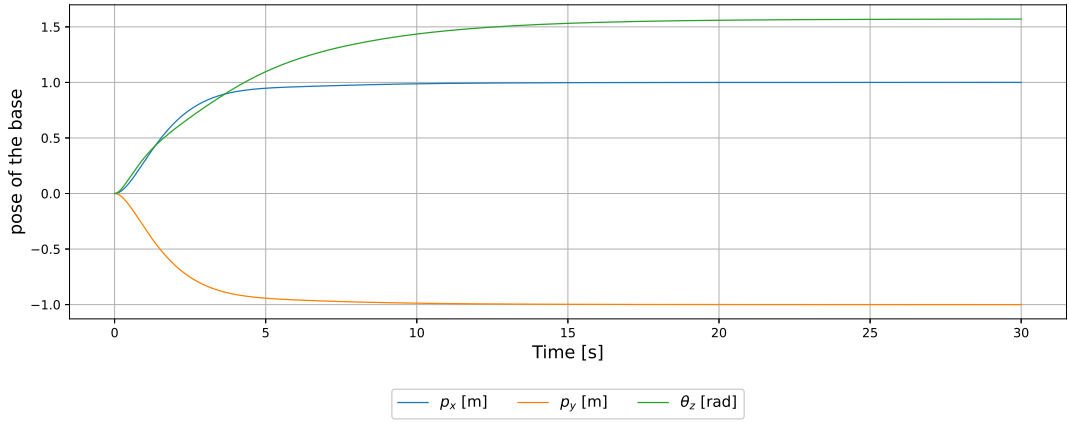

Figure 4.5: Pose of the base.

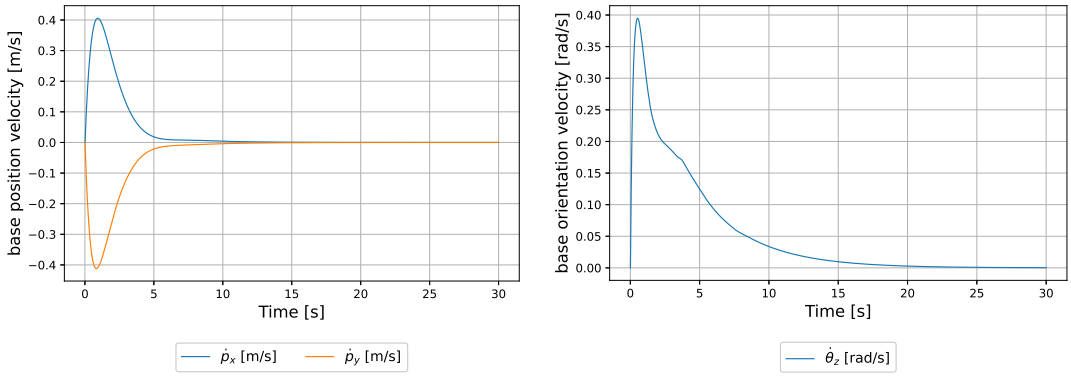

Figure 4.6: Base linear velocities (left) and base angular velocity (right).

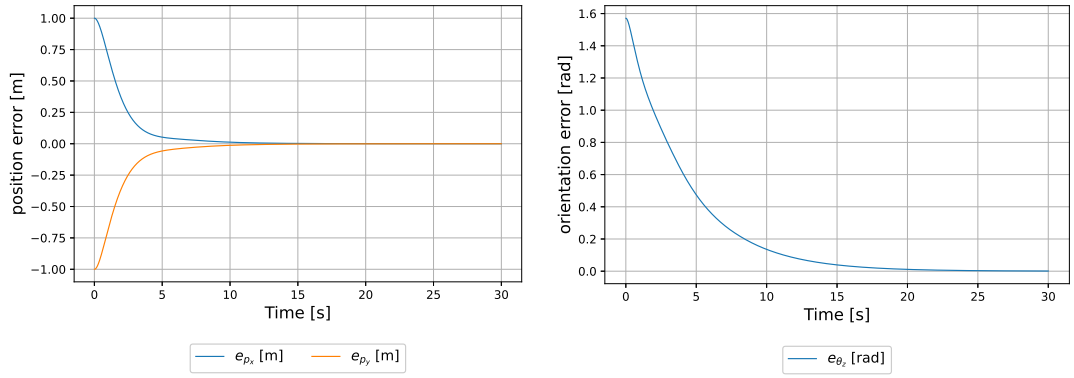

Figure 4.7: Base position errors (left) and base orientation error (right).

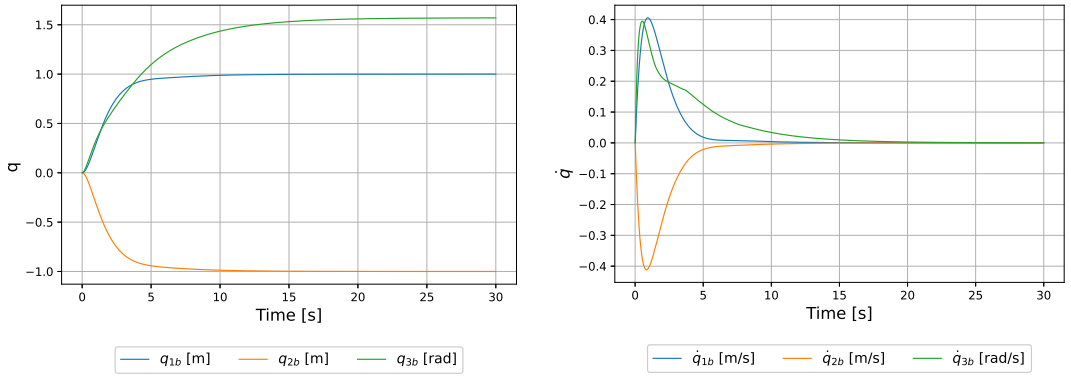

Figure 4.8: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

It is possible to observe how the convergence in the full dynamic case is slower than the case with kinematics only. Note also that in the second case velocities do not grow instantaneously as in the first one, due to inertia and frictions. Anyhow, the final pose is successfully reached by the robot base, also in the case of a fully dynamics environment, accomplishing the task.

Master Arm (right)

In this section, the results obtained with the master arm case are reported. The chosen arm (right) can control the movement of the base, obtaining an high redundant kinematic chain with 10 degrees of freedom. The chosen gain is $\gamma = 1$ and the trajectory time is $t = 5\ s$, meaning that after 5 seconds the robot should already be in the desired pose and reach it following a desired path during this time. Once this time is expired the trajectory is and, the arm simply points toward the configuration which satisfy the inverse kinematic for the end-effector

pose with $\dot{\boldsymbol{q}} = 0$.
The end-effector starts from the initial pose $\boldsymbol{p} = [p_x,\ p_y,\ p_z,\ \theta_x,\ \theta_y,\ \theta_z]$:

$$\boldsymbol{p} = [0.691,\ -0.784,\ 1.059,\ 0.707,\ -2.681,\ -0.630],$$

given by the initial robot's joints configuration $\boldsymbol{q} = [q_{1b}, q_{2b}, q_{3b}, q_1, q_2, q_3, q_4, q_5, q_6, q_7]$:

$$\boldsymbol{q} = [0,\ 0,\ 0,\ 0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269].$$

Then, it has to reach the target pose:

$$\boldsymbol{p}_d = [1.7,\ -0.7,\ 1.3,\ 0,\ \frac{2}{3}\pi,\ 0].$$

Note that the pose of the end-effector of the master arm, is expressed with respect to the pose of the mobile base at instant of time $t = 0\ s$. The initial value of the base joints are always zero, since the initial pose of the base is used to define the center and the orientation of the reference system for the whole simulation (section 3.1).
Following, the results obtained in an environment in which only kinematic is taken into account, are reported to show the behaviour of the system made by arm + mobile base.

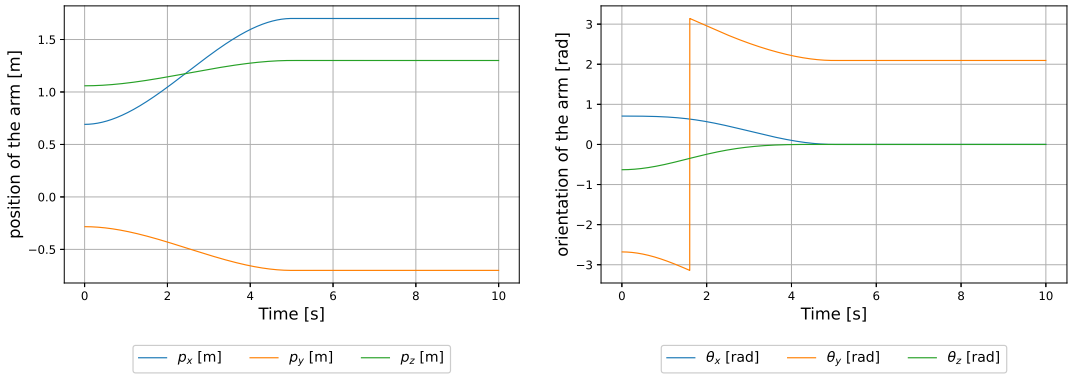

Figure 4.9: End-effector position (left) and orientation (right).

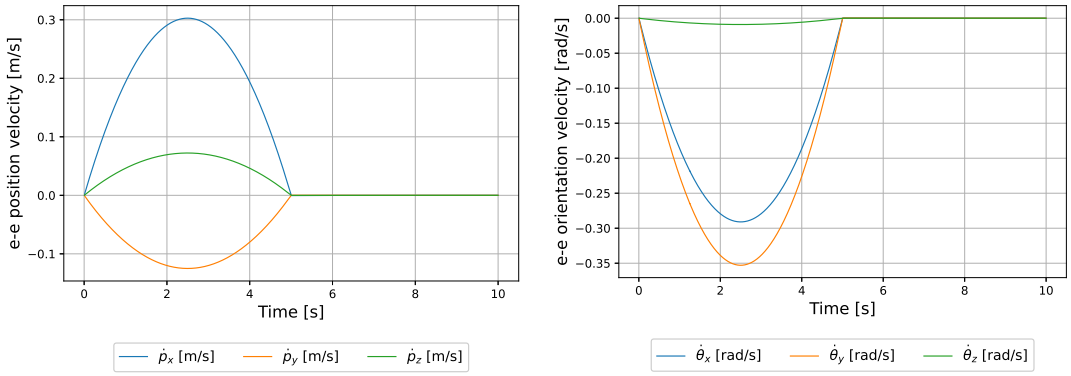

Figure 4.10: End-effector linear (left) and angular (right) velocities.

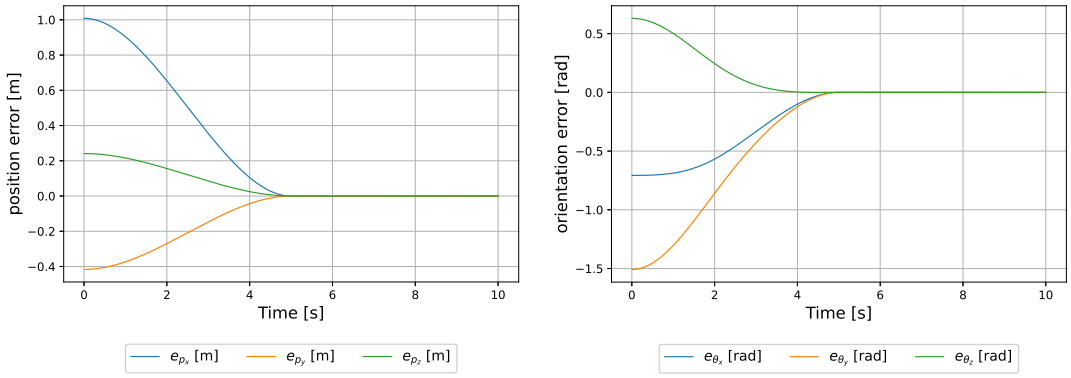

Figure 4.11: Position (left) and orientation (right) errors.

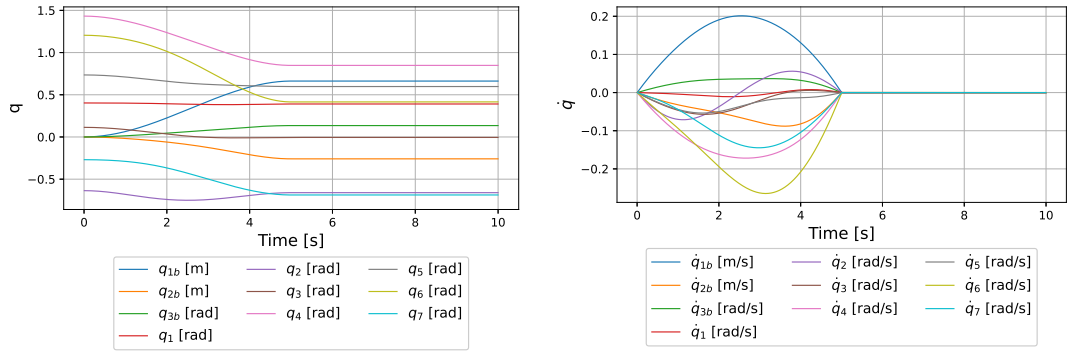

Figure 4.12: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

At the end the robot's arm is in configuration:

$$\boldsymbol{q} = [0.662,\ -0.259,\ 0.135,\ 0.391,\ -0.660,\ -0.005,\ 0.848,\ 0.598,\ 0.416,\ -0.687].$$

The final target is reached, as it is possible to see from the plots, in the desired time. After that, the robot stays in the same position, maintaining the target pose. Note how the introduction of a trajectory helps to avoid instantaneous velocities $\dot{\boldsymbol{q}}$ in the beginning phase of the task, even if dynamics are not present in the simulation.
Following, the results obtained in a full dynamics environment are reported.

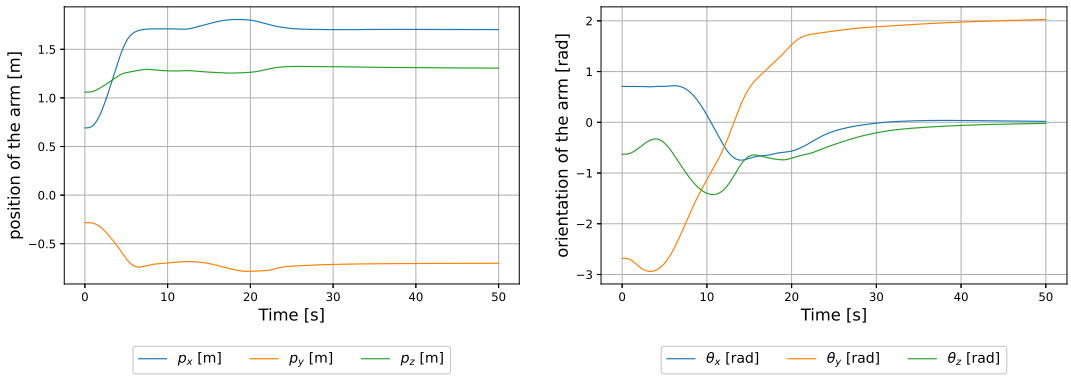

Figure 4.13: End-effector position (left) and orientation (right).

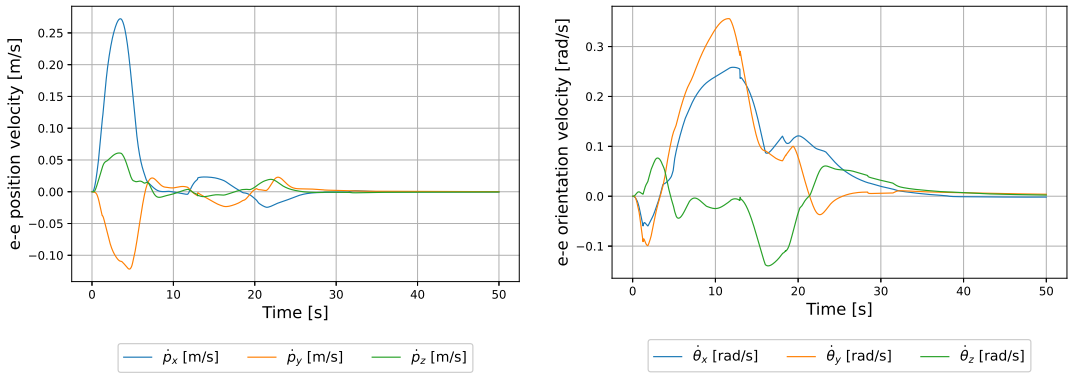

Figure 4.14: End-effector linear (left) and angular (right) velocities.

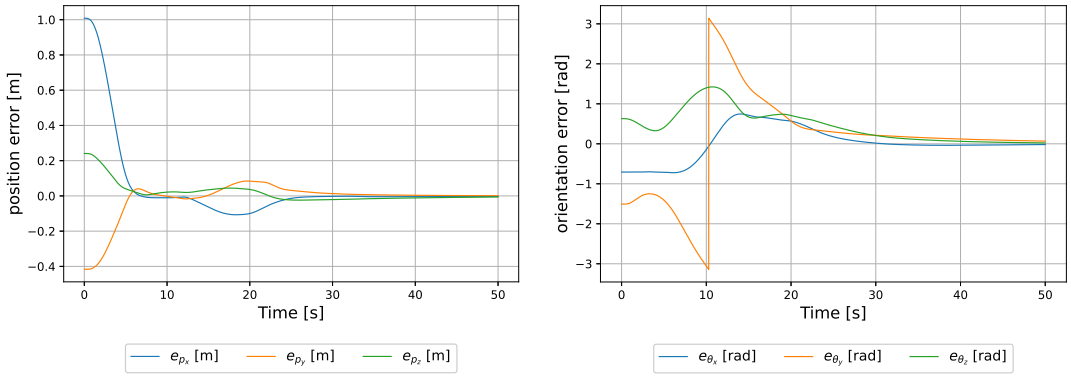

Figure 4.15: Position (left) and orientation (right) errors.

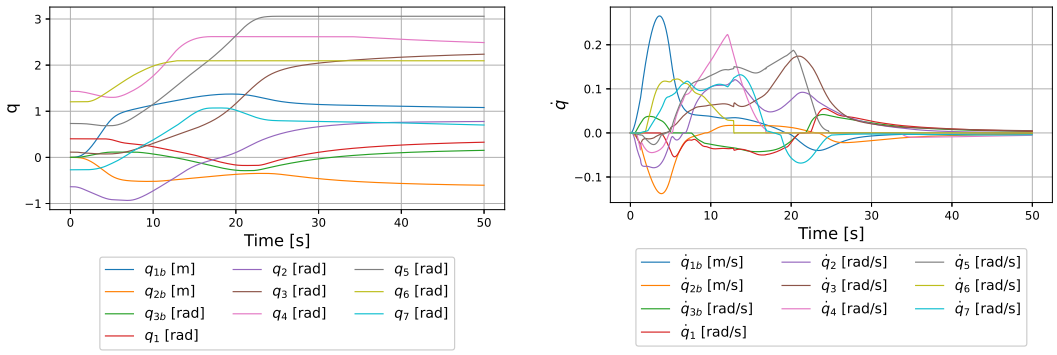

Figure 4.16: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

At the end the robot's arm reaches the configuration.

$$\boldsymbol{q} = [1.078,\ -0.611,\ 0.157,\ 0.335,\ 0.769,\ 2.232,\ 2.475,\ 3.059,\ 2.094,\ 0.711].$$

Note how, the final configuration is different from the one obtained with an only kinematics environment, despite both accomplish the same task. This highlights how the solution may also depend from the environment and the condition of the robot.

As it is possible to see from plots, the convergence is slower in the environment with dynamics. In particular for the orientation of the end-effector. Then, with dynamics, the robot is not able to correctly follow the desired trajectory in the given time. So that, after 5 seconds the inverse kinematic error is not close to zero yet and the convergence is reached after the trajectory time is expired.

Increasing the trajectory time, the behaviour of the arm is simply delayed and, it does not help the end-effector to reach the desired pose in the specified time.

Also an higher gain, however not bigger than 2, would not help in following the trajectory in an environment with full dynamics. So, for the master arm, is impossible to follow a desired trajectory depicted in the Cartesian space, with it's end-effector in an environment with full dynamics, if the trajectory is simply given by the desired velocities in the configuration space for the inverse kinematic problem.

Slave Arm (right)

In this section, the result for the slave arm are reported. So, in this case, the kinematic chain is composed only by the joints of the arm, as for the original robot. The system, with 7 degrees of freedom, is still redundant. The base is fixed and can not move. The chosen gain is $\gamma = 1$ and the trajectory time is $t = 5$ s. The arm starts from the initial pose $\boldsymbol{p} = [p_x,\ p_y,\ p_z,\ \theta_x,\ \theta_y,\ \theta_z]$:

$$\boldsymbol{p} = [0.691,\ -0.284,\ 0.135,\ 0.707,\ -2.681,\ -0.630],$$

given by the initial robot configuration $\boldsymbol{q} = [q_1,\ q_2,\ q_3,\ q_4,\ q_5,\ q_6,\ q_7]$:

$$\boldsymbol{q} = [0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269]$$

Then, it have to reach the target pose

$$\boldsymbol{p}_d = [0.7,\ -0.855,\ 0.4,\ 0,\ \frac{2}{3}\pi,\ 0].$$

Note that the pose of the end-effector is expressed with respect to the pose of the base frame in position $[0,\ 0,\ 0.924]$ m with respect to the mobile base frame, and with the same orientation.
Following, the results obtained in an environment in which only kinematic is taken into account are reported.

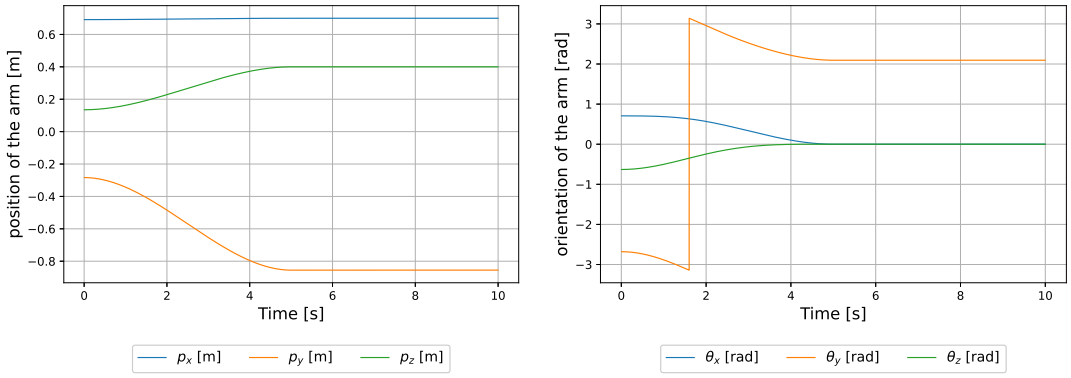

Figure 4.17: End-effector position (left) and orientation (right).

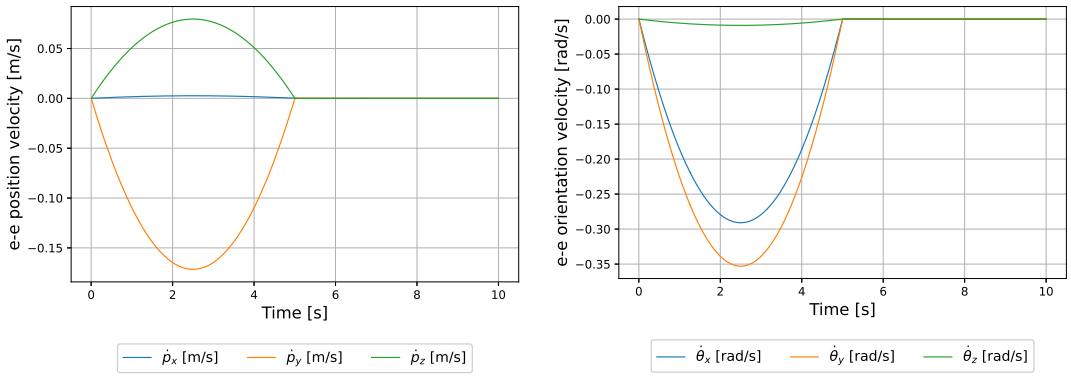

Figure 4.18: End-effector linear (left) and angular (right) velocities.

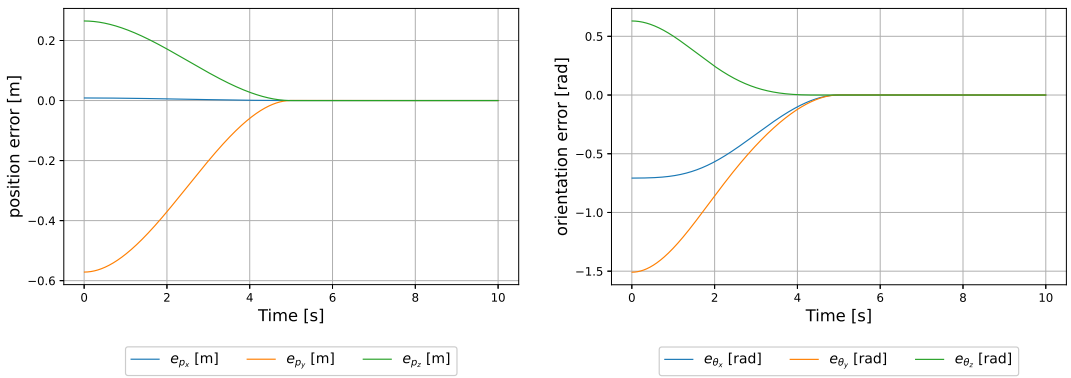

Figure 4.19: Position (left) and orientation (right) errors.

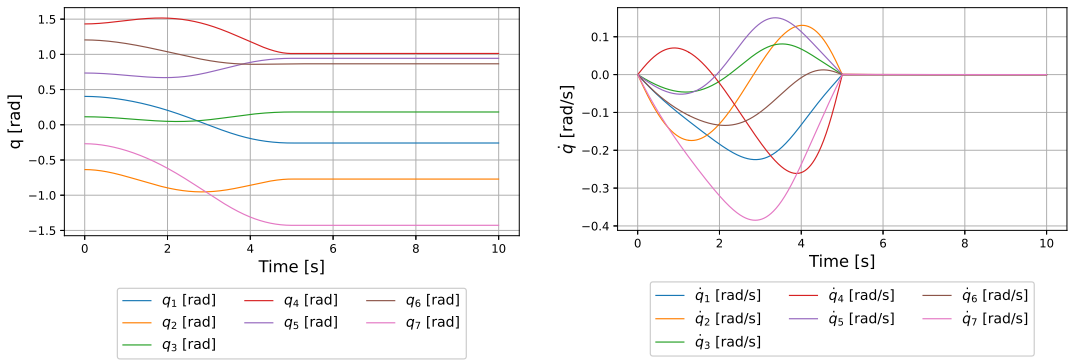

Figure 4.20: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The desired pose for the end-effector is reached with the following joints' configuration in the Cartesian space:

$$\boldsymbol{q} = [-0.258,\ -0.771,\ 0.182,\ 1.013,\ 0.944,\ 0.866,\ -1.427].$$

So that, the task is successfully completed.

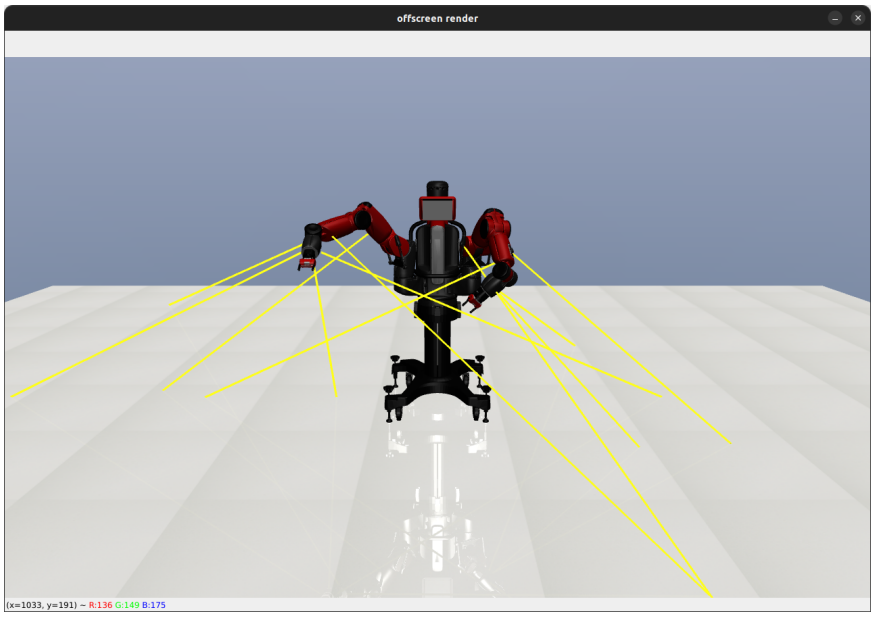

Figure 4.21: Baxter robot with right arm in the desired pose. The left one is still in the initial configuration.

Then, results for the case with full dynamics are reported.

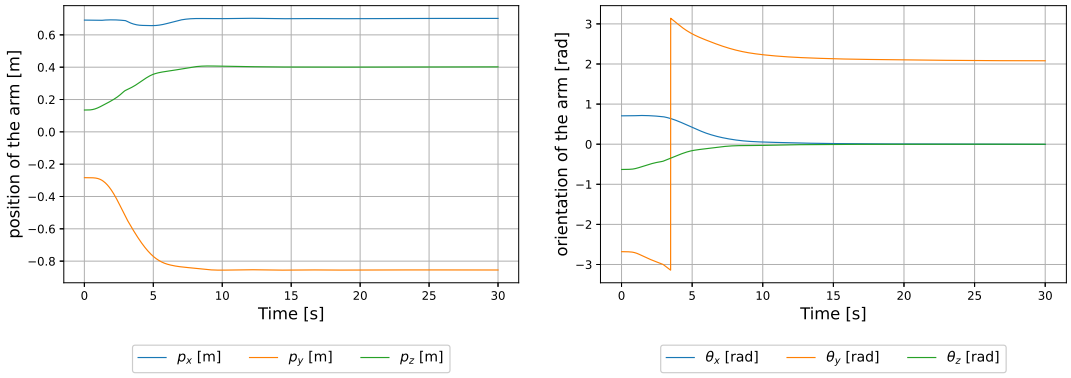

Figure 4.22: End-effector position (left) and orientation (right).

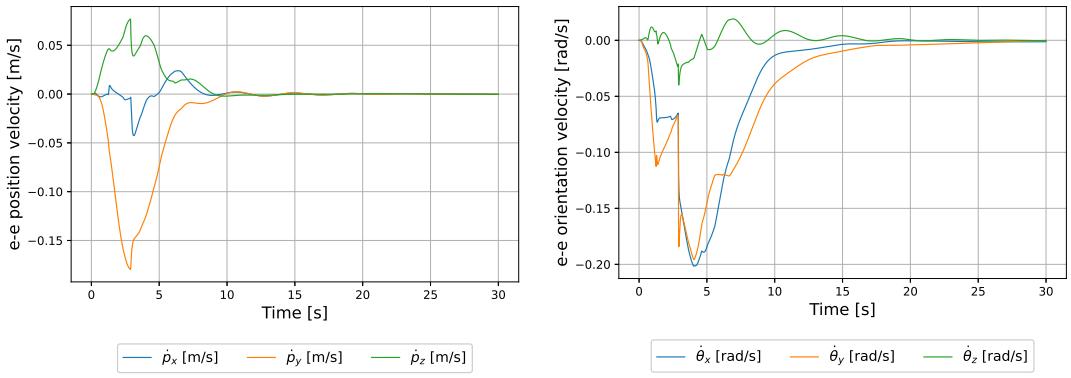

Figure 4.23: End-effector linear (left) and angular (right) velocities.

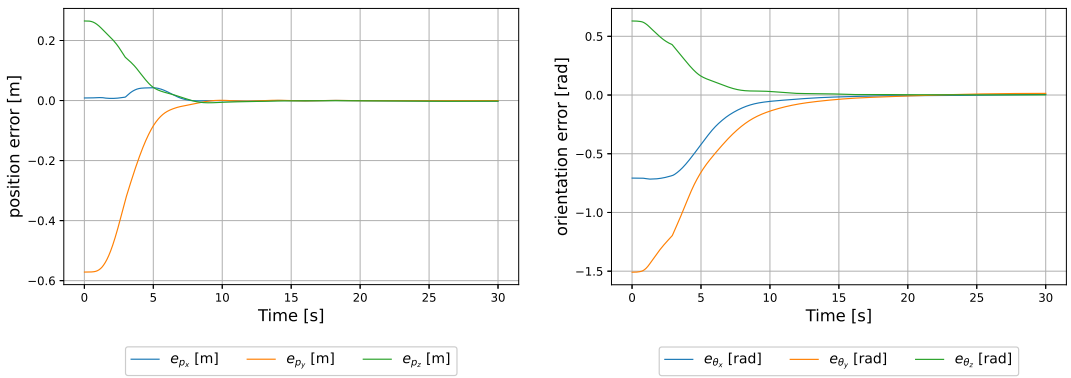

Figure 4.24: Position (left) and orientation (right) errors.

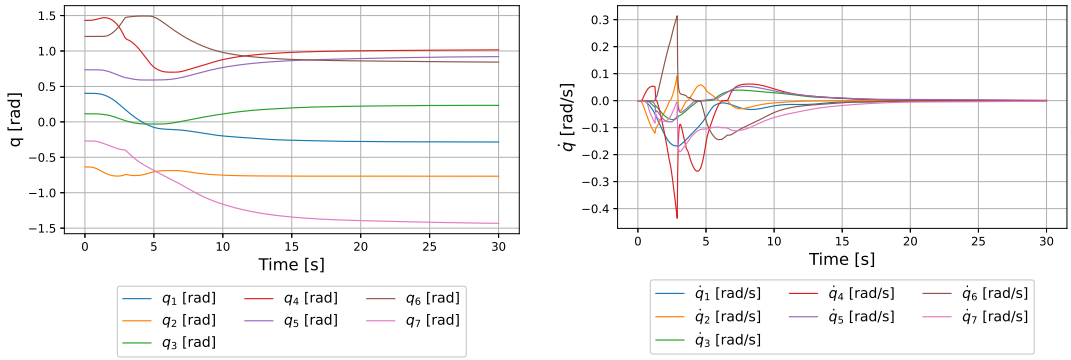

Figure 4.25: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The desired pose for the end-effector is reached with the following joints configuration in the Cartesian space:

$$\boldsymbol{q} = [-0.284,\ -0.767,\ 0.233,\ 1.016,\ 0.920,\ 0.843,\ -1.432].$$

As in the master arm case, the robot is not able to reach the desired pose within the trajectory time and, the configurations of the robot are not the same.

4.1.2 Obstacle Avoidance

In this section, the results obtained for the obstacle avoidance task are reported. The case of the mobile base, master arm and slave arm (only right arm) where taken into account with a full dynamic environment and in one with kinematics only. In the last part, results with dynamics obstacles are reported.

Mobile base

As explained in section 3.4, the base owns 8 sensors placed at 45° of difference. When an obstacle is sensed and the distance is smaller than the chosen value of $r_{k_{base}}$, the sensor is activated and the robot moves away from the obstacle and the Jacobian matrix is $\boldsymbol{J}_o \neq \mathbf{o}_{6\times n}$. The gain for that task is chosen as $\gamma_o = 1$, the trajectory time as $t = 5\ s$ and the minimum distance of activation of the sensors is $r_{k_{base}} = 1\ m$. For a first case, the environment is composed by an empty room, where a wall is close to the robot back at 0.79 m from the sensor placed at 270°, i.e. at 1 m from the pedestal reference frame center. The robot's base starts in pose

$$\boldsymbol{p} = [p_x,\ p_y,\ \theta_z] = [0,\ 0,\ 0] = \boldsymbol{q}.$$

First, the results obtained in the case with kinematics only are reported.

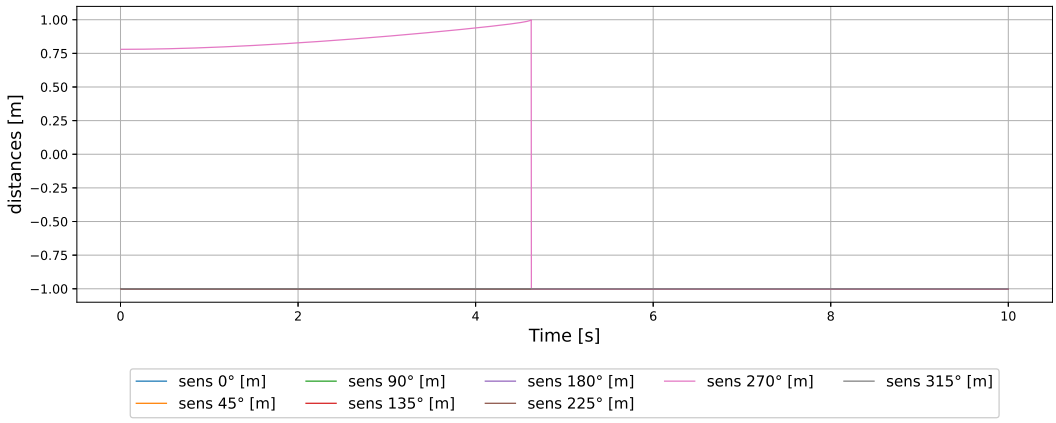

Figure 4.26: Distances of the base sensors.

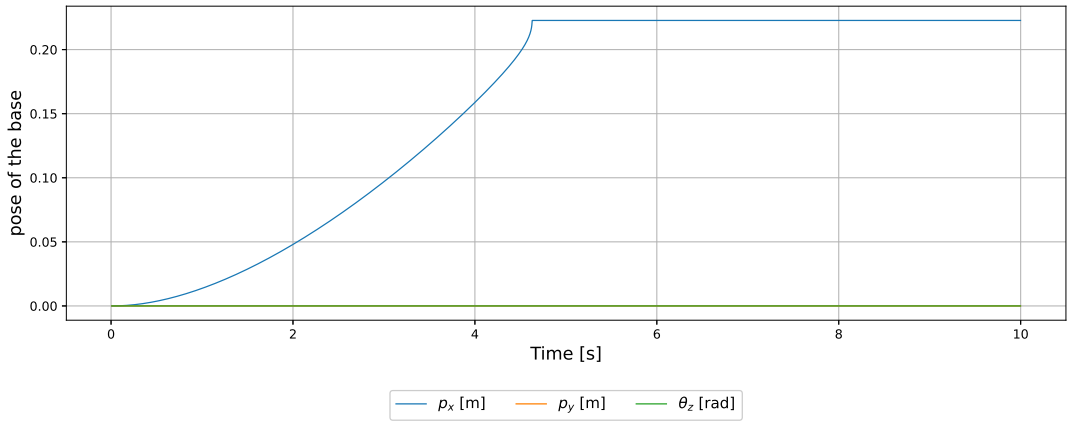

Figure 4.27: Pose of the base.

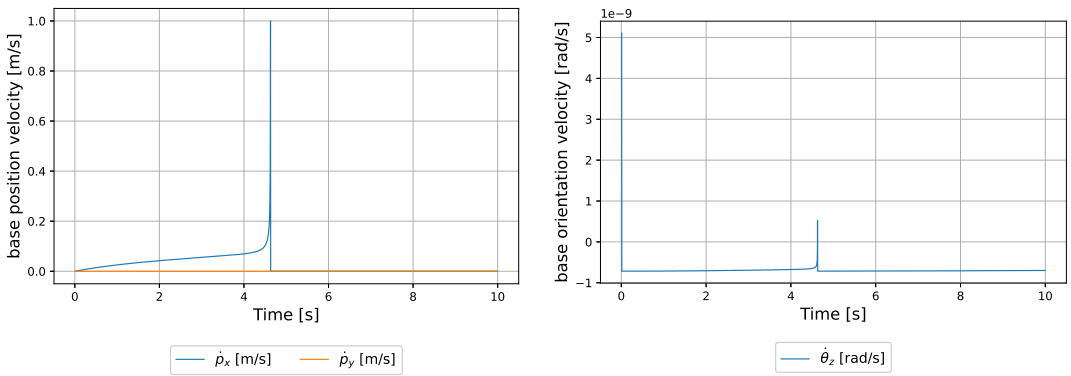

Figure 4.28: Base linear velocities (left) and base angular velocity (right).

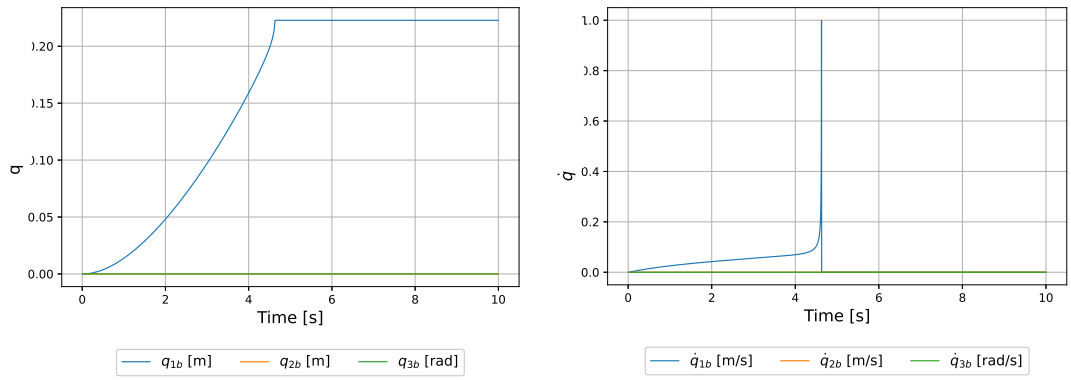

Figure 4.29: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

As it is possible to observe from the plots, the final pose of the base coincides with the final configuration in joints space, i.e.

$$\boldsymbol{p} = [2.228 \cdot 10^{-1},\ -2.454 \cdot 10^{-9} \simeq 0,\ -6.979 \cdot 10^{-9} \simeq 0] = \boldsymbol{q}.$$

As it is possible to observe from plot in figure 4.26, the robot accomplish its task moving away the base from the obstacle. Note that, since the obstacle is positioned behind the robot, it moves forward to maintain a distance of $r_{k_{base}} = 1$ m in the specified trajectory time with the activated sensor. Once the obstacle is far enough, the robot stops.
After that, the same situation was tested with a full dynamics environment.

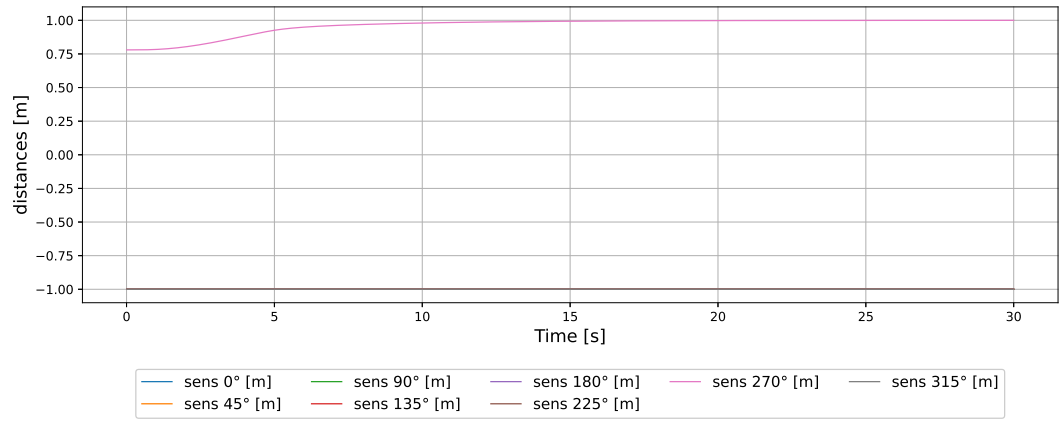

Figure 4.30: Distances of the base sensors.

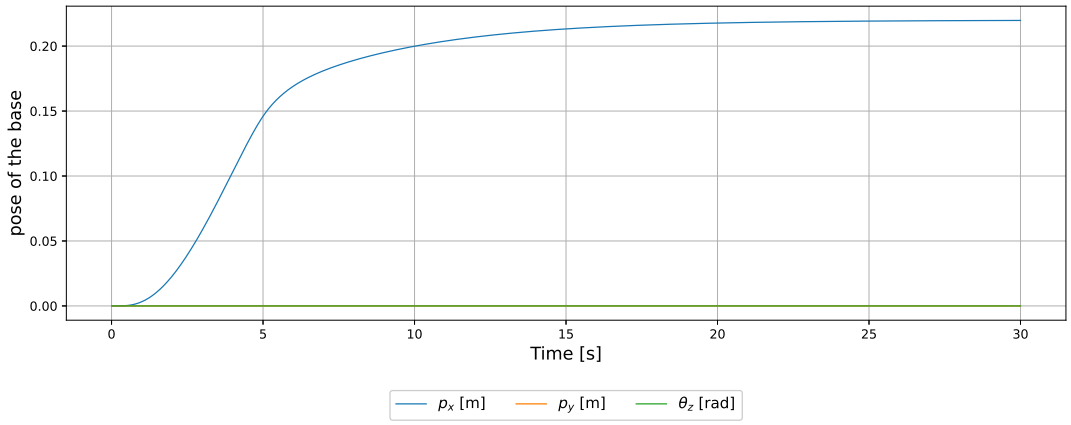

Figure 4.31: Pose of the base.

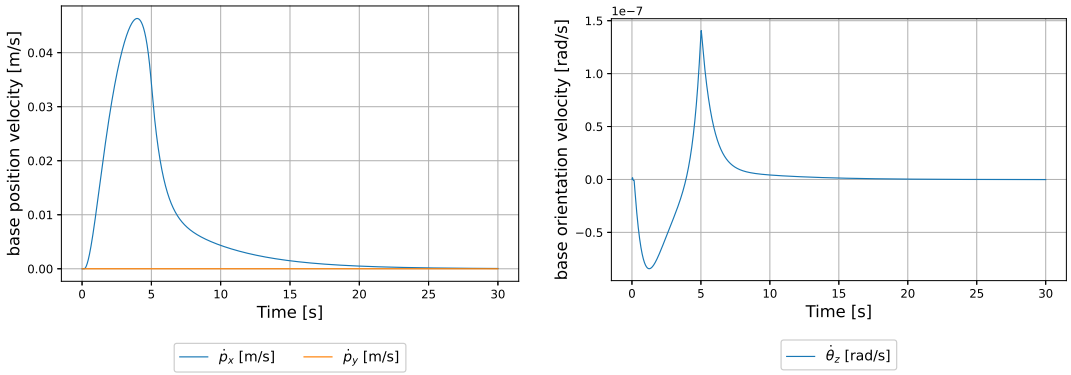

Figure 4.32: Base linear velocities (left) and base angular velocity (right).

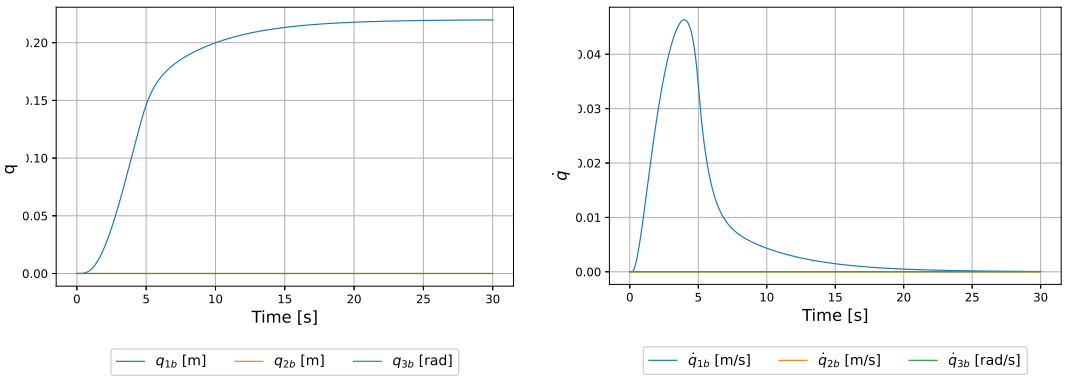

Figure 4.33: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot moves away from the obstacle also in an environment with full dynamics. However, as in the inverse kinematic case, the trajectory can not be completed within the specified time also in this last case. With the introduction

of dynamics it is possible to observe from plot in figure 4.30 how the distance of the robot's sensor from the obstacle remains close to 1. This happens due to the fact that the force applied if $d_k \simeq r_{k_{base}}$ is not enough to overcome frictions. However, the robot can be considered far enough from the obstacle to avoid any collision, then the task is completed with the same pose $\boldsymbol{p}$, and then same configuration $\boldsymbol{q}$, of the only kinematic case.
After that, a second obstacle was placed in the room. A second wall, on the left of the robot, is placed at the same distance of the first one from the sensor oriented at 180°.

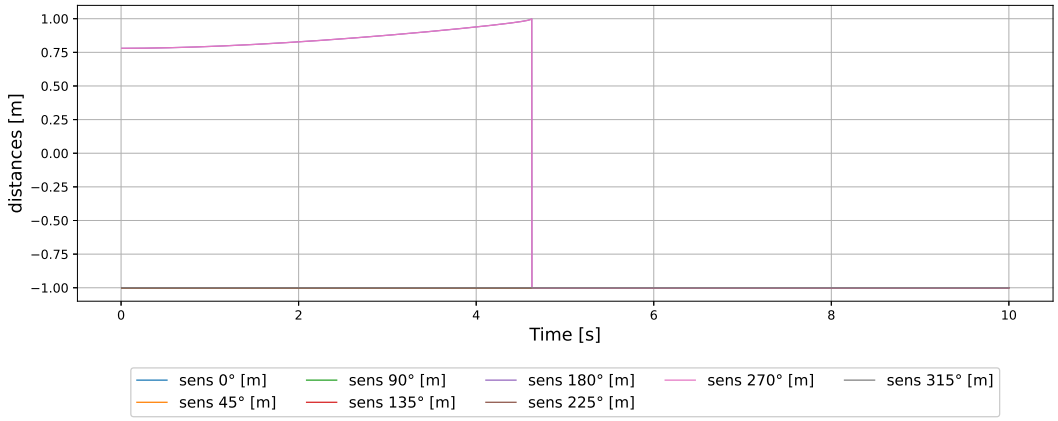

Figure 4.34: Distances of the base sensors.

Note that since the two obstacles are at the same distance from the two sensors, the plots of the distances (figure 4.38) are overlapped.

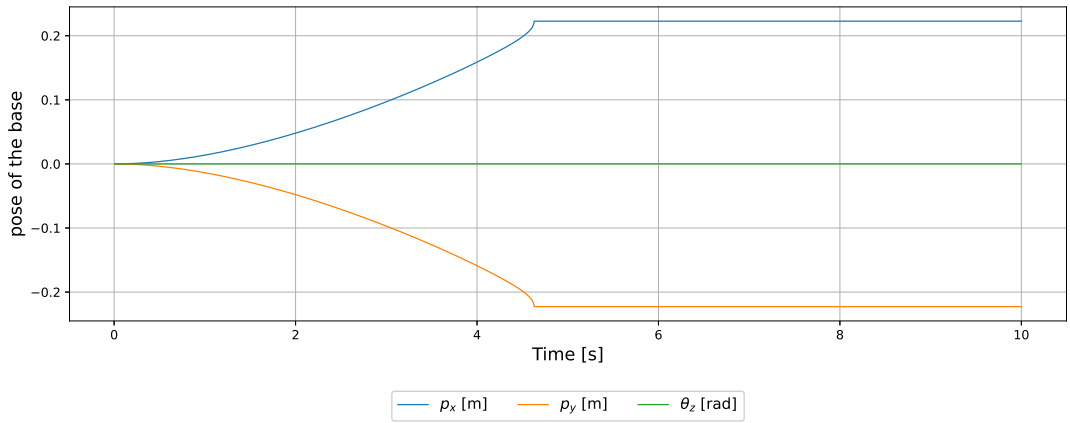

Figure 4.35: Pose of the base.

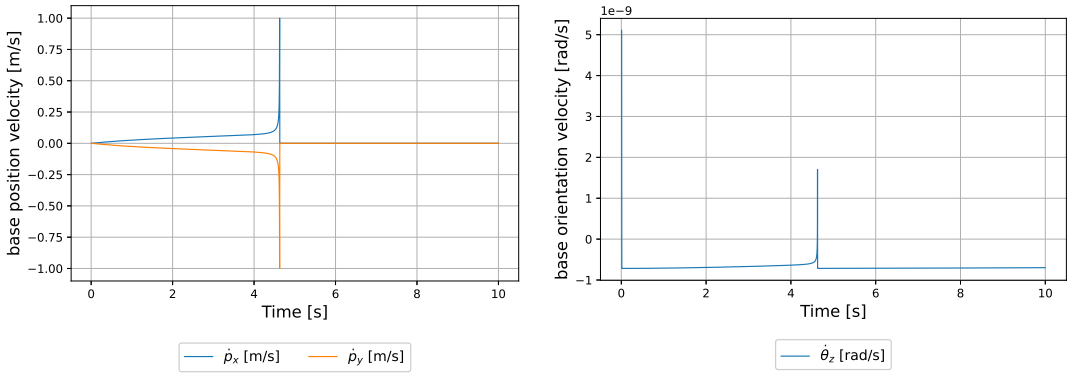

Figure 4.36: Base linear velocities (left) and base angular velocity (right).

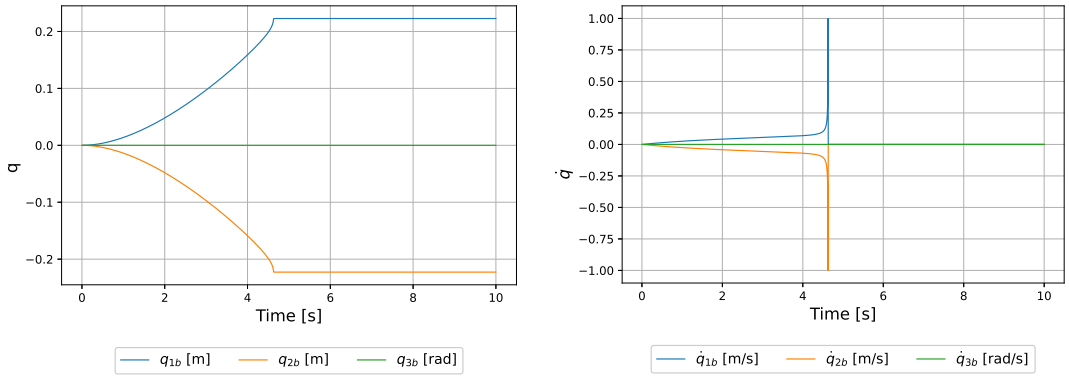

Figure 4.37: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Note how the robot moves way from obstacles in the desired trajectory time. After that, the performances in a full dynamics environment are reported.

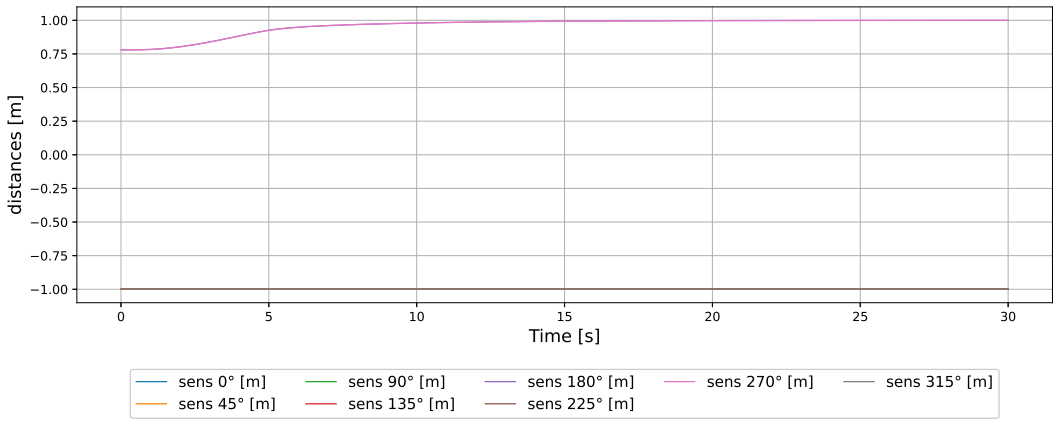

Figure 4.38: Distances of the base sensors.

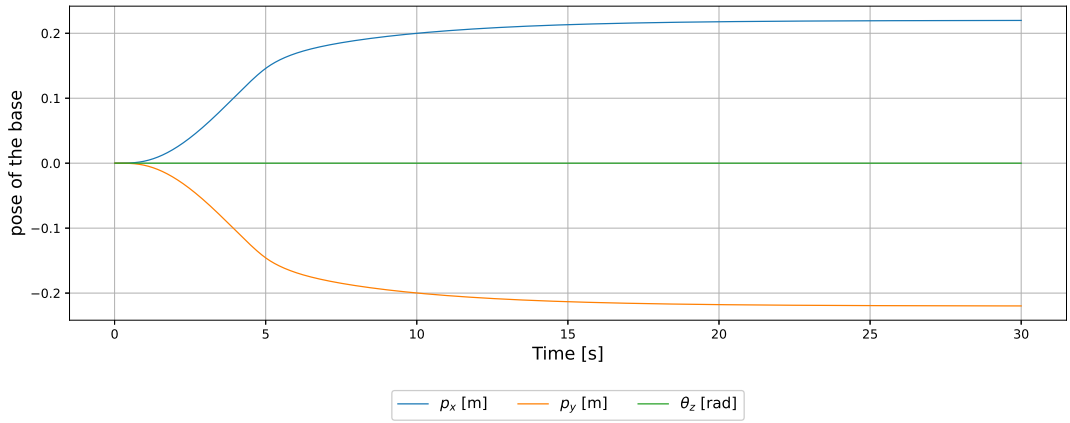

Figure 4.39: Pose of the base.

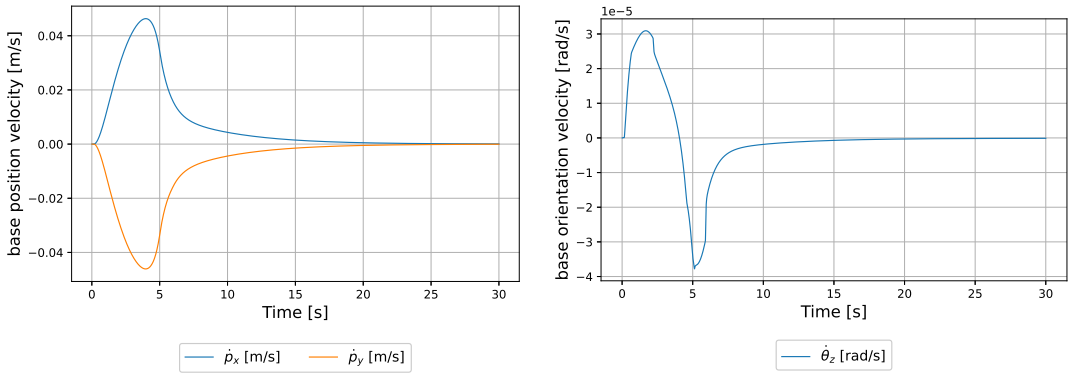

Figure 4.40: Base linear velocities (left) and base angular velocity (right).

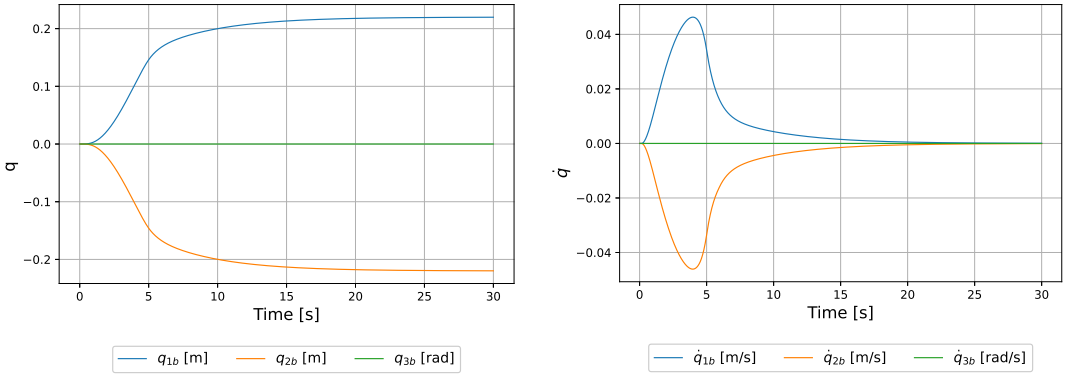

Figure 4.41: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Also in this case, the final configuration in the Cartesian space is the same as the configuration in joint space:

$$\boldsymbol{p} = [2.197 \cdot 10^{-1},\ -2.198 \cdot 10^{-1},\ 3.167 \cdot 10^{-7} \simeq 0] = \boldsymbol{q}.$$

Also with a second obstacle, final pose $\boldsymbol{p}$ and final configuration $\boldsymbol{q}$ are the same for both cases.

Master arm (right)

In this case, the master arm's kinematic chain is considered as composed by 10 degrees of freedom, 3 from the base and 7 from the arm combined together. In order to show a case of this high redundant system, two walls are placed close to the robot. In particular, one is parallel to the x axis and is placed in position $[0, -1, 0]\ m$, while the other one is parallel to the y axis and is placed in position $[-1, 0, 0]\ m$. The trajectory time is $t = 5\ s$, the gain $\gamma_o = 1$, the rest length of the springs $r_{k_{base}} = 0.5$ and $r_{k_{arm}} = s_k = 0.5\ m$. First, result for an only kinematic environment are reported.

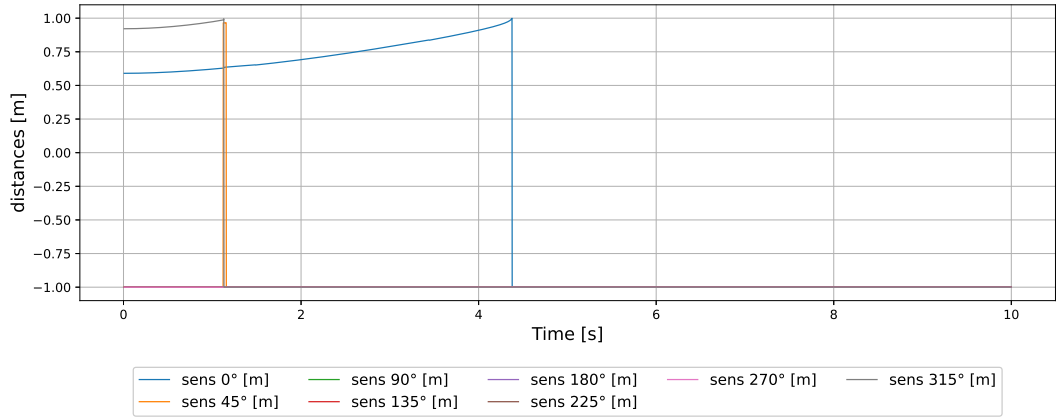

Figure 4.42: Base site distances.

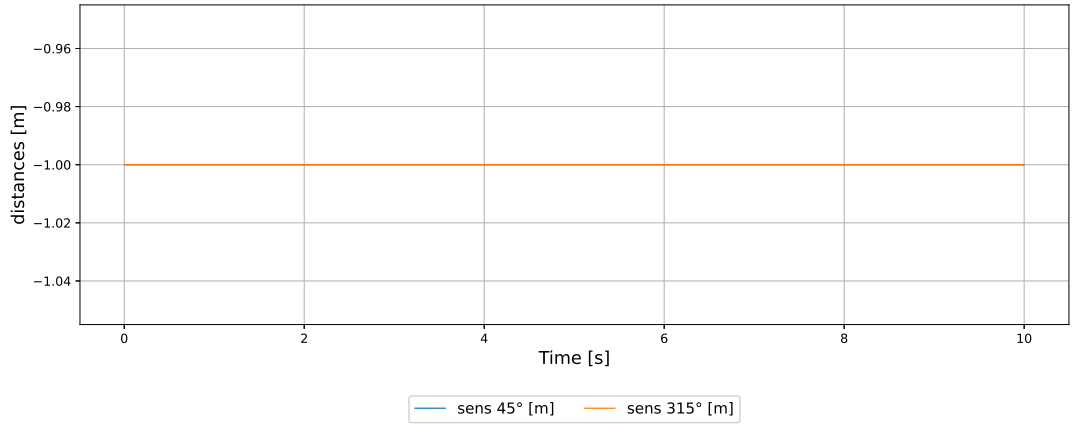

Figure 4.43: Arm 1^{st} site distances.

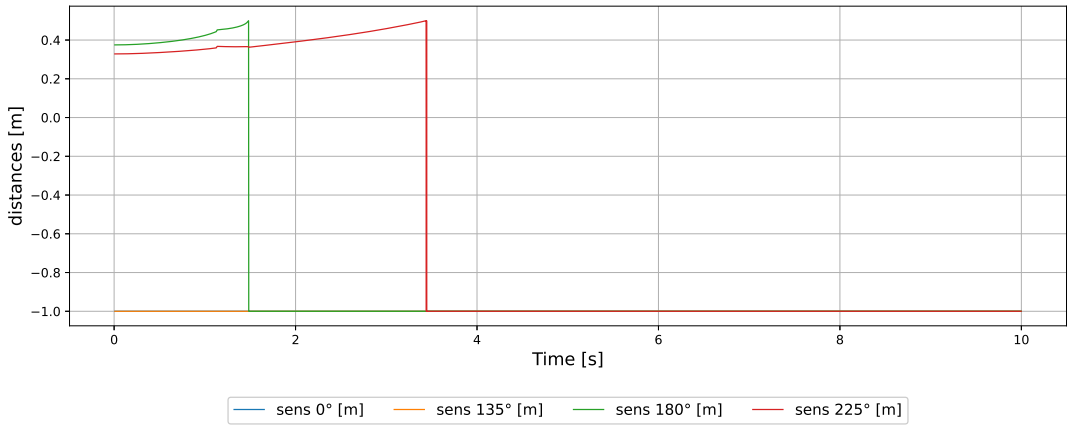

Figure 4.44: Arm 2^{nd} site distances.

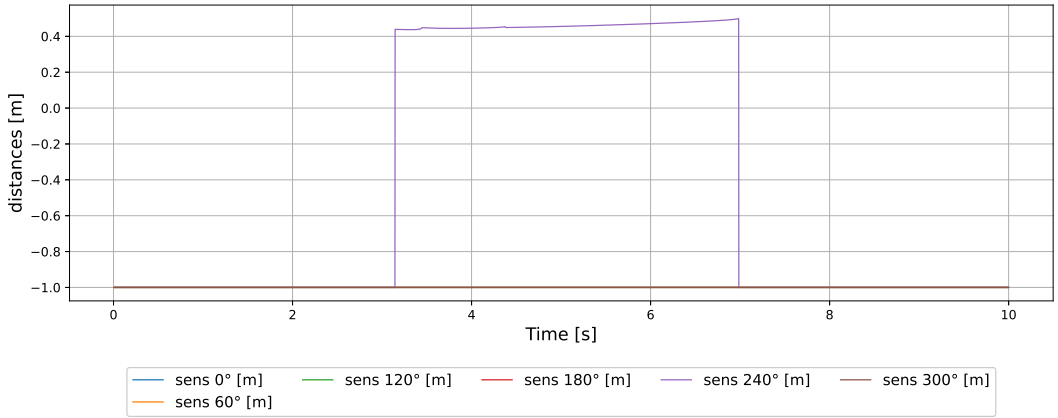

Figure 4.45: Arm 3^{rd} site distances.

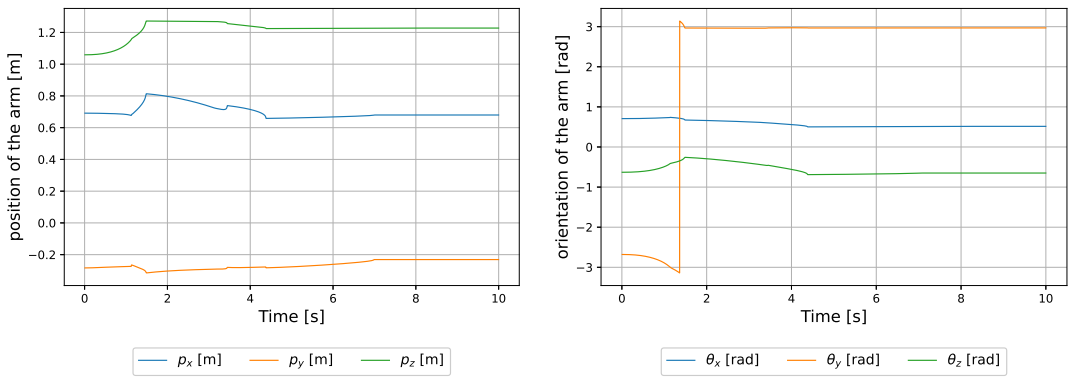

Figure 4.46: End-effector position (left) and orientation (right).

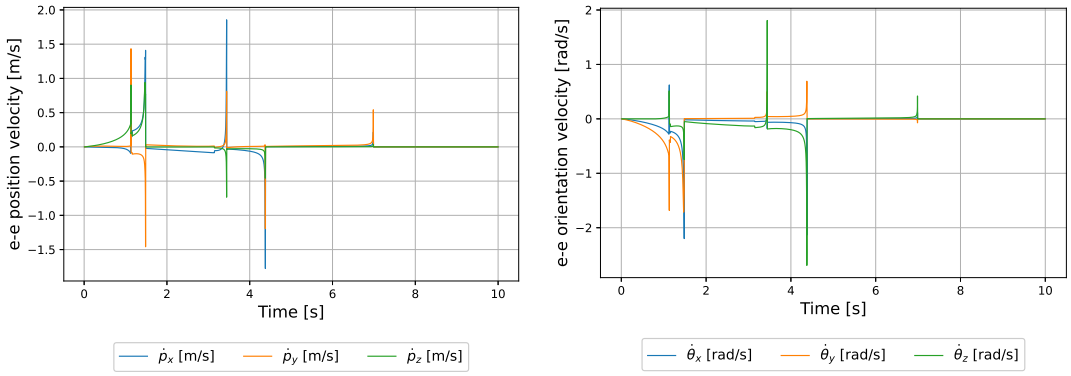

Figure 4.47: End-effector linear (left) and angular (right) velocities.

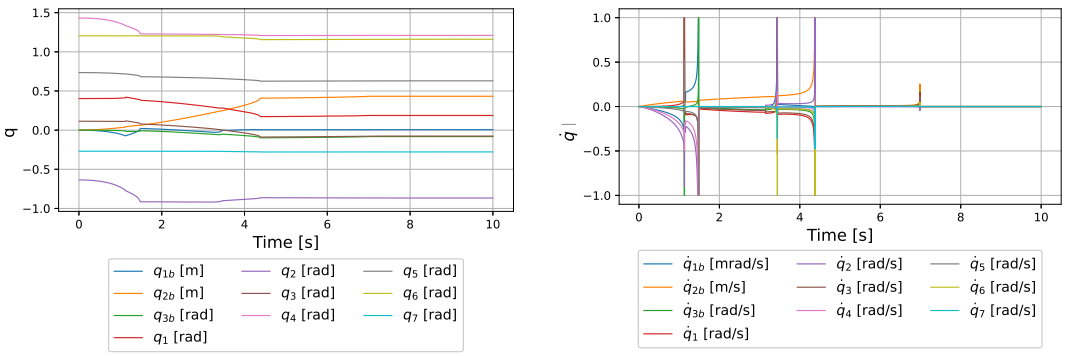

Figure 4.48: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

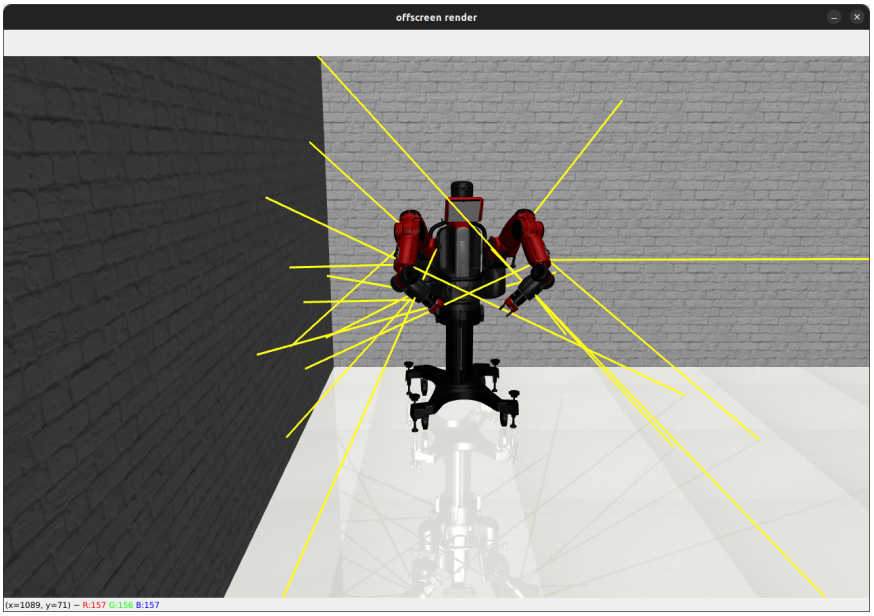

Figure 4.49: Baxter robot moving away from the wall obstacles.

The initial robot configuration is:

$$\boldsymbol{q} = [0,\ 0,\ 0,\ 0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269]$$

and, once the task is completed, the robot is in configuration

$$\boldsymbol{q} = [0.006,\ 0.432,\ -0.082,\ 0.187,\ -0.867,\ -0.076,\ 1.210,\ 0.629,\ 1.160,\ -0.278]$$

with final end-effector pose:

$$\boldsymbol{p} = [0.679,\ -0.231,\ 1.227,\ -0.649,\ 0.514,\ 2.969].$$

The arm moves away from obstacles, as expected. Observe that, in the case in which a sensor pass from the active state to the not active state, an instantaneous velocity peak follows.
Then, a simulation in the same conditions is conduced in a full dynamics environment.

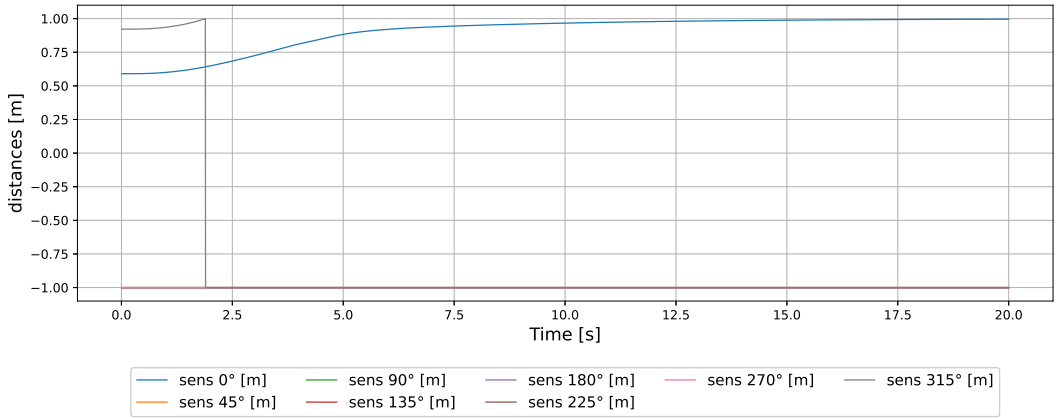

Figure 4.50: Base site distances

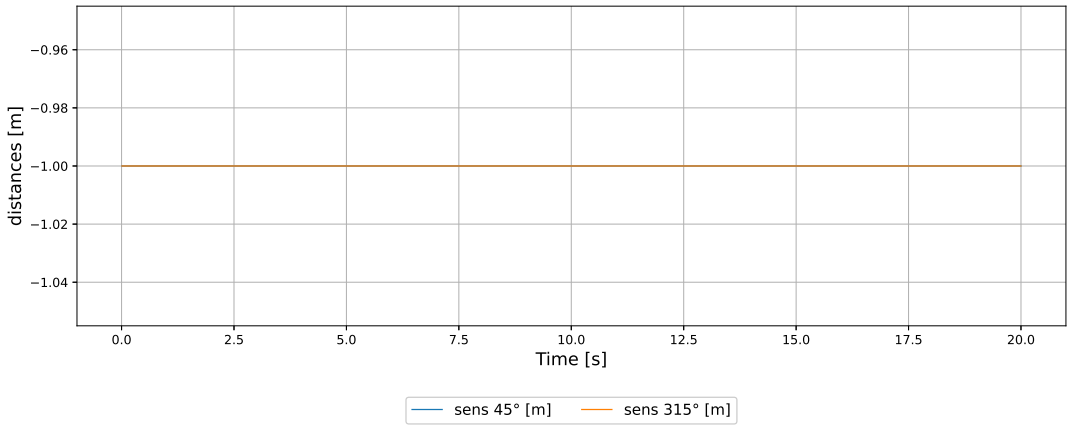

Figure 4.51: Arm 1^{st} site distances.

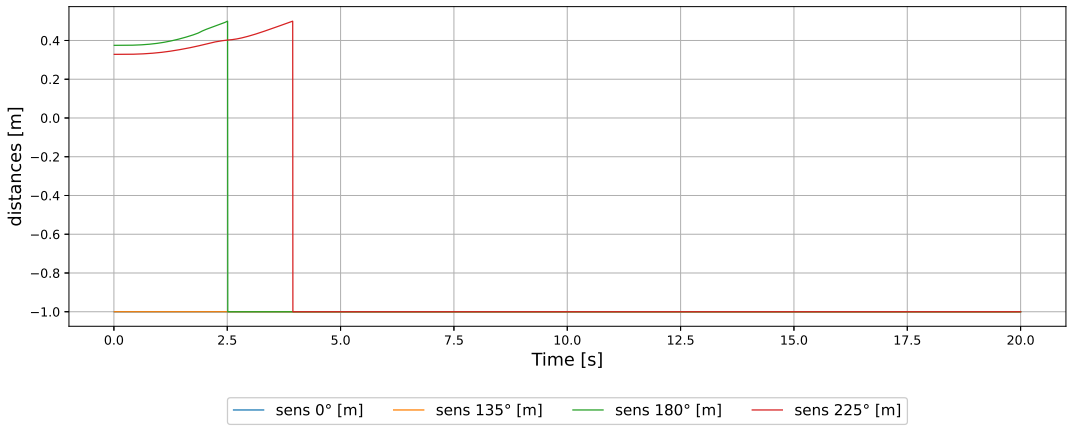

Figure 4.52: Arm 2^{nd} site distances.

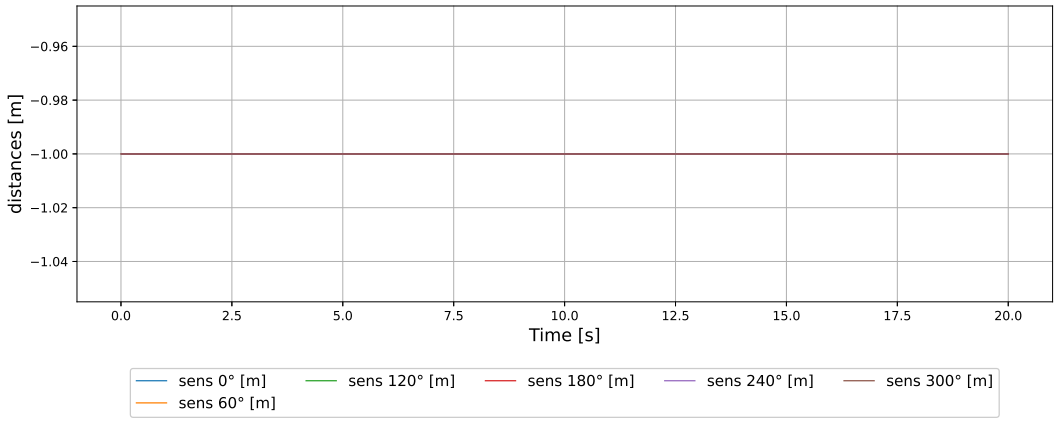

Figure 4.53: Arm 3^{rd} site distances.

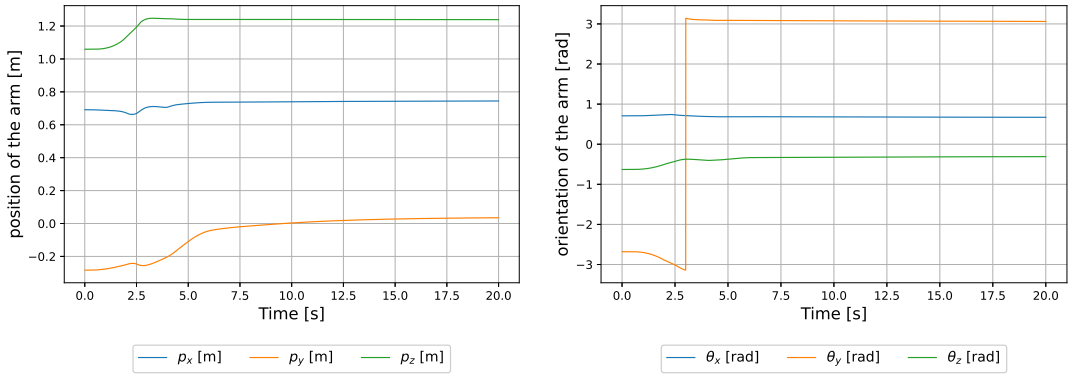

Figure 4.54: End-effector position (left) and orientation (right).

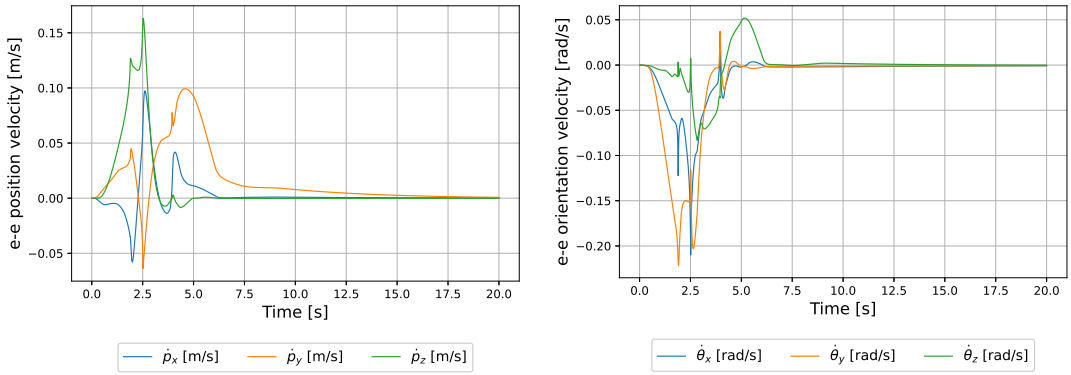

Figure 4.55: End-effector linear (left) and angular (right) velocities.

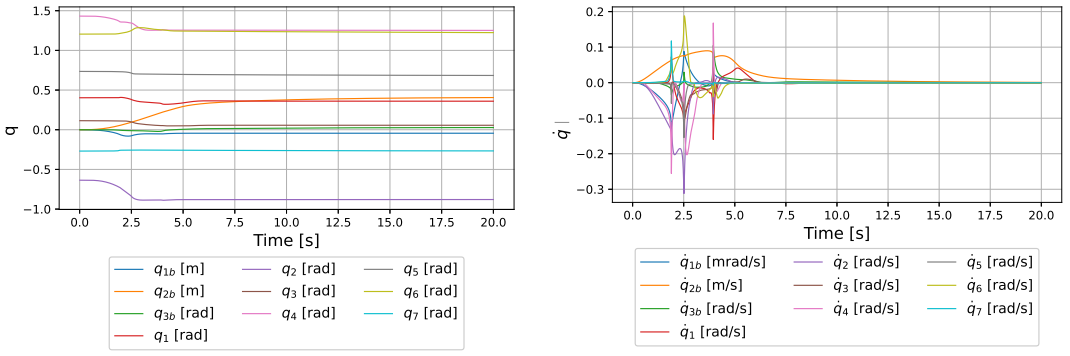

Figure 4.56: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

In this case the final configuration reached by the robot is:

$$\boldsymbol{q} = [-0.044,\ 0.406,\ 0.027,\ 0.359,\ -0.879,\ 0.056,\ 1.252,\ 0.6841.224,\ -0.267]$$

which imply the final pose for the end-effector:

$$\boldsymbol{p} = [0.744,\ 0.0349,\ 1.238,\ -0.310,\ 0.671,\ 3.062].$$

Slave arm (right)

In this case, the master arm is considered with it's kinematic chain composed by 7 degrees of freedom given only by the arm, as for the original robot. In order to show results for this case, a wall is placed close to the robot. In particular, it parallel to the x axis and is placed in position $[0,\ -0.65,\ 0]\ m$ with respect to the robot base frame. The trajectory time is $t = 5\ s$, the gain $\gamma_o = 1$, $r_{k_{arm}} = 0.5\ m$. The robot's arm starts in pose $\boldsymbol{p} = [p_x,\ p_y,\ p_z,\ \theta_x,\ \theta_y,\ \theta_z]$:

$$\boldsymbol{p} = [0.691,\ -0.284,\ 0.13499078,\ 0.707,\ -2.681,\ -0.630]$$

with its end-effector, given by the joints configuration

$$\boldsymbol{q} = [0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269]$$

First, the results obtained in an environment with only kinematics are reported to show the behaviour of the robot.

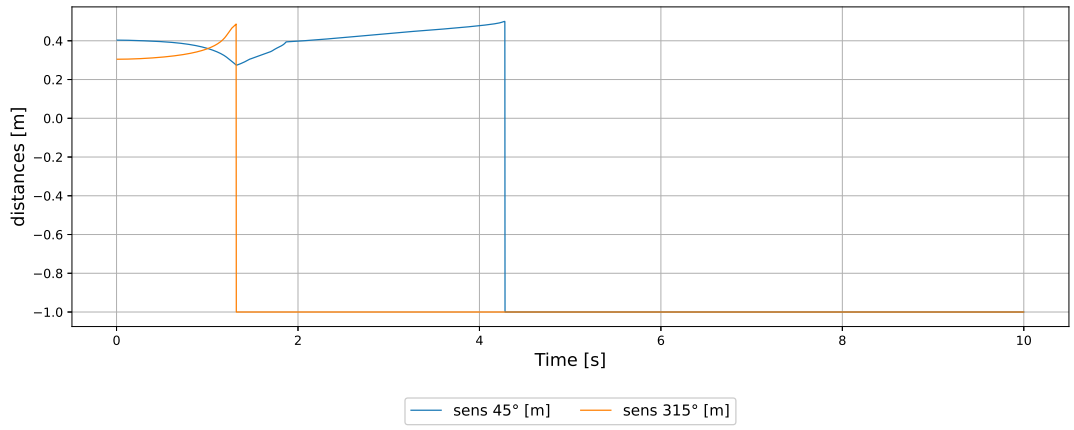

Figure 4.57: Arm 1^{st} site distances.

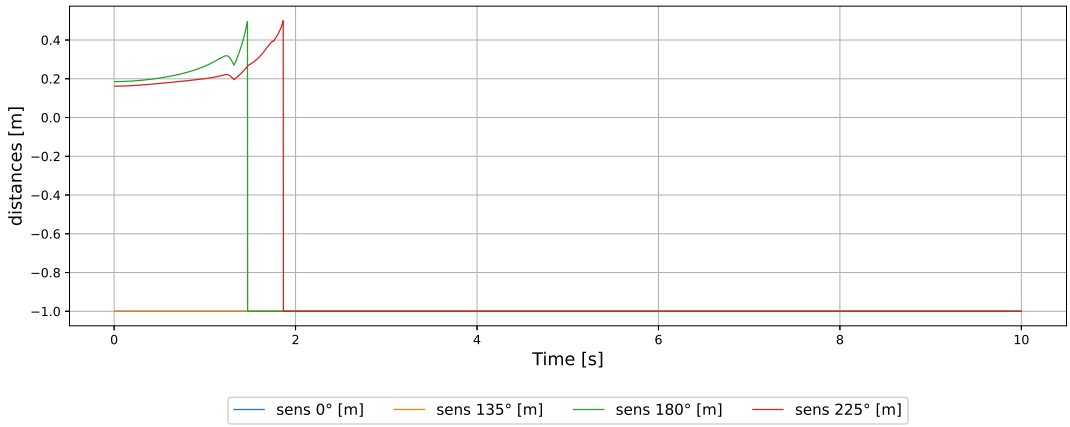

Figure 4.58: Arm 2^{nd} site distances.

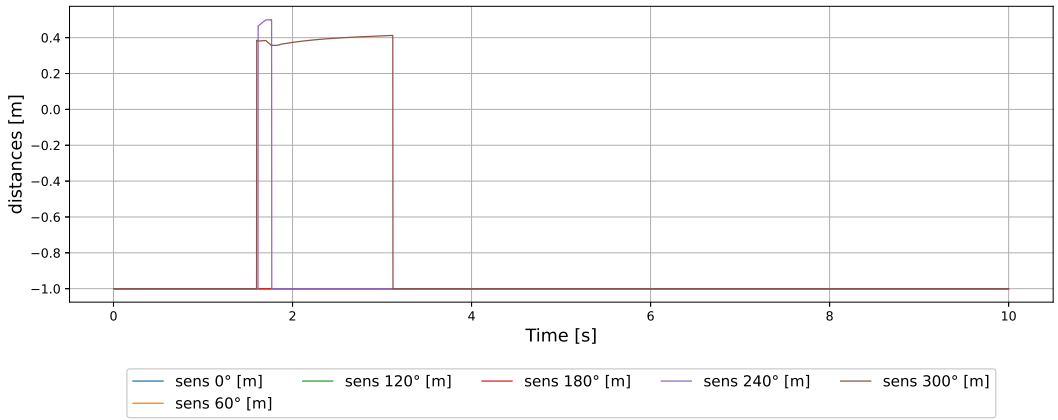

Figure 4.59: Arm 3^{rd} site distances.

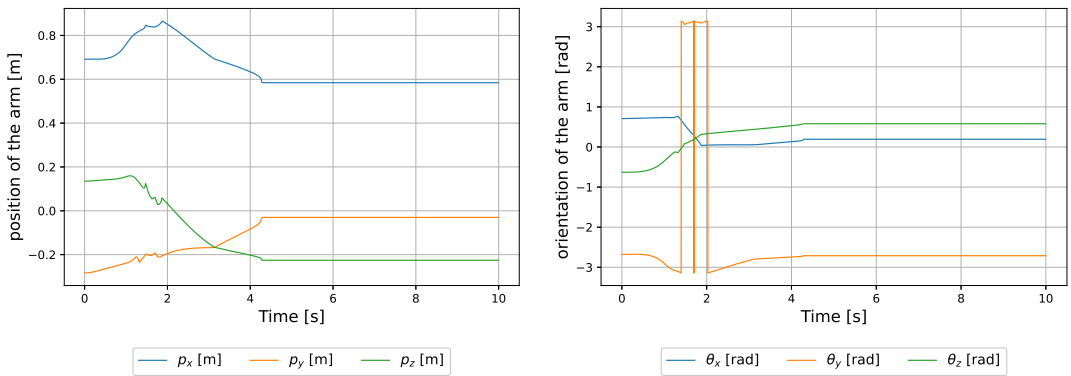

Figure 4.60: End-effector position (left) and orientation (right).

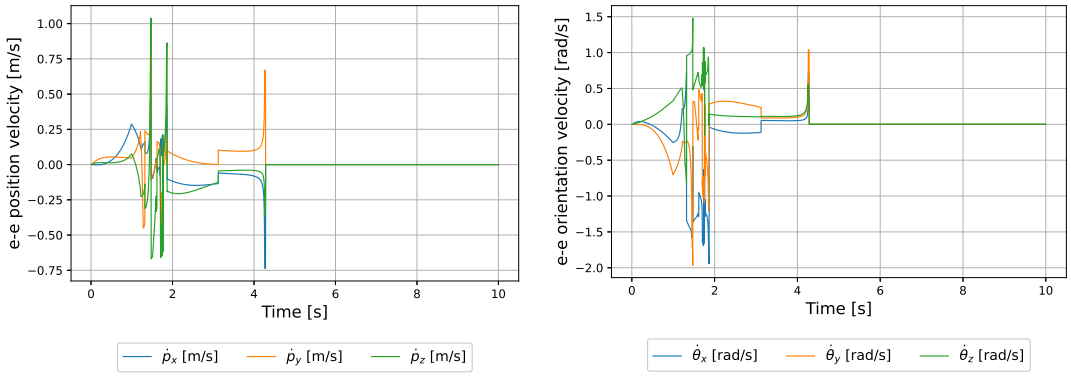

Figure 4.61: End-effector linear (left) and angular (right) velocities.

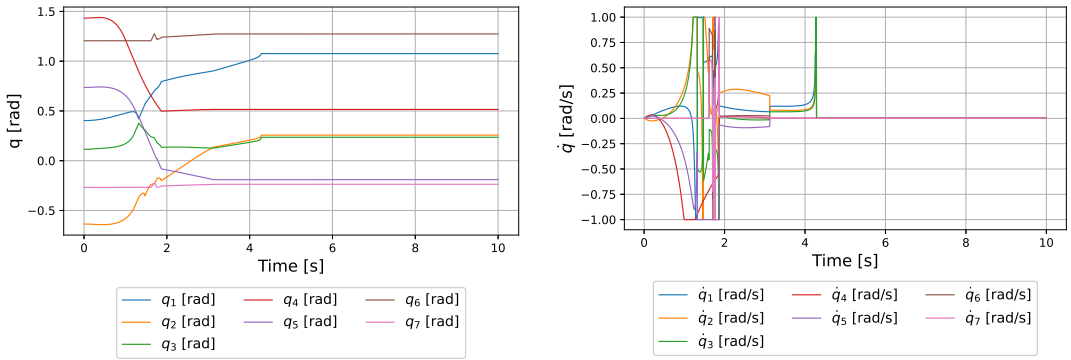

Figure 4.62: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot accomplish its task and moves away from obstacles. It reaches a final pose

$$\boldsymbol{p} = [0.584,\ -0.030,\ -0.226,\ 0.579,\ 0.192,\ -2.712]$$

with a joint configuration

$$\boldsymbol{q} = [1.076,\ 0.257,\ 0.235,\ 0.515,\ -0.191,\ 1.274,\ -0.237].$$

It is possible to observe how a change o active sensors, namely a change of the Jacobian matrix $\boldsymbol{J}_o$, implies instantaneous velocities.
After that, the results obtained in a full dynamics environment are reported.

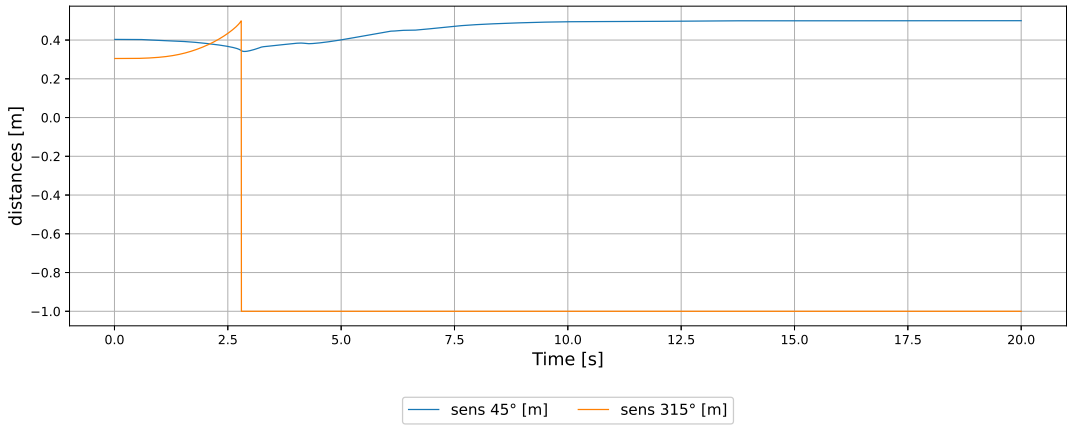

Figure 4.63: Arm 1^{st} site distances.

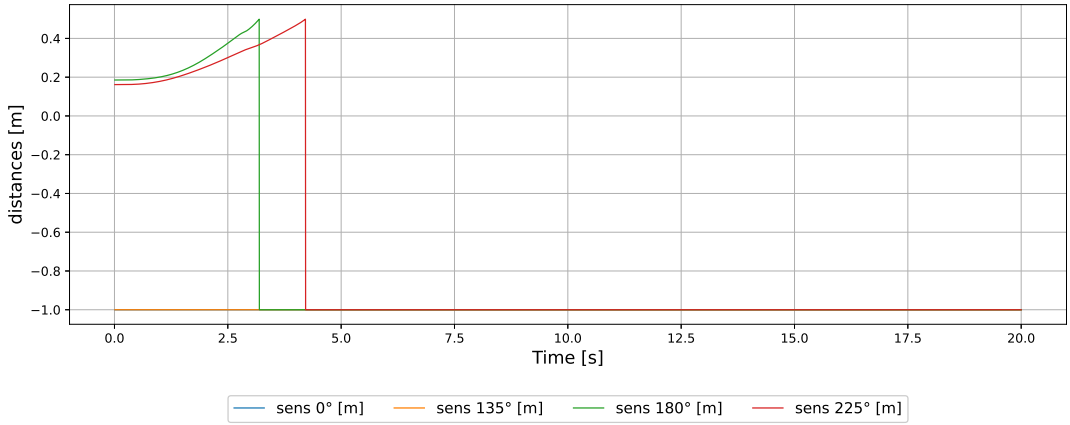

Figure 4.64: Arm 2^{nd} site distances.

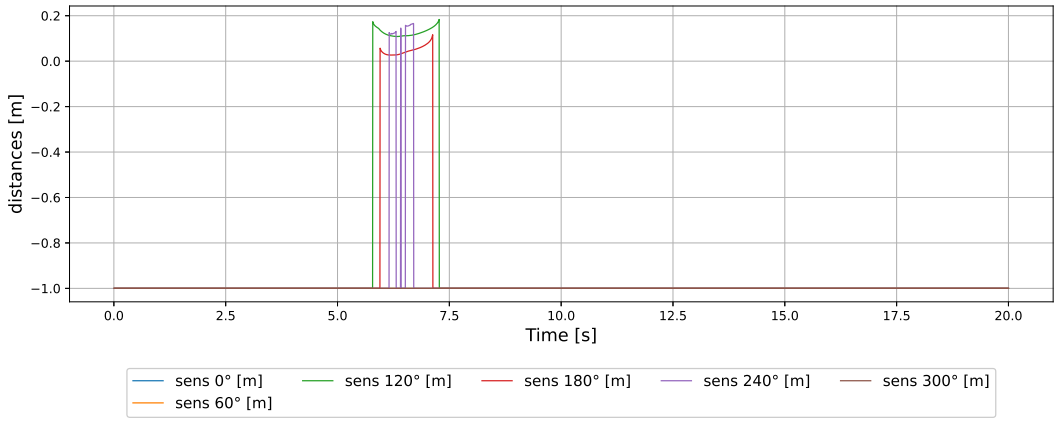

Figure 4.65: Arm 3^{rd} site distances.

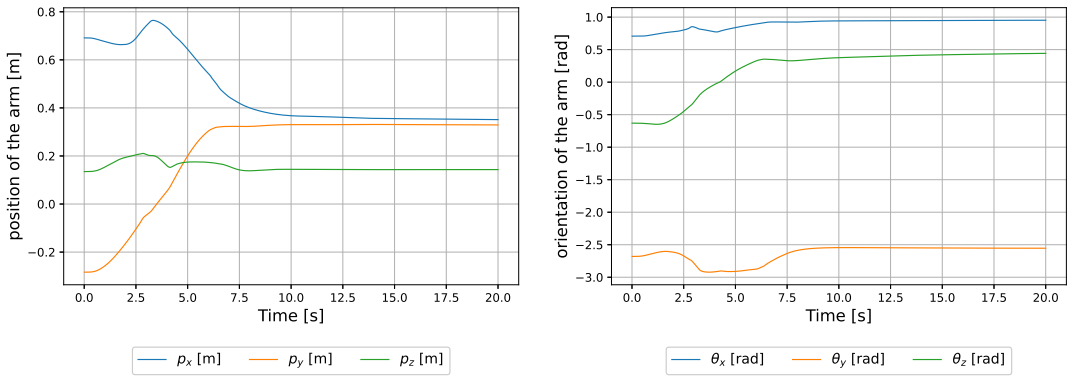

Figure 4.66: End-effector position (left) and orientation (right).

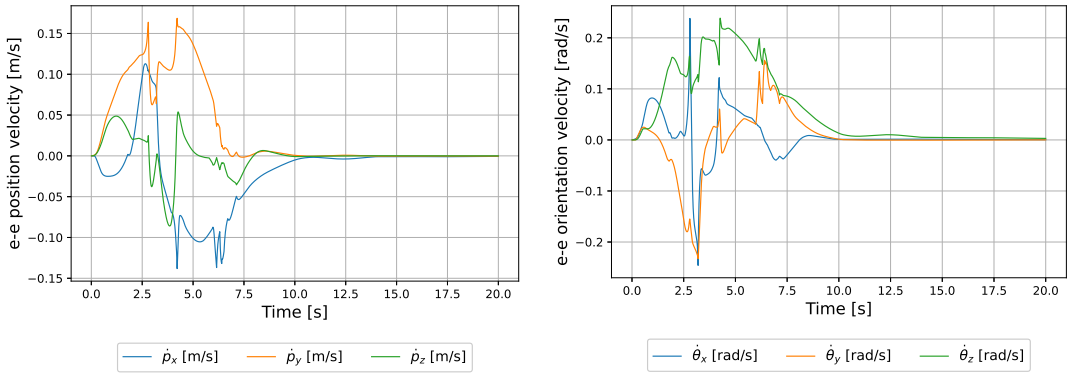

Figure 4.67: End-effector linear (left) and angular (right) velocities.

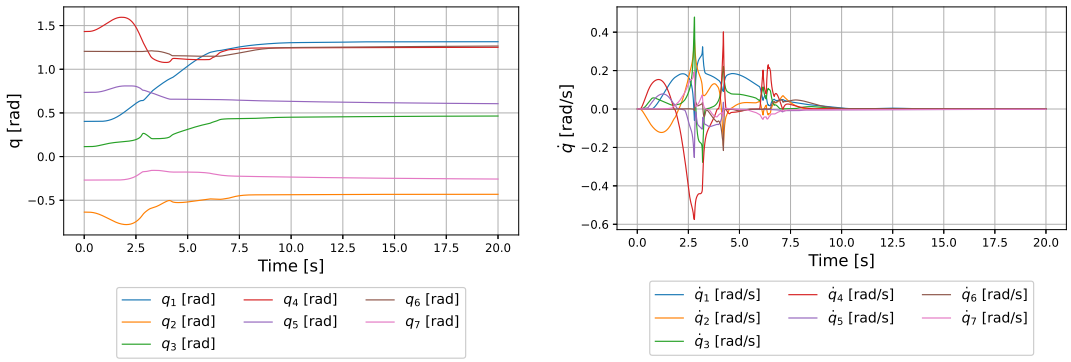

Figure 4.68: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot reaches a final pose

$$\boldsymbol{p} = [0.351,\ 0.329,\ 0.143,\ 0.443,\ 0.952,\ -2.554],$$

with joint configuration

$$\boldsymbol{p} = [1.315,\ -0.432,\ 0.464,\ 1.251,\ 0.606,\ 1.266,\ -0.257].$$

As observed in previous cases, the task is solved in more time with respect to the only kinematic environment, and one distance reaches r_k to infinity.

Dynamic obstacles

It is possible to evaluate the performances of the obstacle avoidance task in the case in which the obstacle moves during the simulation time and it is not fixed. As an example, some cases with the mobile base are reported.
In the first one an object simply move towards the robot, moving along the y axis with negative velocity $v = -0.5\ m/s$ for $4\ s$.
The initial pose of the base is

$$\boldsymbol{p} = [0,\ 0,\ 0]$$

given by the configuration in joint space

$$\boldsymbol{q} = [0,\ 0,\ 0].$$

First, the result for the case with kinematics only are reported.

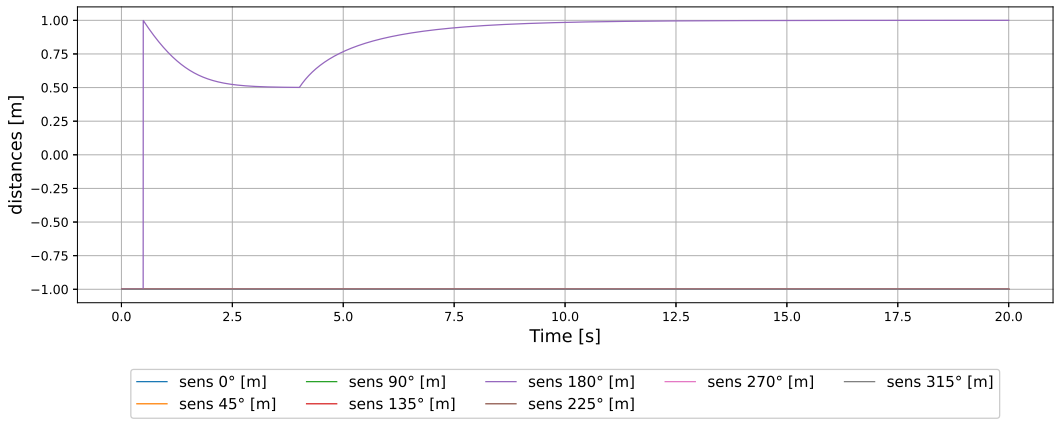

Figure 4.69: Distanced of the base sensors.

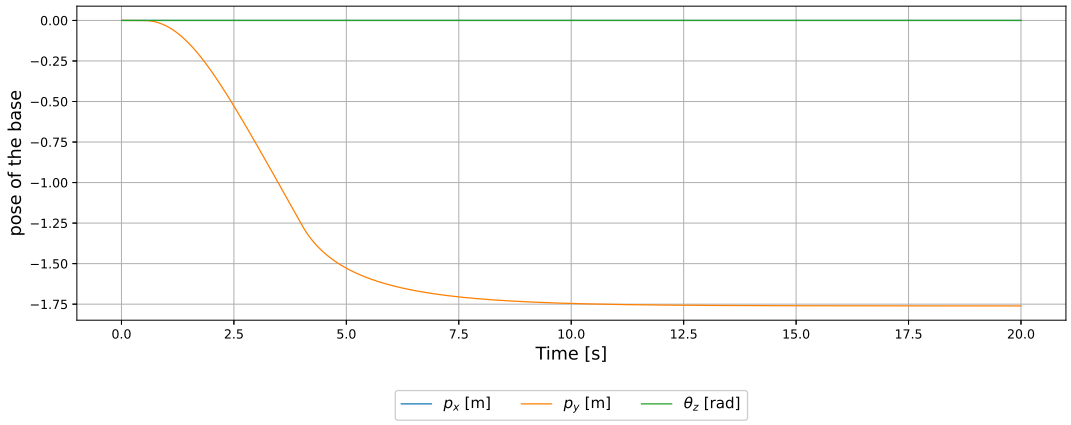

Figure 4.70: Pose of the base.

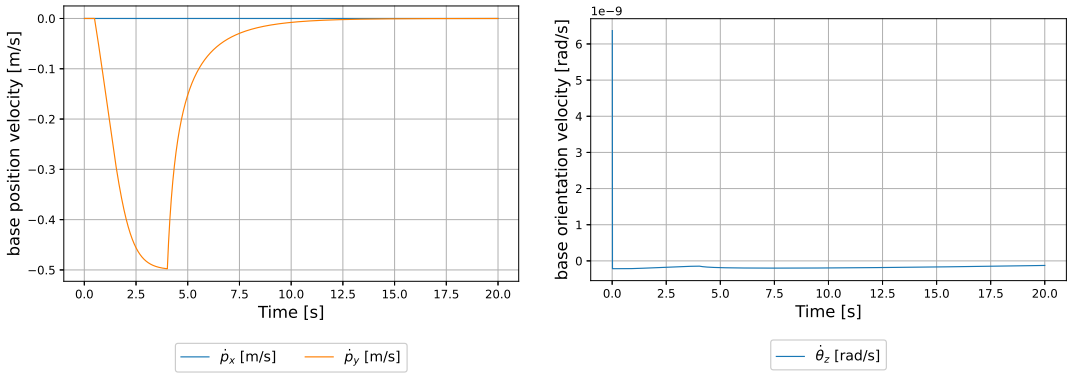

Figure 4.71: Base linear velocities (left) and base angular velocity (right).

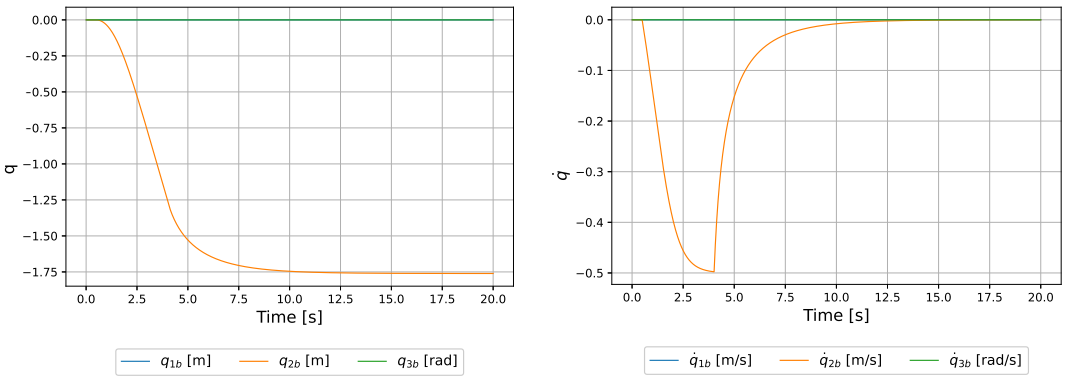

Figure 4.72: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

From plot in figure 4.69 it is possible to note that at the beginning of the simulation no obstacles are sensed. Then, the dynamic obstacle is sensed by the 180° sensor and the robot starts to move. In the first phase the sensed distance

is still decreasing due to the fact that the obstacle is faster with respect to the response imposed by the robot. When it is closer, the robot starts to move faster to avoid collision and the slope of the distance measure is almost flat. Once the the obstacle stops, it is possible to see how the robot moves away until the sensed distance is almost equal to the rest length of the spring, namely $d_k \simeq r_{k_{base}}$. At the end the robot reaches final configuration

$$\boldsymbol{q} = [0,\ -1.761,\ 0].$$

After that, the full dynamic case is showed.

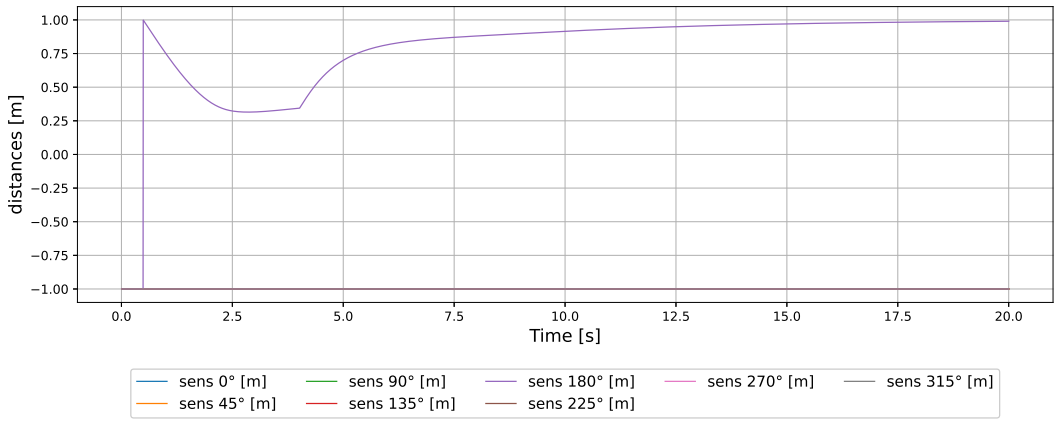

Figure 4.73: Distanced of the base sensors.

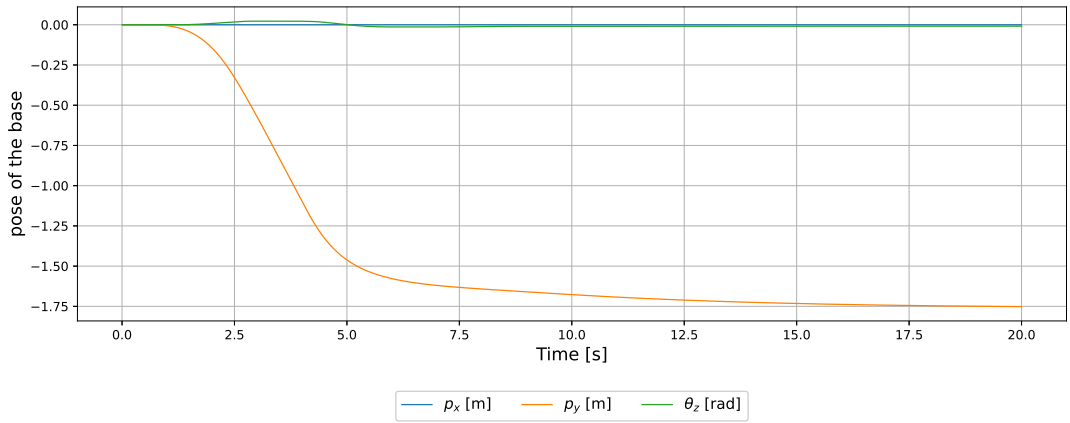

Figure 4.74: Pose of the base.

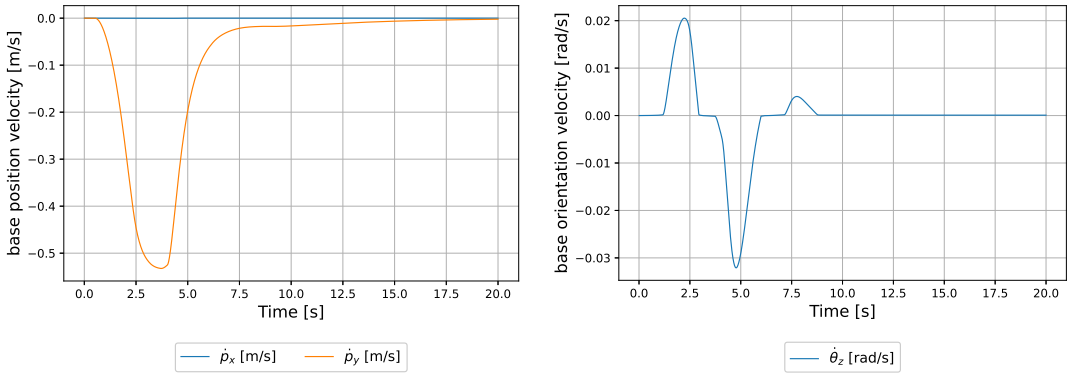

Figure 4.75: Base linear velocities (left) and base angular velocity (right).

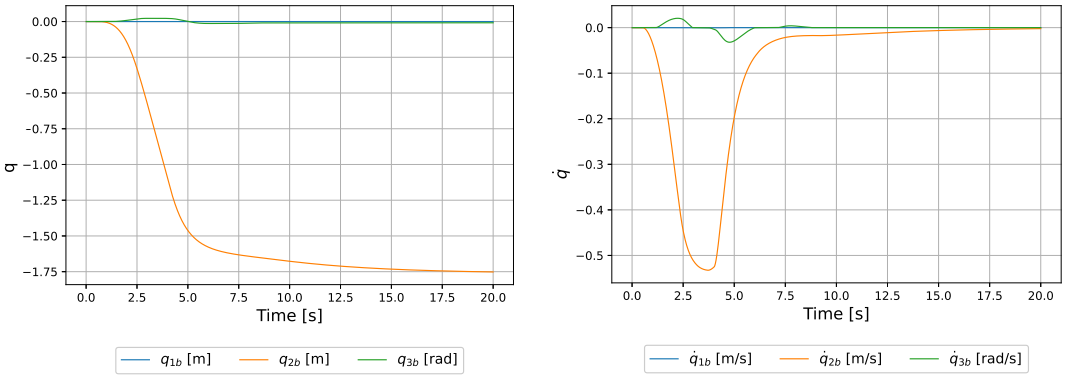

Figure 4.76: End-effector linear (left) and angular (right) velocities.

At the end, the robot reaches the final configuration:

$$\boldsymbol{q} = [0,\ -1.746,\ 0].$$

It is possible to observe how, also in the full dynamic case, the robot is able to escape from dynamic obstacles.

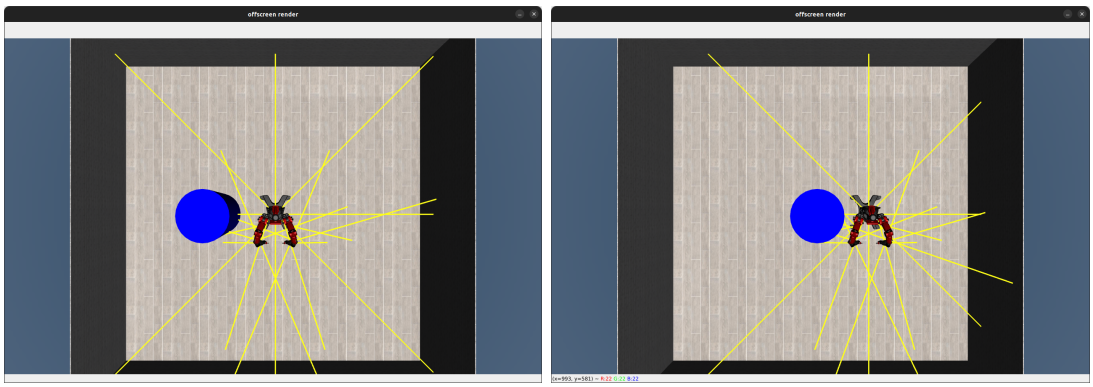

Figure 4.77: Example of the robot moving away from a dynamic obstacle.

4.1.3 Maximization of the Manipulability Measure

In this section the *maximization of the manipulability measure* 2.1.4 will be exploited with simulations and results. Only the arms will be considered, since the base is not involved in this task, and its inclusion will not provide benefit to the final result. It is important to precise that the maximum is still a local one, which is influenced by the initial configuration of the arm. It is impossible to define a priory a global maximum (or the value of the local one) for this measure.
In this case, only the results for the slave arm are reported (only right since the left is almost the same), since the base is not implied in the solution of this task as previously said. The chosen gain for this demonstration is $k_0 = 100$. The initial robot configuration in joint space is

$$\boldsymbol{q} = [0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269]$$

with initial pose of the end-effector

$$\boldsymbol{p} = [0.691,\ -0.284,\ 0.135,\ 0.707,\ -2.680,\ -0.630].$$

The manipulability measure in the initial configuration is $w_{manip} = 0.124$. First, the plots for the only kinematic case are reported.

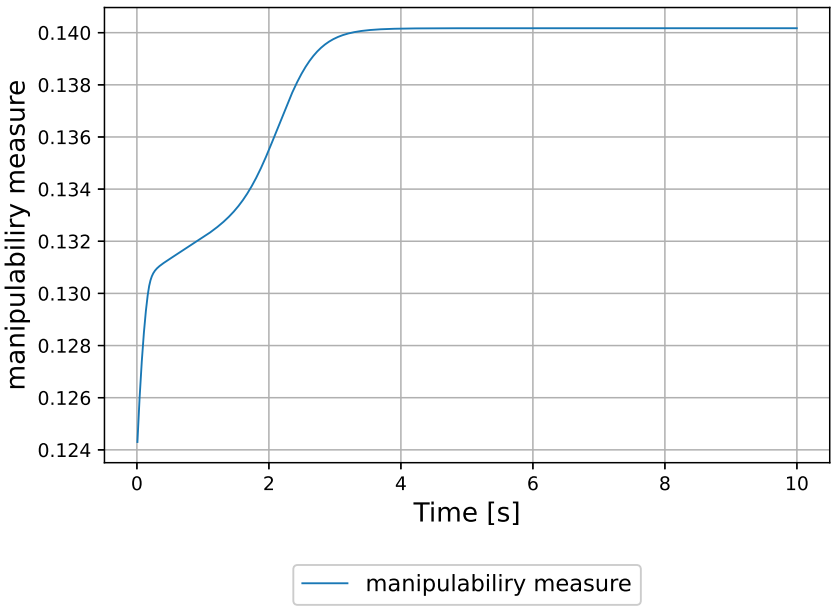

Figure 4.78: Manipulability measure of the right arm.

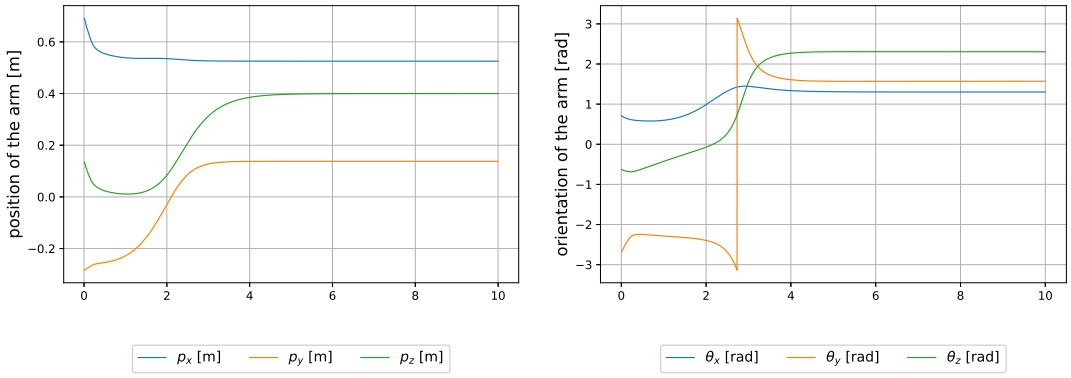

Figure 4.79: End-effector position (left) and orientation (right).

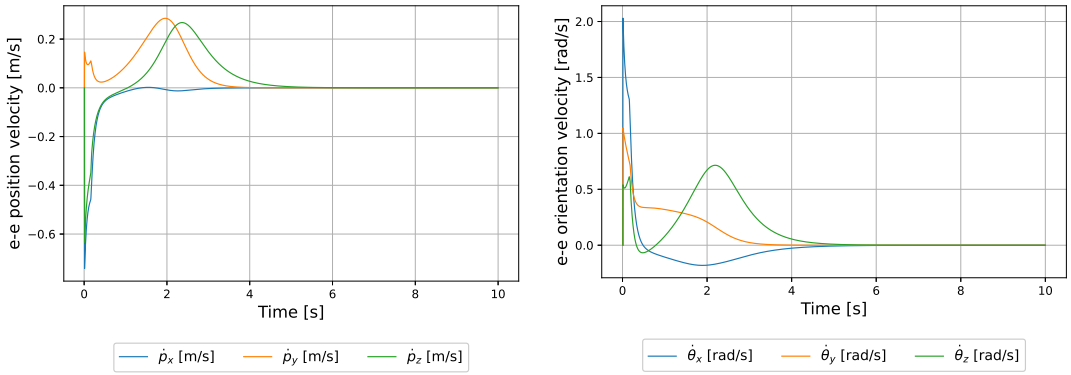

Figure 4.80: End-effector linear (left) and angular (right) velocities.

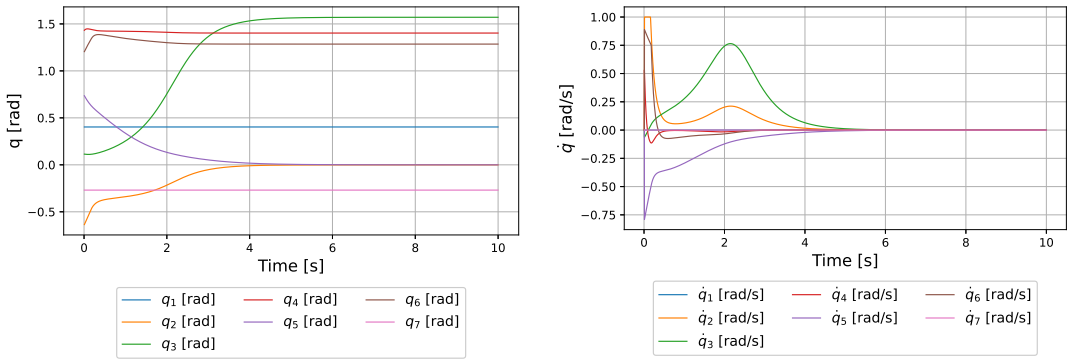

Figure 4.81: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Then the robot ends the maximization process with configuration

$$\boldsymbol{q} = [4.030 \cdot 10^{-1},\ -2.948 \cdot 10^{-7},\ 1.571,\ 1.403,\ 3.220 \cdot 10^{-5},\ 1.285,\ -2.690 \cdot 10^{-1}],$$

with end-effector pose

$$\boldsymbol{p} = [0.525,\ 0.138,\ 0.399,\ 2.306,\ 1.301,\ 1.571]$$

and manipulability measure $w_{manip} = 0.1402$.
Then, the plots of the full dynamics case are reported.

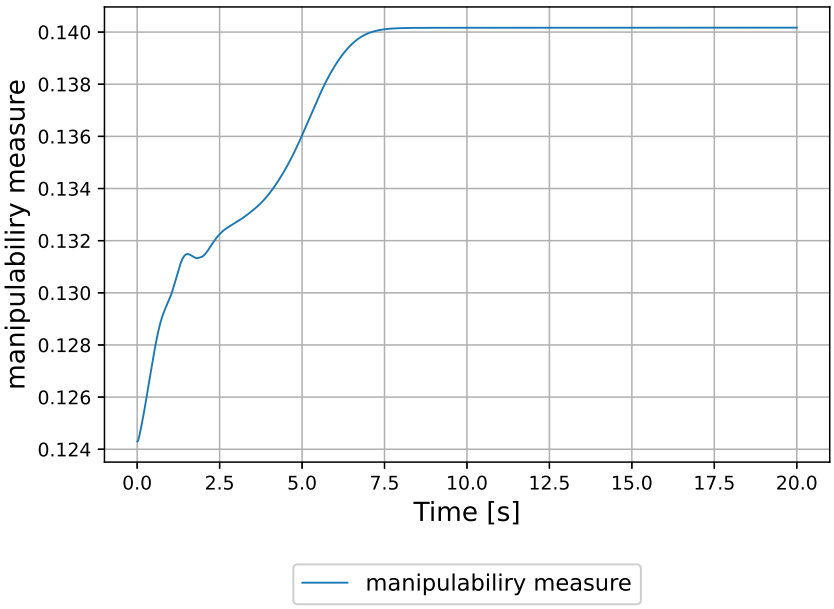

Figure 4.82: Manipulability measure of the right arm.

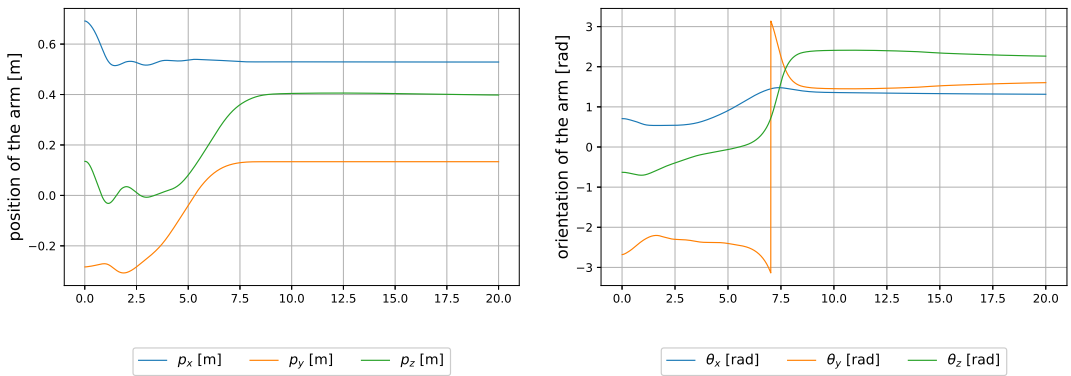

Figure 4.83: End-effector position (left) and orientation (right).

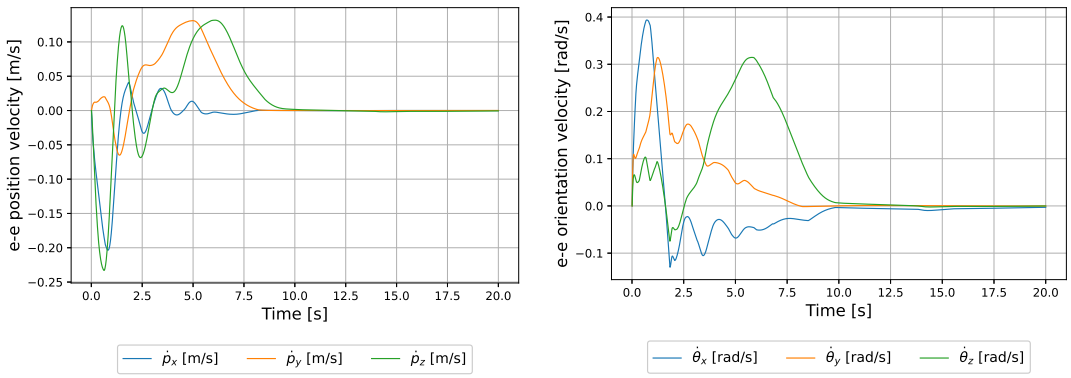

Figure 4.84: End-effector linear (left) and angular (right) velocities.

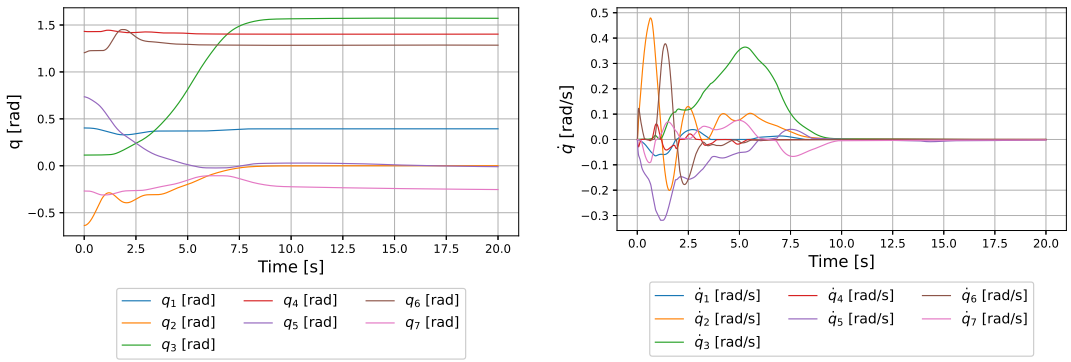

Figure 4.85: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Then the arm reaches the joints configuration

$$\boldsymbol{q} = [3.940 \cdot 10^{-1},\ 6.602 \cdot 10^{-4},\ 1.572,\ 1.403,\ -9.711 \cdot 10^{-3},\ 1.285,\ -2.532 \cdot 10^{-1}]$$

the end-effector pose

$$\boldsymbol{p} = [0.529,\ 0.134,\ 0.398,\ 2.264,\ 1.314,\ 1.604].$$

and manipulability measure $w_{manip} = 0.1402$.

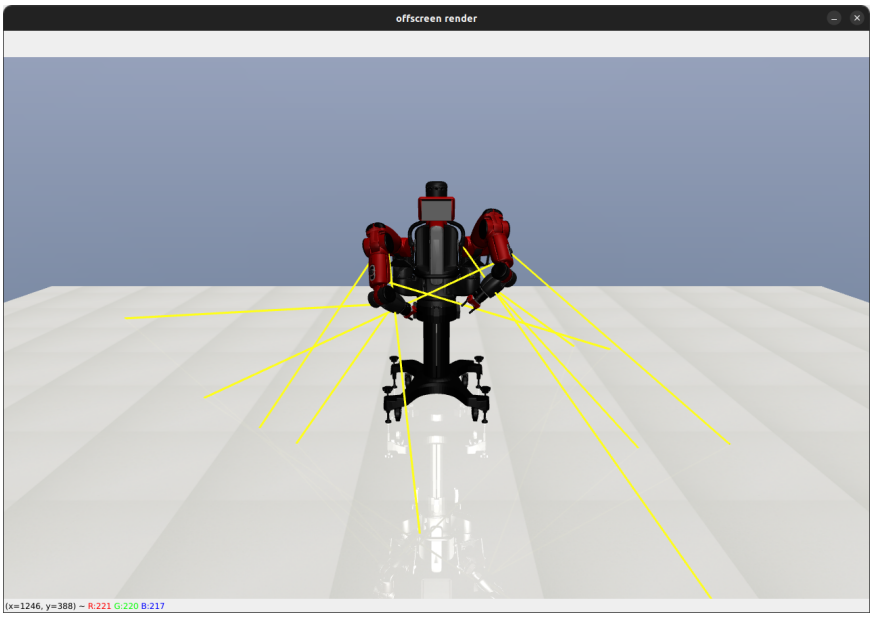

Figure 4.86: Maximization of the manipulability of the right arm.

It is possible to note how the manipulability measure is the same at the end of both simulations, with dynamics in environment and not. Once the arm reaches the local maximum it stops and do not move in order to exploit the neighborhood of the solution. However, the convergence to such configuration is slower in the second case and velocities are less smooth. Finally note that the two different configurations and poses reached are similar in the two different cases.

4.1.4 Maximization of the Distance from Mechanical Joint Limits

In this section the *maximization of the distance from mechanical joint limits* 2.1.5 will be exploited with simulation results. Only the arms will be considered, since the base is not involved in this procedure, and it's inclusion will not provide benefit to the final result. Unlike the previous maximization process, in this case it is possible to derive a global maximum which is $w_{m.j.l.} = 0$.

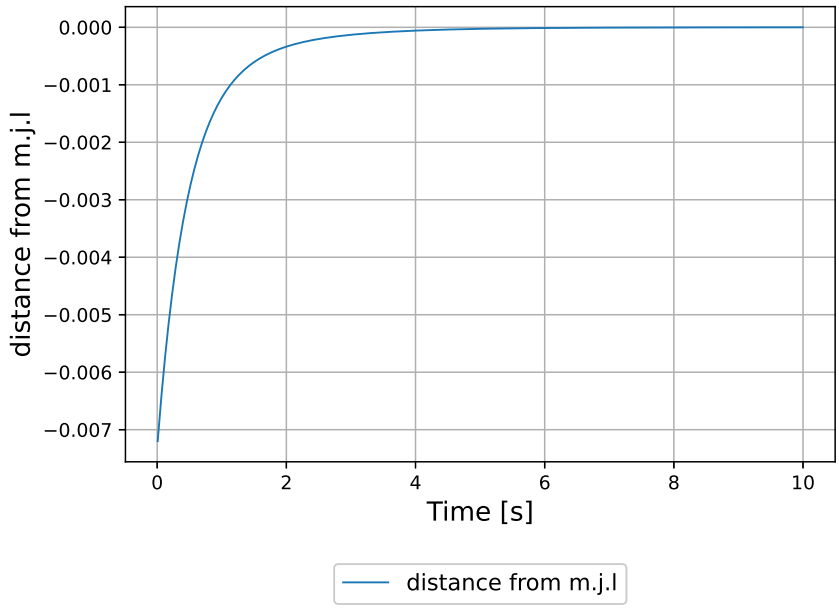

Figure 4.87: Distance from mechanical joint limits measure of the right arm.

Independently from the original configuration of the arm, it will tend to nullify the measure without local maximums, pointing toward the global one. The robot's arm starts in initial configuration

$$\boldsymbol{q} = [0.403,\ -0.636,\ 0.114,\ 1.432,\ 0.735,\ 1.205,\ -0.269]$$

with initial pose of the end-effector

$$\boldsymbol{p} = [0.691,\ -0.284,\ 0.135,\ 0.707,\ -2.680,\ -0.630].$$

First, the plots for the only kinematic case are reported.

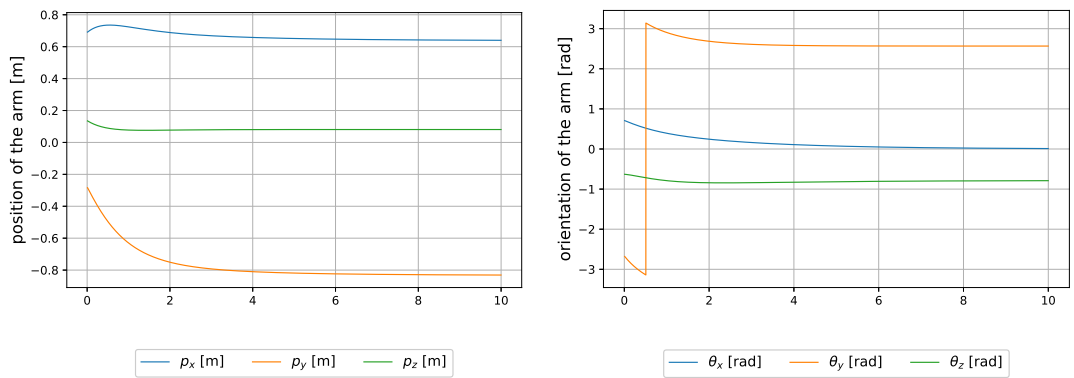

Figure 4.88: End-effector position (left) and orientation (right).

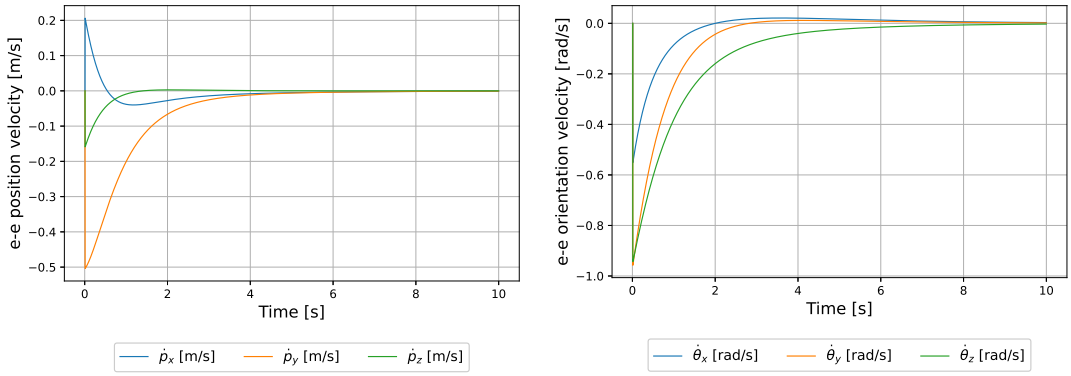

Figure 4.89: End-effector linear (left) and angular (right) velocities.

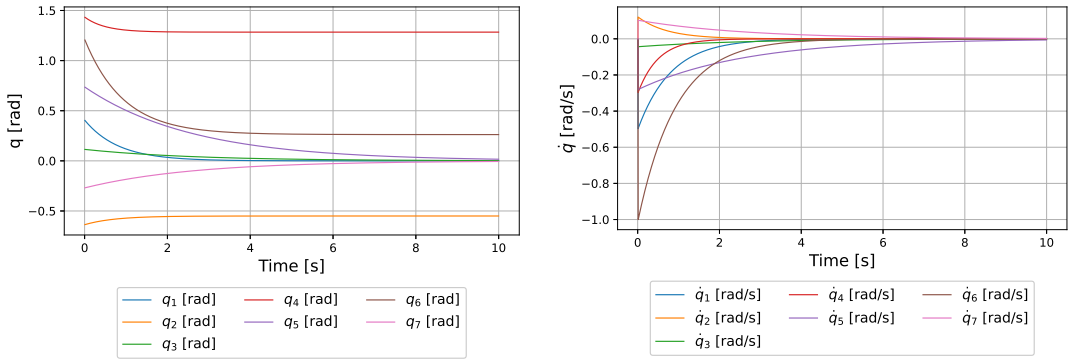

Figure 4.90: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot reaches the final arm configuration

$$\boldsymbol{q} = [-3.392{\cdot}10^{-7}, -5.500{\cdot}10^{-1}, 2.511{\cdot}10^{-3}, 1.284, 1.627{\cdot}10^{-2}, 2.620{\cdot}10^{-1}, -5.985{\cdot}10^{-3}]$$

with final pose

$$\boldsymbol{p} = [0.639,\ -0.831,\ 0.081,\ -0.789,\ 0.011,\ 2.567].$$

The initial value of the distance from mechanical joint limits was $w_{m.j.l.} = -0.0071$, at the end the value is $w_{m.j.l.} = -5.8630 \cdot 10^{-7} \simeq 0$.
After that, the performances in a full dynamics environment are reported.

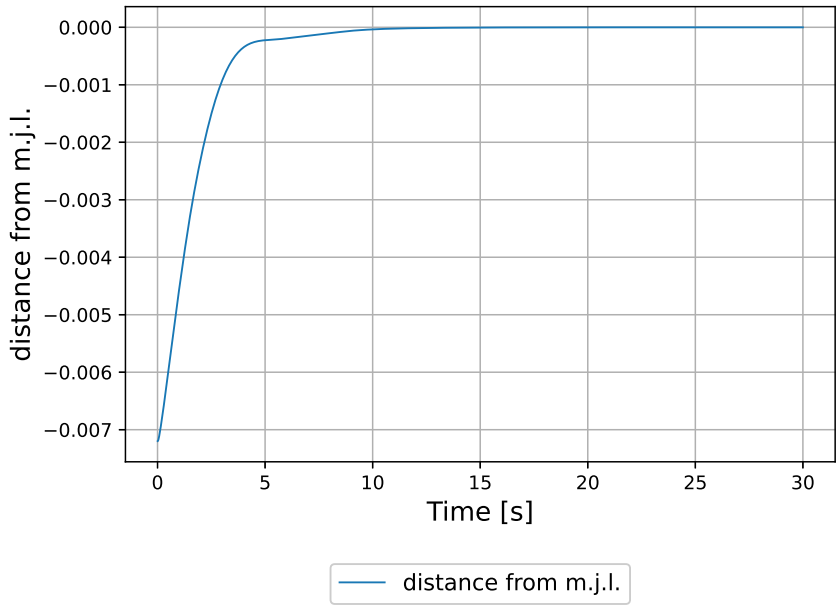

Figure 4.91: Distance from mechanical joint limits measure of the right arm.

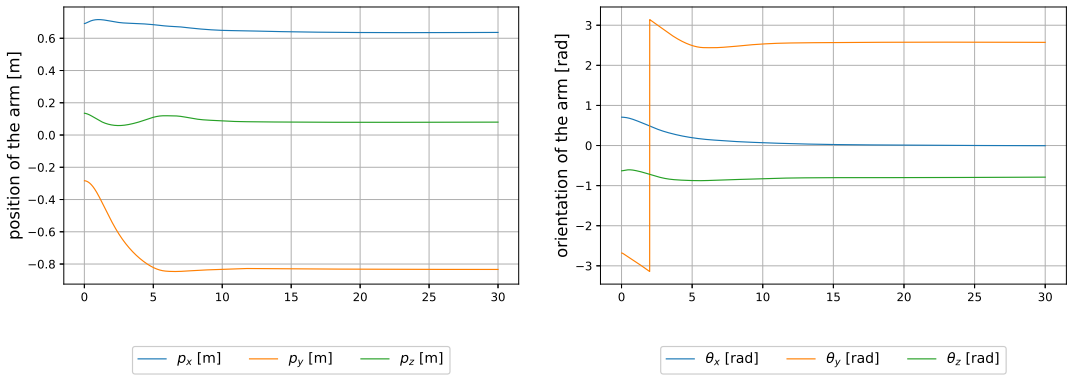

Figure 4.92: End-effector position (left) and orientation (right).

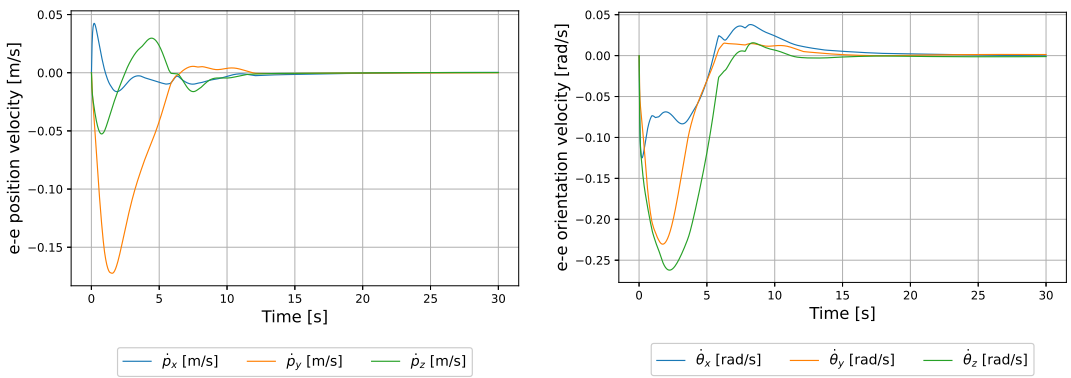

Figure 4.93: End-effector linear (left) and angular (right) velocities.

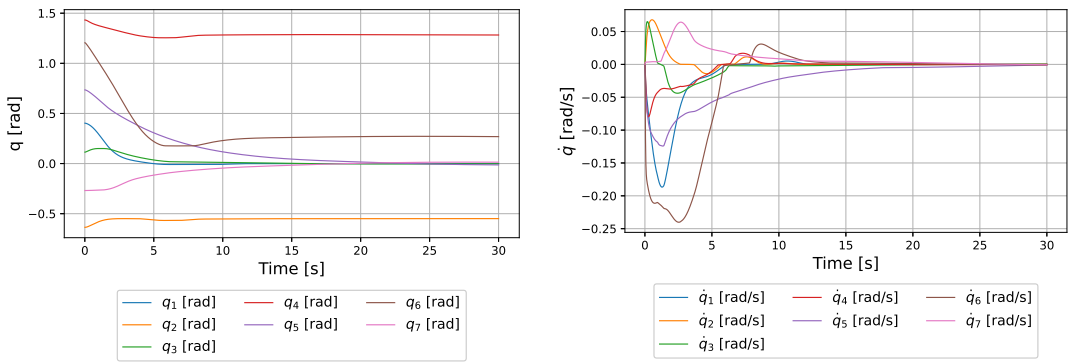

Figure 4.94: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

The robot reaches the final configuration for the arm

$$\boldsymbol{q} = [6.608{\cdot}10^{-4}, -5.491{\cdot}10^{-1}, -1.852{\cdot}10^{-3}, 1.282 - 1.343{\cdot}10^{-2}, 2.687{\cdot}10^{-1}, 1.387{\cdot}10^{-2}]$$

and the pose

$$\boldsymbol{p} = [0.636,\ -0.833,\ 0.079,\ -0.788,\ -0.004,\ 2.573]$$

The initial value of the distance from mechanical joint limits was $w_{m.j.l.} = -0.0071$, at the end the value is $w_{m.j.l.} = -9.864 \cdot 10^{-7} \simeq 0$.

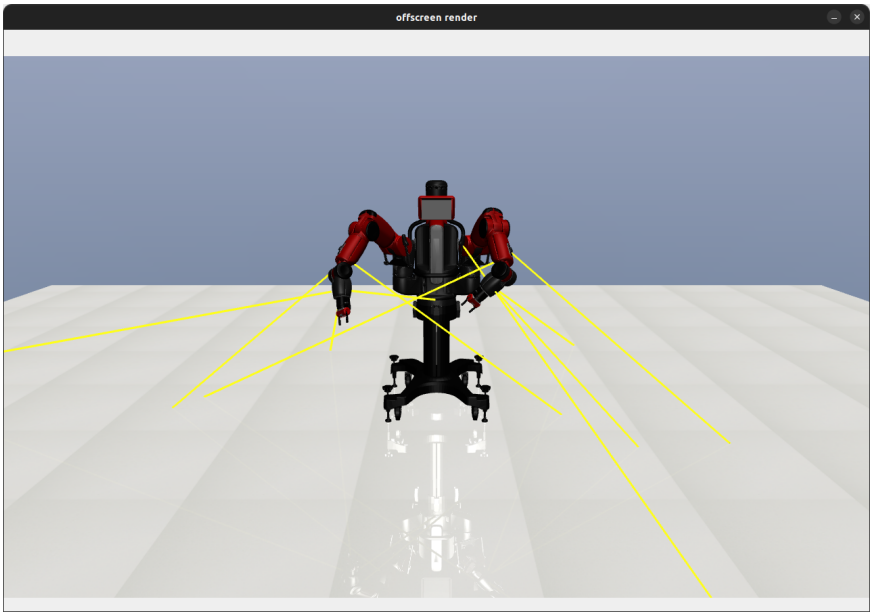

Figure 4.95: Maximization of the distances from mechanical joint limits with the right arm.

4.1.5 Useless distracting task: turn head

An extra and useless task was introduced to show robustness to useless tasks of the algorithm. In this case, the robots head plate is simply allowed to rotate, with the addition of a revolute joint. Here the performances of that task are reported, imposing on the joint a velocity of 0.1 rad/s.

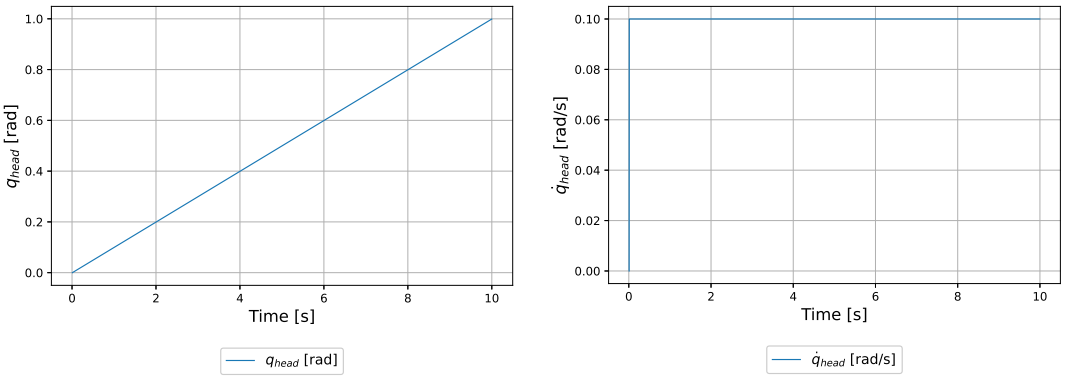

Figure 4.96: Position (left) and velocity (right) of the head joints.

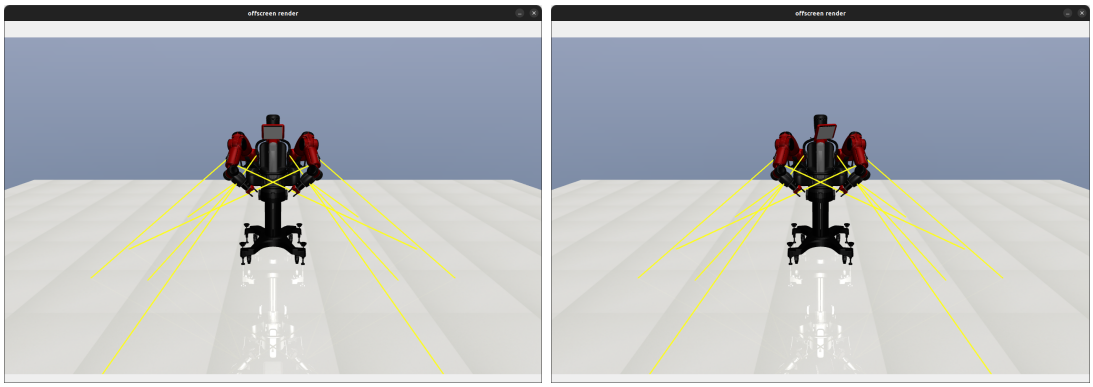

Figure 4.97: Baxter robot with straight head pan (left) and with rotating head pan (right).

4.2 Tasks combination

In this section, the combinations trough null space projector (section 2.1.6) of different tasks are presented. Both the order of the combinations and the parameters are chosen to show the effect on the robot, for the learned stacks part refer to the Genetic Programming chapter (chapter 5). The following results are all obtained using the right arm of the robot.

4.2.1 Inverse Kinematic & Maximization of Manipulability

The results are compared with Inverse Kinematic task alone and then with Maximization of Manipulability. The parameters are the same used in section 4.1 with the same initial configuration and target pose for the right slave arm. No obstacles are present in the simulation and the results are showed in an environment with kinematic only.

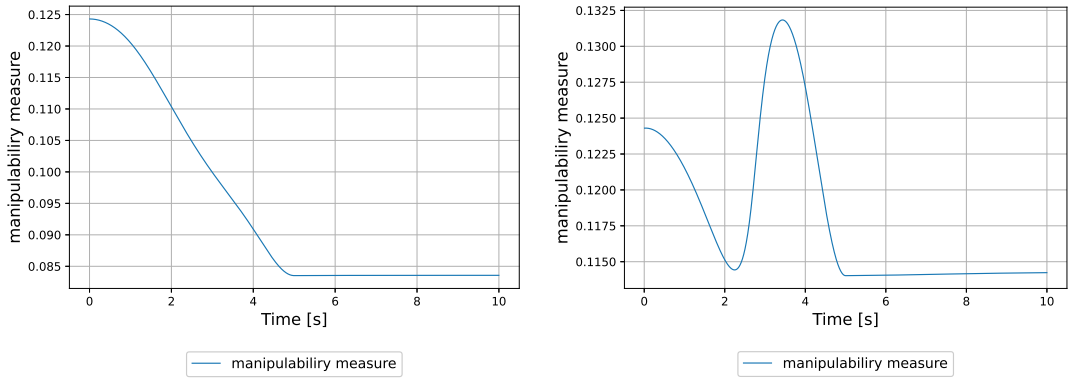

Figure 4.98: On the left the manipulability measure with the only Inverse Kinematic task, on the right the same measure with the combination of the two tasks.

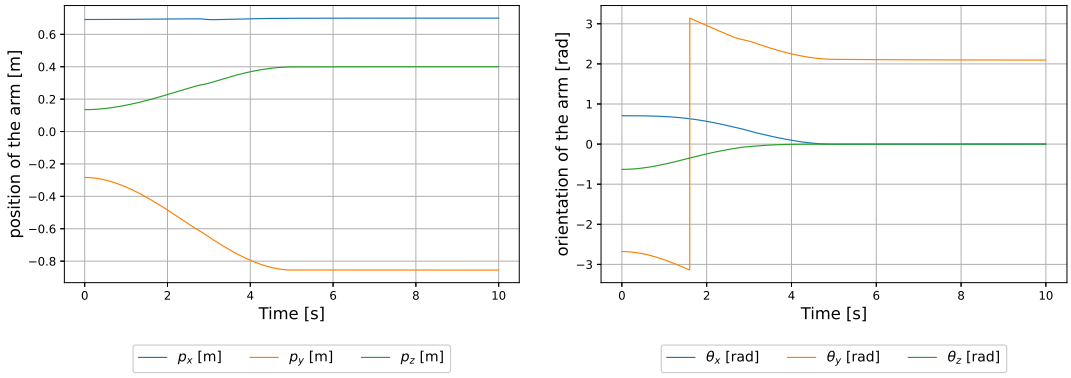

Figure 4.99: End-effector position (left) and orientation (right).

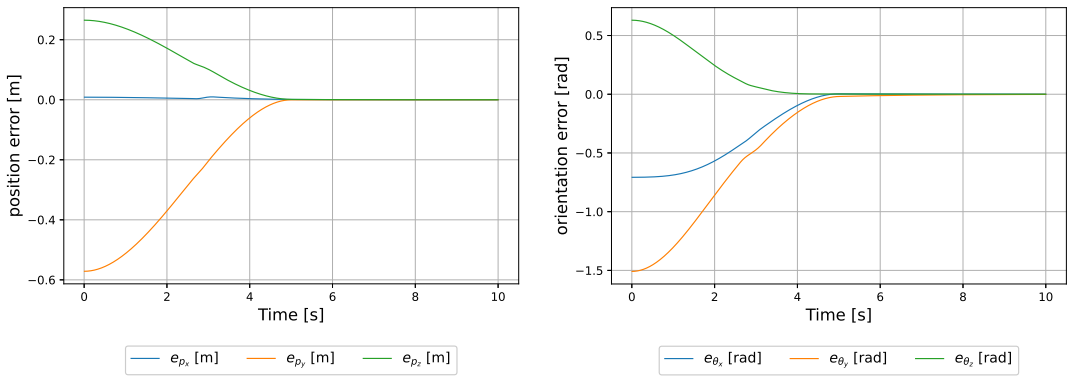

Figure 4.100: Position (left) and orientation (right) errors.

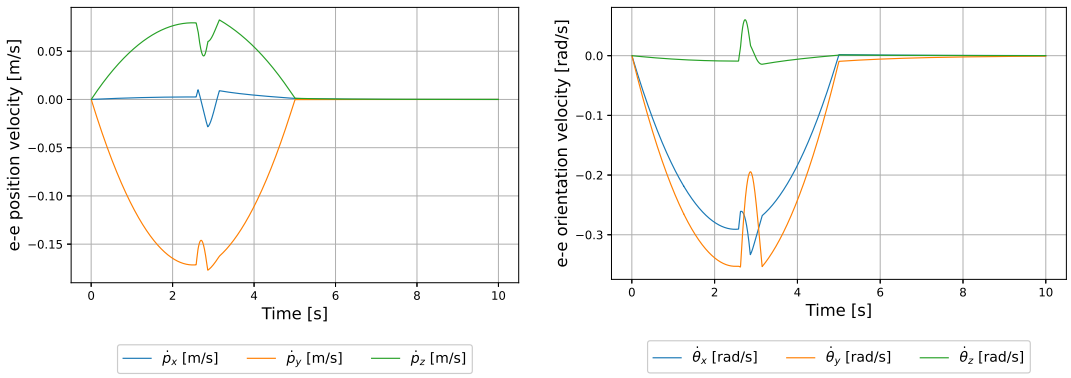

Figure 4.101: End-effector linear (left) and angular (right) velocities.

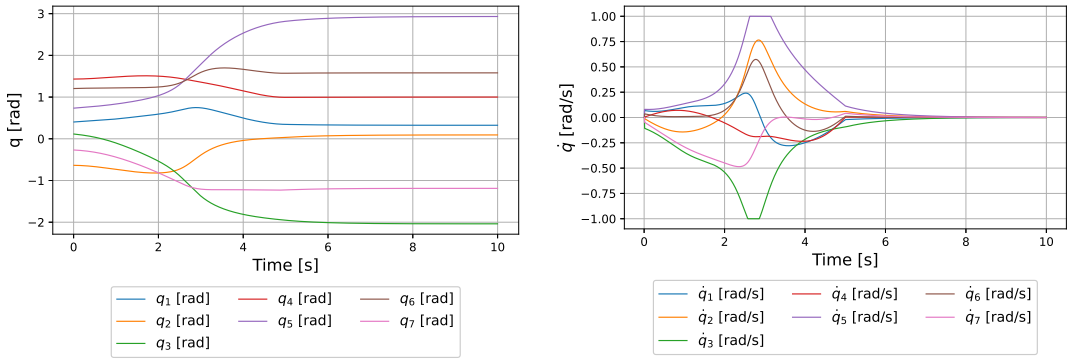

Figure 4.102: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

It is possible to observe from figure 4.98 that the manipulability measure is increased solving the same inverse kinematic problem with the arm. Looking at the plots in figure 4.100, it is possible to observe that the task primary task, the inverse kinematic, can be solved respecting the trajectory time and nullifying the

error of both position and orientation. Making a comparison with the plots in section 4.1.1, it is possible to observe that the velocities in joint configuration are different, with $\dot{q}_3$ and $\dot{q}_5$ which saturate for a period of time. Furthermore, null space combination should not modify the position and orientation of the end-effector of the main task. However it is possible to notice how the trajectory is slightly different in the same period of the saturations. Without a constraint on the maximum and minimum of velocities, the trajectory of the end-effector would be the same. At the end the final Manipualbility Measure value is $w_{manip.} = 0.114$, while without the prioritized combination and with Inverse Kinematic only is $w_{manip.} = 0.0836$. The final end-effector pose is the same of section 4.1.1, wile the final joint configuration is

$$\boldsymbol{q} = [0.324,\ 0.092,\ -2.038,\ 1.001,\ 2.932,\ 1.581,\ -1.188],$$

which is different from the one obtained in section 4.1.1 for Inverse Kinematic only. This happens because of the presence of a secondary task, that has an impact on the joint velocities.
Note that the initial manipulability measure was $w_{manip.} = 0.124$, bigger than the final one. In fact, the maximization of the measure has a secondary order of priority with respect to the Inverse Kinematic task.

4.2.2 Obstacle avoidance & Inverse Kinematic

Obstacle along the path

In a fist case case an obstacle was added along the path of the right slave arm. The objective of the robot is to reach the desired pose with its end-effector, without colliding with the obstacle and maintaining the maximum distance from it, namely $r_{k_{arm}}$. The parameters are the same used in section 4.1 for both the tasks. Clearly in this case the obstacle avoidance has an higher priority with respect to the inverse kinematic task and, the obstacles are placed is such a way that the robot collide if the prioritized order is inverted or only the Inverse Kinematic is executed.

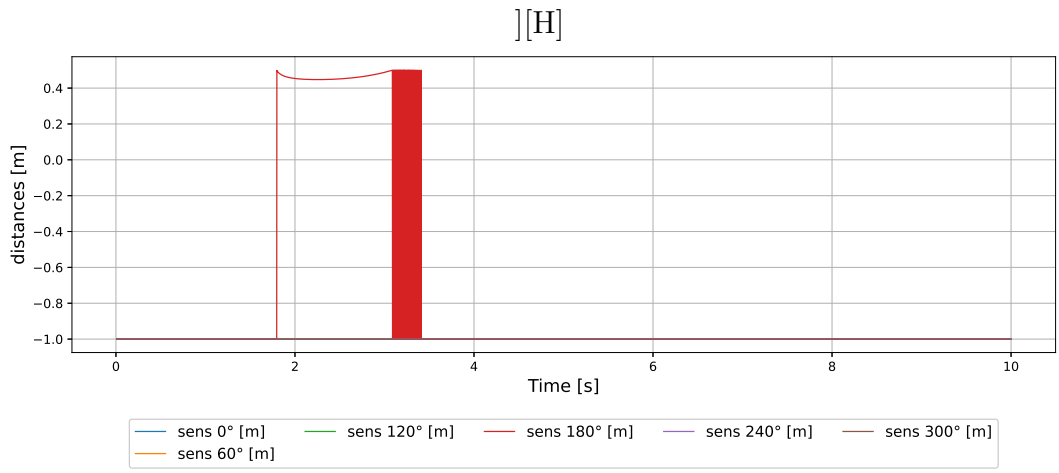

Figure 4.103: 3^{rd} site sensor distances.

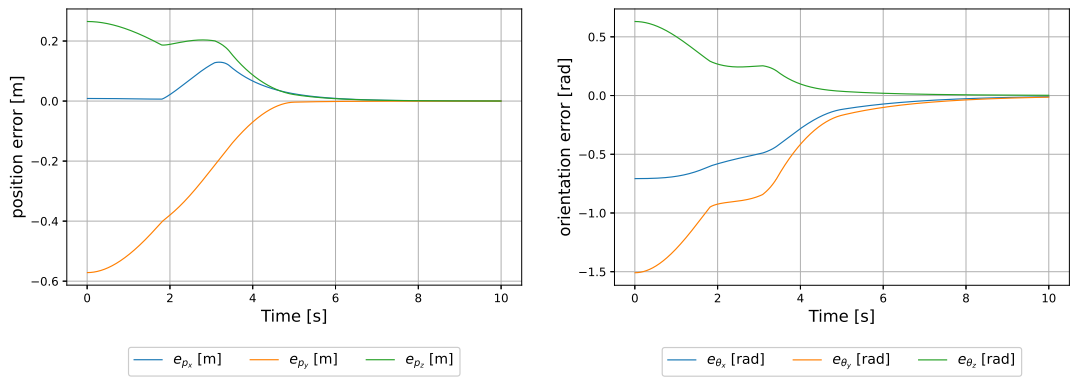

Figure 4.104: Position (left) and orientation (right) errors.

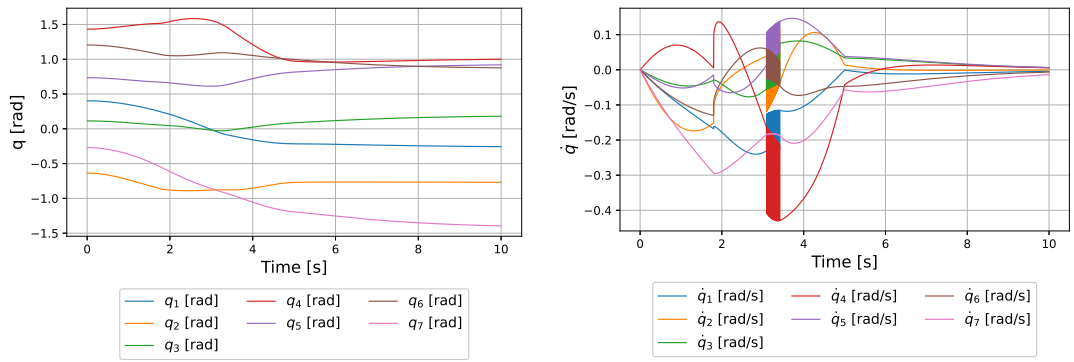

Figure 4.105: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

As it is possible to observe from the plots, the arm can reach the desired pose without colliding with the obstacles. However, the behaviour of the robot is delayed due to the fact that it maximizes its distance from the obstacle to r_k.

From plot in figure 4.103, it is possible to see how the obstacle is sensed at almost 2 s from the beginning of the simulation. The robot starts to move away from it and velocities change as it is possible to see from plot in figure 4.105. From this last plot, it is also possible to see how the chattering acts, imposing instantaneous velocities and not a smooth trajectory. The time in which this happens, coincides with the time in which in plot in figure 4.103 the distance oscillates between being sensed with distance r_k and not being sensed, changing the matrix $\boldsymbol{J}_o$ at each simulation step.
At the end of the process, the robot ends with configuration

$$\boldsymbol{q} = [-0.267,\ -0.703,\ 0.210,\ 1.044,\ 0.872,\ 0.825,\ 1.392].$$

Obstacle in the Inverse Kinematic position

As second case, an obstacle (a sphere of radius $r = 0.3\ m$) is placed in order to include, in its volume, the Inverse Kinematic target. This will prevent the arm to reach the desired pose but, on the other hand, it will show how the null space projection affects the execution of a duty of the robot. In particular it will prove the effectiveness of the obstacle avoidance task, also while the robot attempts to reach an unfeasible pose. In this example, the chattering avoidance between tasks is crucial to prevent high instantaneous velocities as possible. However, as it is possible to see from plots, chattering can not be avoided at all.
The chosen parameters are $\gamma = 1$ and $t_{trajectory} = 5\ s$ for the inverse kinematic task and $\gamma_o = 1$, $t_{trajectory} = 0.5\ s$ and $r_{k_{arm}} = s_k = 0.2\ m$ for the obstacle avoidance task. The target pose for the right end-effector is :

$$\boldsymbol{p} = [0.7,\ -0.855,\ 0.4,\ 0,\ \frac{2\pi}{3},\ 0].$$

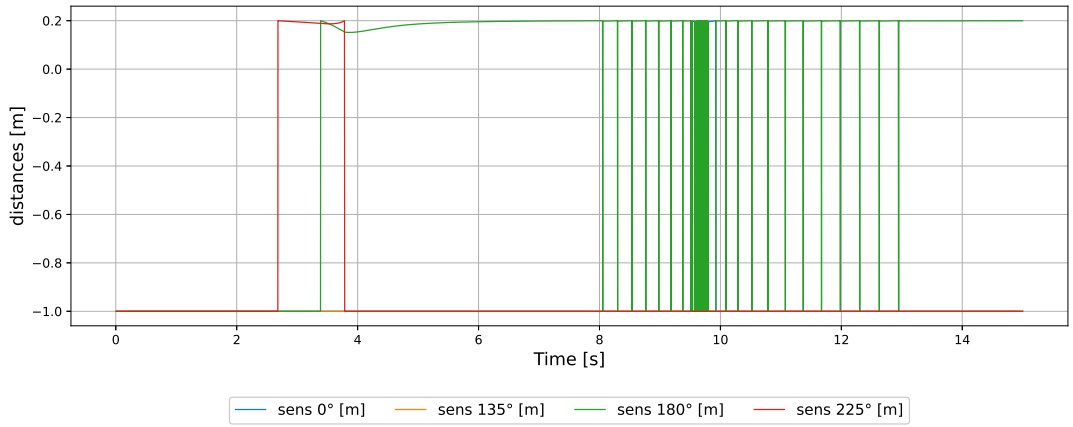

Figure 4.106: 2^{nd} site sensor distances.

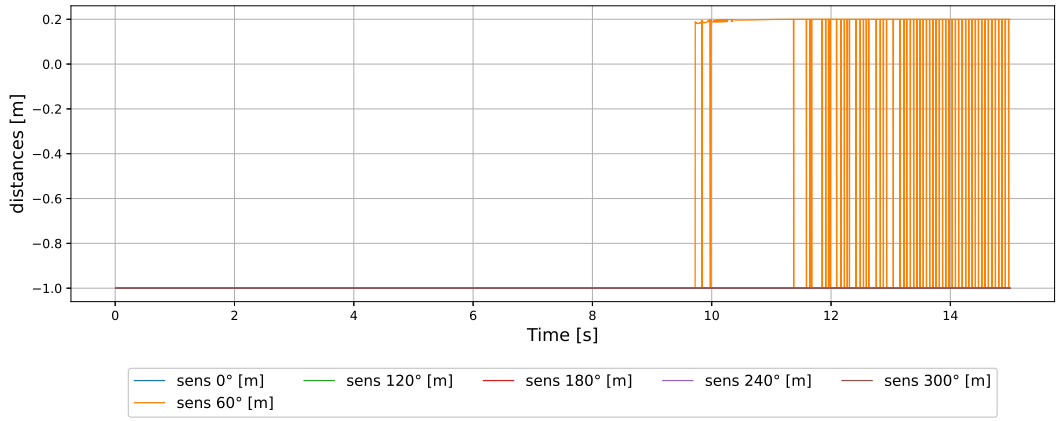

Figure 4.107: 3^{rd} site sensor distances.

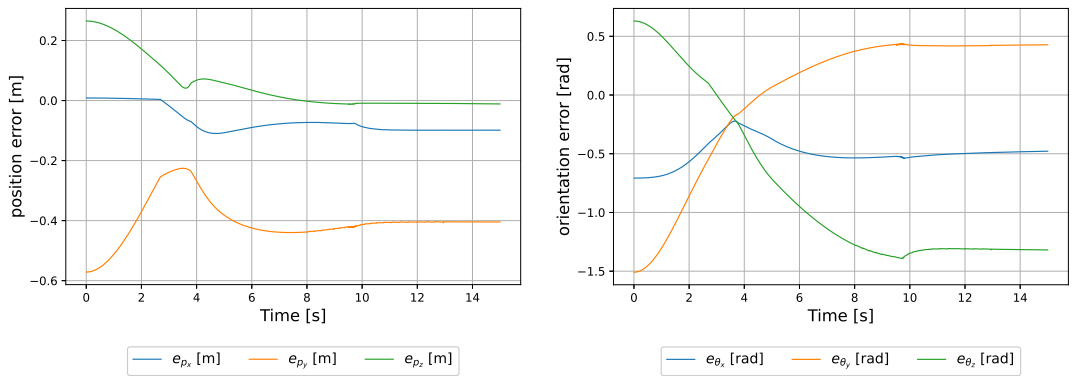

Figure 4.108: Position (left) and orientation (right) errors.

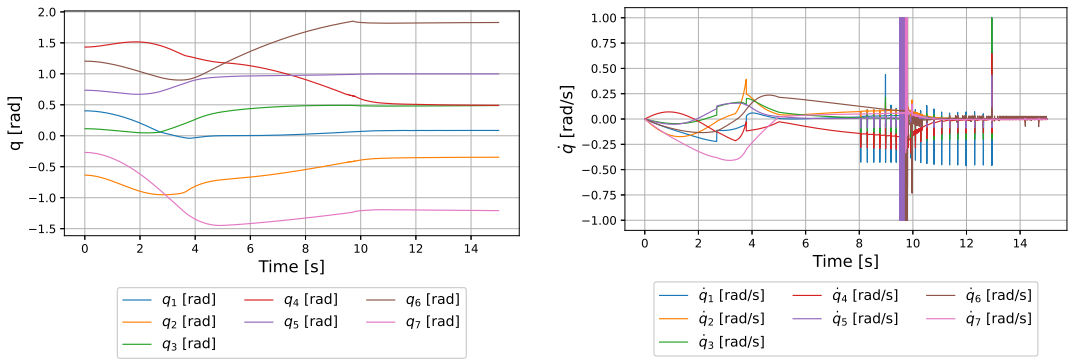

Figure 4.109: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

As it is possible to note, the end-effector can not reach the desired target pose, due to the fact that the obstacle is in the target position. The robot reaches a sort of equilibrium, maintaining the distance from the obstacle. However, as it is possible to observe from plot in figure 4.107, the distance oscillate between the value of s_k and a value slightly bigger, creating chattering and instantaneous velocities (figure 4.109).
The robot reaches the final configuration:

$$\boldsymbol{q} = [0.071,\ -0.392,\ 0.484,\ 0.595,\ 0.995,\ 1.829,\ -1.209]$$

and the final pose

$$\boldsymbol{p} = [0.787,\ -0.442,\ 0.409,\ 1.348,\ 0.530,\ 1.669],$$

which is not the desired one. Anyway, also if the pose is not the correct one, the robot avoid the collision with the obstacle, maintaining the distance and, in the defined priority order, this is the main objective.

4.2.3 Useless distracting task & Inverse Kinematic

In this section the objective is to prove that this task will prevent the robot to reach its goal. The combination is given first by the priority order $[n_2,\ n_5]$ and then $[n_5,\ n_2]$, where n_2 indicates the inverse kinematic task, while n_5 the useless task. The right arm, as slave, has to accomplish the same task of section 4.1.1.
In the first case, while the priority order is $[n_2,\ n_5]$, the result is the same of section 4.1.1 and the arm complete the task. Only the errors of the pose are reported to show the goodness of the combination.

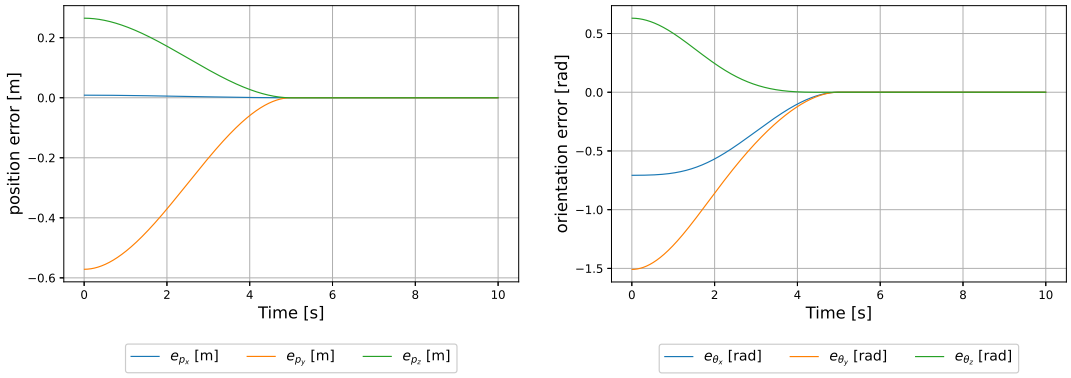

Figure 4.110: Position (left) and orientation (right) errors.

On the other hand, if the priority order is given by $[n_5,\ n_2]$, the robot can not reach the target pose since it is "distracted" from the useless task.

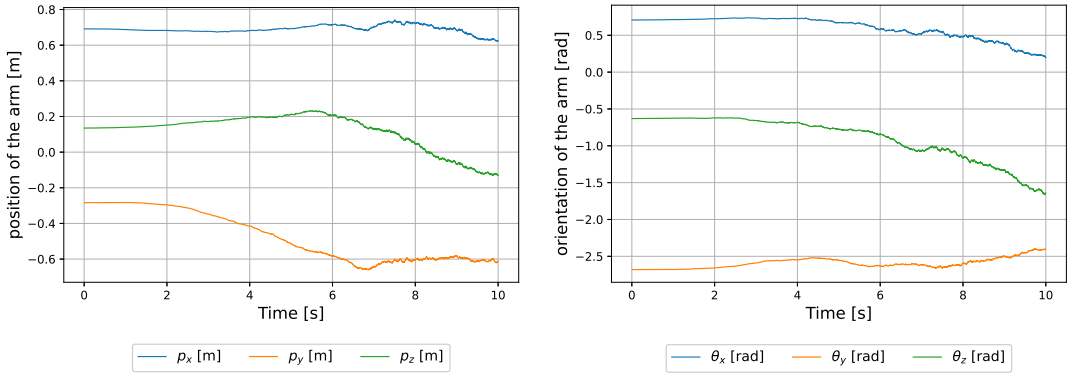

Figure 4.111: End-effector position (left) and orientation (right).

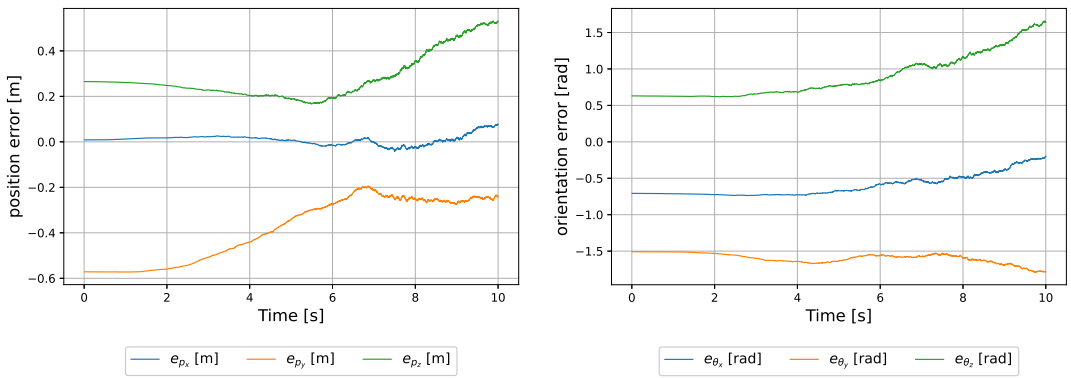

Figure 4.112: Position (left) and orientation (right) errors.

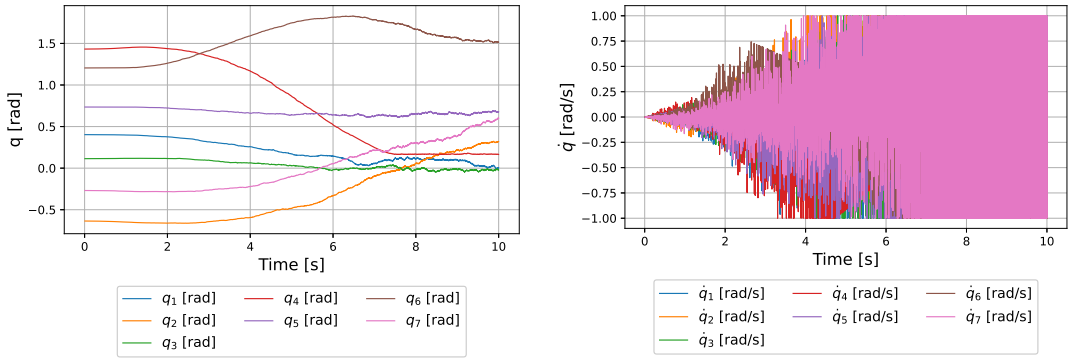

Figure 4.113: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

As it is possible to observe from the plots, the errors of position and orientation diverge and there are instantaneous velocities. This happens due to the choice of the Jacobian of the useless task, which is a random matrix. In the null space combination, the $\boldsymbol{N}$ matrix (equation 2.49) is affected by the randomness of the Jacobian. This, allows the robot to obtain a certain amount of freedom of movement at each iteration, which ca :

$$\dot{\boldsymbol{q}} = \mathbf{o}_7 + \boldsymbol{N}\dot{\boldsymbol{q}}_{I.K.}.$$

If the Jacobian was chosen such that $\boldsymbol{N} = 0$, then $\dot{\boldsymbol{q}} = 0$ and the robot would be motionless.

4.2.4 All tasks

In this section, all the tasks together are tried. The priority order of the combination is given by $[n_1, n_2, n_3, n_4]$, where the labels correspond to the one reported in section 3.7.
The parameters are selected as:

- $\gamma_o = 1$, $t_o = 5\ s$, $r_{k_{base}} = r_{k_{arm}} = 0.5\ m$ and $s_k = 0.2\ m$ for the obstacle avoidance task
- $\gamma = 1$ and $t = 5\ s$ for the inverse kinematic task
- $k_0 = 100$ for the maximization of the manipulabilty task
- $k_0 = 100$ for the maximization of distances from m.j.l. task.

Then, the right arm in slave configuration have to reach the same target pose starting from same condition of section 4.2.2 in an environment with kinematic only.

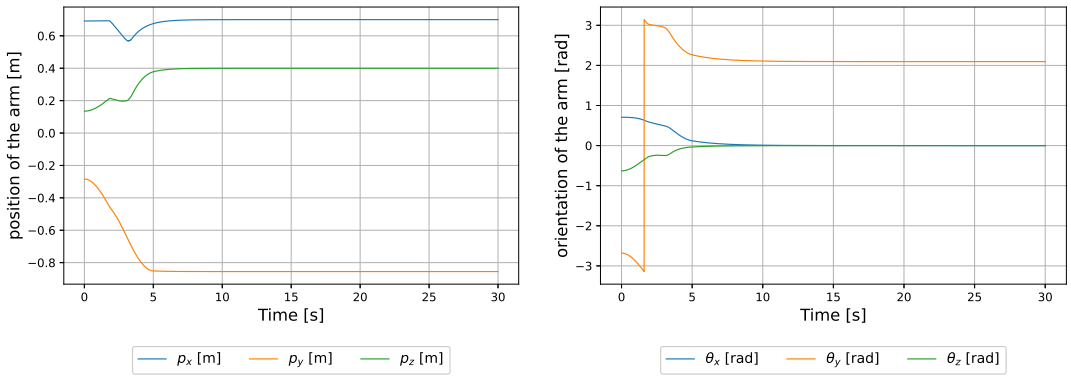

Figure 4.114: End-effector position (left) and orientation (right).

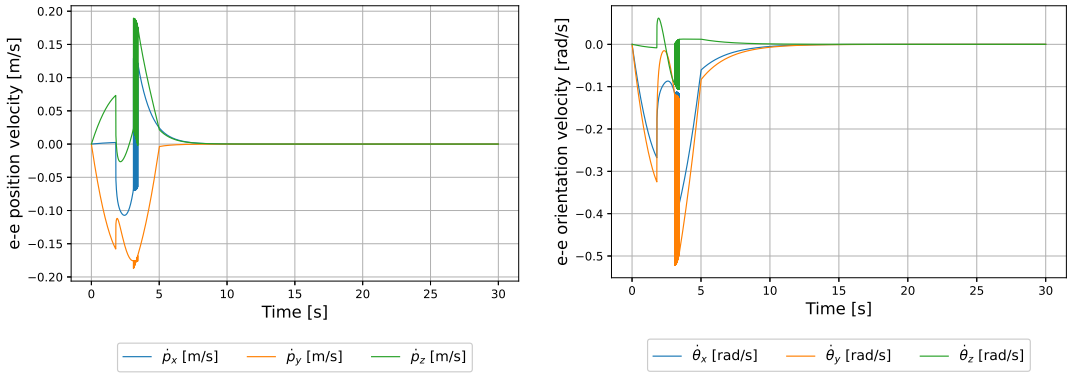

Figure 4.115: End-effector linear (left) and angular (right) velocities.

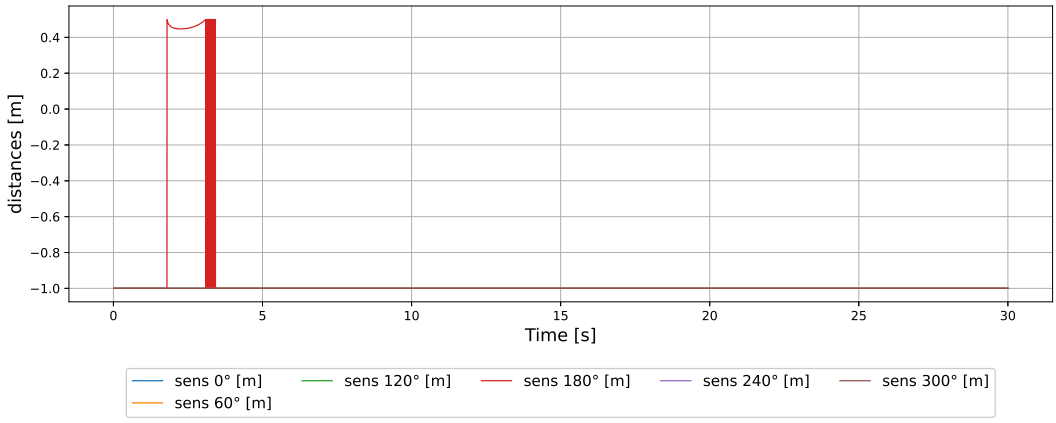

Figure 4.116: 3^{rd} site sensor distances.

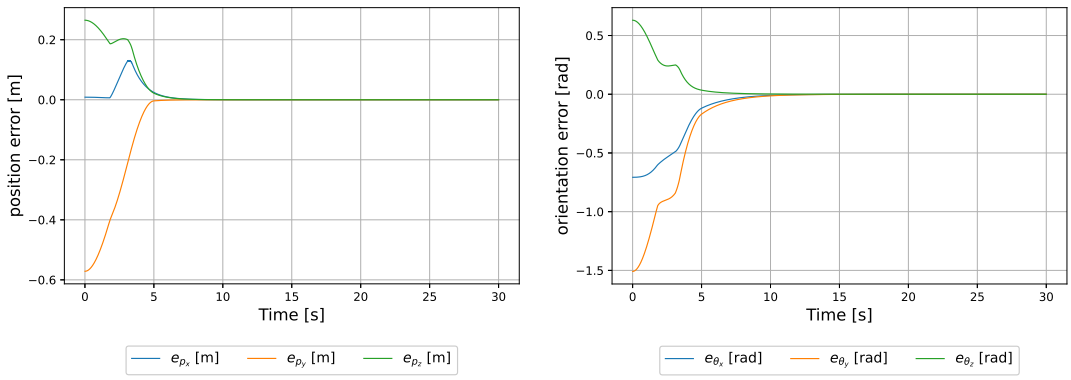

Figure 4.117: Position (left) and orientation (right) errors.

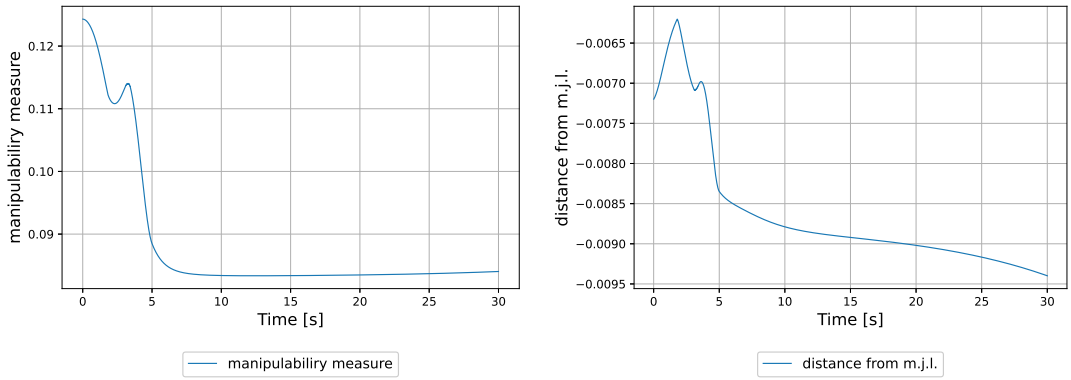

Figure 4.118: Manipulability measure (left) and distance from mechanical joint limits measure (right)

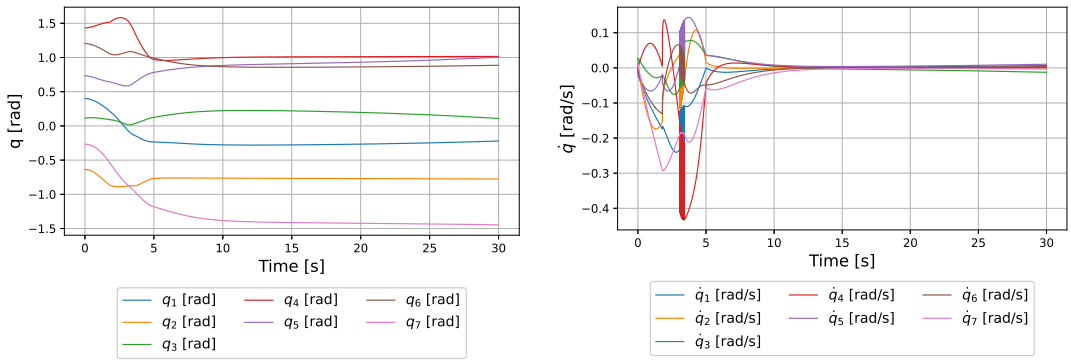

Figure 4.119: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

In this case, that combines all the task in the same simulation, several things can be observed. First of all, the robot can reach the final desired pose, without colliding with the obstacle and keeping distances from it. The target pose, however, can not be reached in the specified trajectory time because of the object

positioned along the path.
Once the robot is in the desired pose with its end-effector, the arm starts to move in order to maximize the manipulability of the arm. In fact, the velocities $\dot{\boldsymbol{q}}$ are not null, and from plots it is possible to observe the manipulability increasing while distance from mechanical joint limits is decreasing, since it has a lower priority order. This means that in this case, there are not enough degrees of freedom to solve all the task in the same simulation. Then looking at the plots which refer to the end-effector, once it has reached the desired pose it stays fixed. Even if the arm is moving, confirming that the null space projector would not modify the end-effector velocities.

4.2.5 Weighted/Non-weighted combination of the tasks with null space projector

One of the objective of this work, was to introduce a convex combination of tasks via null space projection with prioritized order. So that, each task should have a wight w_i such that $\sum_{i=1}^{n} w_i = 1$, where n is the number of tasks in the stack. These weights, would have been multiplied with the result of the null space projection of the corresponding task like

$$\dot{q} = w_a \dot{q}_a + N_a w_b \dot{q}_b + w_c N_{ab} \dot{q}_c + w_d N_{abc} \dot{q}_d. \tag{4.1}$$

Unfortunately this solution turns out to be unfeasible. In fact, the scaling with a weight of the velocities once they are already transformed in the joint space, modify the structure of the solution preventing the robot to reach its goal. This leads to a wrong solution of the task that can not be predicted in advance.
The choice, after the problem arises, was to simply take off weights. In fact the null space projector already defines a priority order, and no weighting of the tasks is needed.
So that, the final combination trough null space projection is simply given by

$$\dot{q} = \dot{q}_a + N_a \dot{q}_b + N_{ab} \dot{q}_c + N_{abc} \dot{q}_d. \tag{4.2}$$

Despite the result achieved is not correct, it is interesting to observe the effect of the addition of a weight in front of the velocities in joint space.
However, this holds for velocities obtained with equations 2.20 and 2.37 obtained with a matrix multiplication with the Jacobian. Regarding velocities obtained

with equation 2.44, the multiplication of a weight simply scale the gradient making it lower. This leads to a delayed solution of the task but still correct. This result hold also in the case in which the robot simply solves a single task without the null space projector.

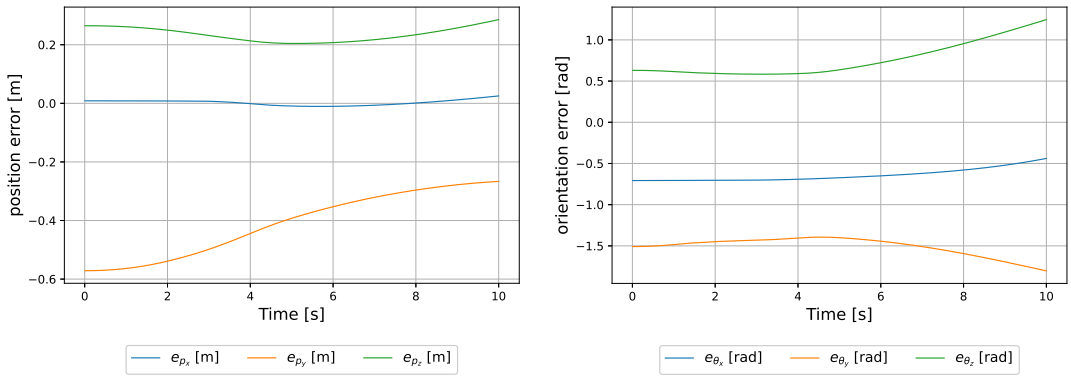

Figure 4.120: Example of position (left) and orientation (right) errors of a weighted task.

The case of the error pose reported in figure 4.120 is the same of section 4.1.1, but the velocity is weighted by $w = 0.1$ like $\dot{q} = 0.1 \cdot \dot{q}_{I.K.}$ As it is possible to observe from the plot, the arm do not converge toward the desired pose but, it diverges. This prevents the robot to accomplish its task.

Chapter 5

Genetic programming: simulations and results

In this chapter, the results obtained during the learning phase are presented. All the simulations, in this chapter, are conducted in an environment without dynamics and with kinematics only. This will increase the efficiency of the learning process, making the simulations lighter, without compromising the final result.

5.1 Initialization and Genetic Operations

First of all, some examples of the behaviour of the genetic programming algorithm are showed, in order to print out its functioning. An *initial population* will be created and, after that, the genetic operations will be conduced on the stacks.
In this case the stacks are not executed and their cost will always be -1. In cases like that, in which two or more algorithms have the same cost during the genetic selection phase, one is chosen randomly between the two with equal probability.
In this first phase the order of the stack is randomly selected.
This examples, will also provide a clearer overview of the representation of stacks in this work (section 2.2.2). In fact, it is possible to observe how they are built as lists in Python programming language. Then, each list, contains the cost in first position and another list for each specific task, with all the information necessary to carry it out the simulation.

Initialization

Following, an example of a new initialized population of 10 individuals is reported:

- $\mathbf{1}^{st}$ stack: [[-1, 'n2', True, [0.4377713247475583, 6.521411444358634]], 'n4', True, [50.00909144224303]], 'n3', False, [56.421085714393506]], 'n1', True, [0.25835470924121684, 0.24470071750461747, 0.19660605013514482, 1.2459840375539286, 0.8877340468345674]]]

- $\mathbf{2}^{nd}$ stack: [-1, 'n1', True, [0.26403069227677545, 0.1908338455487012, 0.16420314839027117, 0.5917863860534613, 8.521208744722797]], 'n4', True, [78.52002002748539]], 'n3', True, [8.597694924717548]], 'n2', False , [1.3619810520123647, 7.168015466457151]]]

- $\mathbf{3}^{rd}$ stack: [-1, 'n4', True, [47.10999991854254]], 'n3', True, [93.14232630274813]], 'n2', True, [0.3333698436407364, 2.867010799285742]], 'n1', True , [0.6537029304865376, 0.5455380810746031, 0.1793292369376181, 1.4802895349743042, 4.591419638659162]]]

- $\mathbf{4}^{th}$ stack: [-1, 'n2', True, [1.2571116111531975, 4.693018504655625]], 'n4', True, [15.180605683240477]], 'n3', True, [53.68055662600112]], 'n1', True, [0.47982203469839146, 0.430663748054481, 0.26316973328512744, 1.594759204620307, 1.4545608701340917]]]

- $\mathbf{5}^{th}$ stack: [-1, 'n3', False, [43.20689379085039]], 'n1', True, [0.5308147271544323, 0.23119341884484415, 0.14683128553032876, 0.05440570282885515, 5.53081815779331]], 'n2', True, [1.411818903537741, 0.2680903097583587]], 'n4', True, [30.38956576585312]]]

- $\mathbf{6}^{th}$ stack: [-1, 'n2', True, [0.7698344633722882, 5.317272612659643]], 'n4', True, [78.18596159177291]], 'n1', True, [0.5832866180394054, 0.4804730748782273, 0.20407729261852686, 0.6185014908706934, 5.4982000531614545]], 'n3', True, [23.734895914374444]]]

- $\mathbf{7}^{th}$ stack: [-1, 'n2', True, [1.4660259721220972, 6.0421345737932715]], 'n3', True, [91.21953645994083]], 'n4', True, [64.40584293882264]], 'n1', True, [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]]

- $\mathbf{8}^{th}$ stack: [-1, 'n2', True, [0.4662287689639555, 8.481043602536278]], 'n3', True, [66.25061268553638]], 'n1', False, [0.90794984265897, 0.35307503428243947, 0.2555955959937183, 0.019232730536018705, 5.624467556960831]], 'n4', True, [37.06160559816853]]]

- $\mathbf{9}^{th}$ stack: [-1, ['n1', True, [0.6012094714952207, 0.24535103413632867, 0.14525220096750563, 1.1604918455059652, 5.773392667136015]], ['n3', True, [46.473248533818676]], ['n2', True, [1.8266519451799703, 0.5945901348604077]], ['n4', True, [54.65815744955243]]]

- $\mathbf{10}^{th}$ stack: [-1, ['n2', True, [0.9147172126995893, 9.244523804605869]] , ['n3', True, [14.297853669217108]], ['n1', True, [0.947887902163306, 0.9161438894951406, 0.6193458091093882, 1.9579185107021035, 9.277438338142053]], ['n4', True, [47.737015624086176]]]

Some examples of *genetic operations* are following reported. It is possible to observe how, the stacks are all different each other, with different orders and parameters.
Remember that, the genetic selection is made pairing the stacks and selecting as survivor the one with the lowest cost in general, but since all the stacks have $cost = -1$ one of the two is randomly selected. The other one will be lost forever.

Crossover

In this section some examples of *crossover* are reported.
From the two stacks:

> [-1, ['**n2**', True, [0.4377713247475583, 6.521411444358634]], ['n4', True , [50.00909144224303]], ['**n3**', False, [56.421085714393506]], ['n1', True , [0.25835470924121684, 0.24470071750461747, 0.19660605013514482, 1.2459840375539286, 0.8877340468345674]]],

> [-1, ['n2', True, [1.4660259721220972, 6.042134573793271 5]], ['n3', True, [91.21953645994083]], ['**n4**', True, [64.40584293882264]], ['**n1**', True , [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]],

it is possible to obtain the following offspring from their combination, using the priority order of the first one, which is randomly selected among the two:

> [-1, ['**n2**', True, [0.4377713247475583, 6.521411444358634]], ['**n4**', True, [64.40584293882264]], ['**n3**', False, [56.421085714393506]], ['**n1**', True , [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]].

Instead, from the two parents stacks:

[-1, ['**n4**', True, [47.10999991854254]], ['n3', True, [93.14232630274813]], ['n2', True, [0.3333698436407364, 2.867010799285742]], ['n1', True , [0.6537029304865376, 0.5455380810746031, 0.1793292369376181, 1.4802895349743042, 4.591419638659162]]],

[-1, ['**n1**', True, [0.6012094714952207, 0.24535103413632867, 0.14525220096750563, 1.1604918455059652, 5.773392667136015]], ['**n3**', True, [46.473248533818676]], ['**n2**', True, [1.8266519451799703, 0.5945901348604077]], ['n4', True , [54.65815744955243]]],

it is possible to obtain the following offspring from the combination of the two, using a new randomly selected priority order:

[-1, ['**n2**', True, [1.8266519451799703, 0.5945901348604077]], ['**n1**', True, [0.6012094714952207, 0.24535103413632867, 0.14525220096750563, 1.1604918455059652, 5.773392667136015]], ['**n4**', True, [47.10999991854254]] ,['**n3**', True, [46.473248533818676]]].

Clearly, if the priority order is already fixed, the crossover will combine different stacks without the opportunity to randomly change the order of the offspring. In this way, the priority order will always be the same.

Mutation

In this section some examples of *mutation* are reported.
From the parent stack:

[-1, ['n4', True, [47.10999991854254]], ['n3', True, [93.14232630274813]], ['n2', True, [0.3333698436407364, 2.867010799285742]], ['n1', **True** , [0.6537029304865376, 0.5455380810746031, 0.1793292369376181, 1.4802895349743042, 4.591419638659162]]],

it is possible to obtain trough mutation, activation/deactivation of a task, the following offspring:

[-1, ['n4', True, [47.10999991854254]], ['n3', True, [93.14232630274813]], ['n2', True, [0.3333698436407364, 2.867010799285742]], ['n1', **False** , [0.6537029304865376, 0.5455380810746031, 0.1793292369376181, 1.4802895349743042, 4.591419638659162]]].

From the stack

[-1, ['n2', True, [**1.4660259721220972**, 6.0421345737932715]], ['n3', True, [91.21953645994083]], ['n4', True, [64.40584293882264]], ['n1'
, False, [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]],

it is possible to obtain through the change of a parameter a new offspring stack:

[-1, ['n2', True, [**0.6537029304865376**, 6.0421345737932715]], ['n3', True, [91.21953645994083]], ['n4', True, [64.40584293882264]], ['n1'
, False, [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]].

From the stack

[-1, ['n2', True, [1.4660259721220972, 6.0421345737932715]], ['**n3**', True, [91.21953645994083]], ['**n4**', True, [64.40584293882264]], ['n1', True
, [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]],

it is possible to obtain trough a swap of priority order

[-1, ['n2', True, [1.4660259721220972, 6.0421345737932715]], ['**n4**', True, [64.40584293882264]],['**n3**', True, [91.21953645994083]], ['n1', True
, [0.7698789180277057, 0.6693923027427455, 0.5512658023268667, 1.3693159425961379, 8.886790343317696]]].

Clearly, if the priority order is already fixed, the mutation can not change it.

5.2 Best Prioritized Order of the Stack of Tasks

In the first phase, it is important to derive the *best possible priority order for the stack* of tasks trough experience, thanks to simulations. The order of the stack can be randomly initialized in the first generation of stacks and the order of the offspring can be changed as explained in section 3.9 and showed in section 5.1.
In order to achieve the best prioritized order, many simulations in different environments and cases were conduced. Furthermore, several cost functions were considered.
However it is possible to notice that the Obstacle Avoidance task has a crucial role also in the case in which the distances from obstacles are not considered in the cost. On the other hand, in the case in which user wants to move the robot

towards a certain pose to perform a duty, the precision (and then the Inverse Kinematic tasks) plays a crucial role in the prioritized order. Differently, the remaining tasks are not crucial and may intervene in other ways in the same duty depending on the specific cost function.
It is important to remember that the initial robot's arms configuration is randomly initialized at the beginning of each simulation during the learning phase. Nevertheless, the plots reported are obtained testing the best algorithms starting the robot with base configuration

$$\boldsymbol{q}_{base} = [0,\ 0,\ 0]$$

and arms configuration

$$\boldsymbol{q}_{arm} = [\pm 0.403,\ -0.636,\ \pm 0.114,\ 1.432,\ \pm 0.735,\ 1.205,\ \mp 0.269]$$

for right and left arm in joint space.
In this first phase, while learning the best possible order for the stack, the parameters are fixed and the tasks are always active. Since this, during mutation and crossover, only the order of the tasks can change. This helps in finding the widest number of possible combinations. Following, the fixed parameters list for each task is reported:

- **Inverse Kinematic**: [2, 5]
- **Collision Avoidance**: [1, 0.5, 0.1, 2, 0.1]
- **Maximization of the Manipulability measure**: [100]
- **Maximization of distances from Mechanical Joint Limits**: [100].

5.2.1 Priority of Obstacle Avoidance

First of all, the order of the *Obstacle Avoidance task* is fixed. In fact, if one or more obstacles are present into the environment it is crucial to avoid them, expect in particular lucky cases in which the robot avoids them thanks to a particular combination of velocities. Clearly, if no obstacles are present in the environment the priority position of this task is not irrelevant, since it is possible for the robot to collide also with itself.
So that, at the end of the genetic process all the stacks have the task n_1 in the first position, independently from the designed cost function by the user. Otherwise,

the stacks will have a really high cost in most of the cases, which means that a collision occurs.
As an example, the arm of the robot has to reach the target pose $\boldsymbol{p}_d = [0.1, -0.9, -0.1, 0, \frac{4\pi}{3}, 0]$ and an obstacle is placed close to the target position. At the end of 10 generations, all the stacks have almost the same configuration and the best result is:

> [[0.13358175950958473, 'n1', True, [1, 0.5, 0.1, 2, 0.1]], ['n2', True, [2, 5]], ['n3', True, [100]], ['n4', True, [100]]]

(in which the cost function is a weighted combination given by $cost = ||error_{pose}||^2$). On the other hand, from a mutation was obtained

> [[10^{10}, 'n2', True, [2, 5]], ['n1', True, [1, 0.5, 0.1, 2, 0.1]], ['n3', True, [100]], ['n4', True, [100]]],

where the cost is too high since the arm collided with the obstacle. It is possible to note from this example that, even if the distances which are related to the obstacle avoidance task are not included in the cost function, it is crucial that this task is in first priority position to prevent collisions.

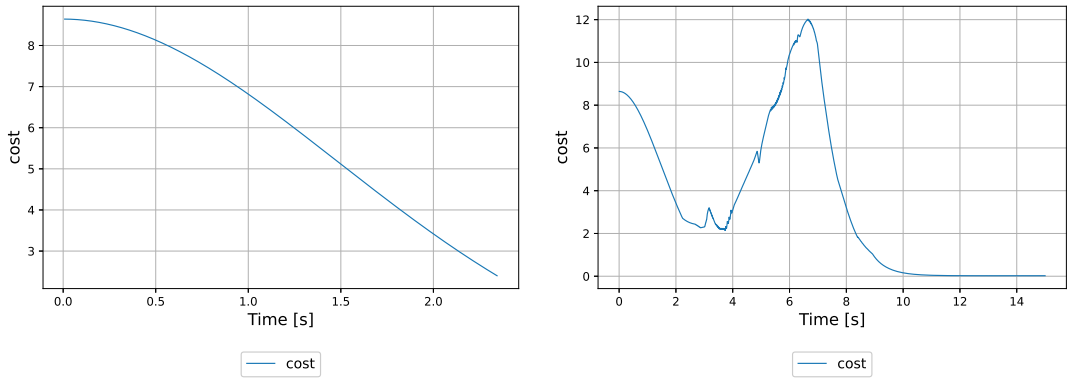

Figure 5.1: Example of the cost evolution during the execution of a duty. On the left the Obstacle avoidance task is not in first position, on the right it is. The cost function is given by: $cost = ||error_{pose}||^2$.

As it is possible to note from the plots in figure 5.1 the simulation is stopped before the arm can reach the desired pose with its end-effector. This because the arm collided with an obstacle and, since this, cost 10^{10} is assigned to the stack once the simulation is stopped.

5.2.2 Priority of Inverse Kinematic

Once the first position in the priority order is assigned, it is possible to proceed with the second one.
With the current tasks, it is mandatory to choose the *Inverse Kinematic* one.
Simulating for 10 generations, fixing n_1 with higher priority the Inverse Kinematic always has second priority position if the precision is included in the cost function. Otherwise the robot is not able to reach the desired pose. Differently from the Obstacle Avoidance task, if the measure of this task is not included in the cost, there will be no consequences on the achievement of the final goal.
This result is given by the fact that, if one of the other two remaining tasks come first than the Inverse Kinematic, the robot end-effector is prevented from reaching the desired pose, since for all the three tasks the same Jacobian is used. This result can be theoretically proved. Starting from the null space equations 2.49 and 2.19, it is possible to write

$$\big(\boldsymbol{I}_n - \boldsymbol{J}^{\dagger}\boldsymbol{J}\big)\boldsymbol{J}^{\dagger}\boldsymbol{v}_{IK} = \boldsymbol{J}^{\dagger} - \boldsymbol{J}^{\dagger} = \mathbf{o}_n. \tag{5.1}$$

Since this, the effect of the task on the arm will be null, preventing the achievement of the tasks target. The same will not happen if maximization of the Manipulability or distance from Mechanical Joint Limits tasks have a lower priority with respect to Inverse Kinematic since the velocities given by equations 2.43 and 2.46 are given by gradients and do not imply the usage of the pseudo-inverse Jacobian.
This result is confirmed through the learning simulation of the robot, as said at the beginning of the section, obtaining the same result without including this a priori knowledge.
Following an example of this behaviour is reported in the empty environment, with a simplified stack and $cost = ||\boldsymbol{e}_{pose}||$.

Stack [-1, ['$\boldsymbol{n2}$', True, [1, 5]], ['$\boldsymbol{n3}$', True, [100]]]:

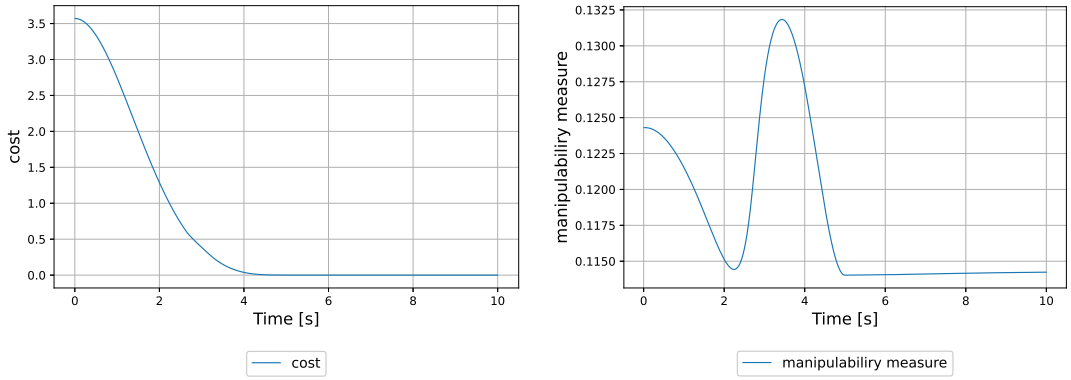

Figure 5.2: Cost (left) and manipulability measure (right) evolution during the simulation time.

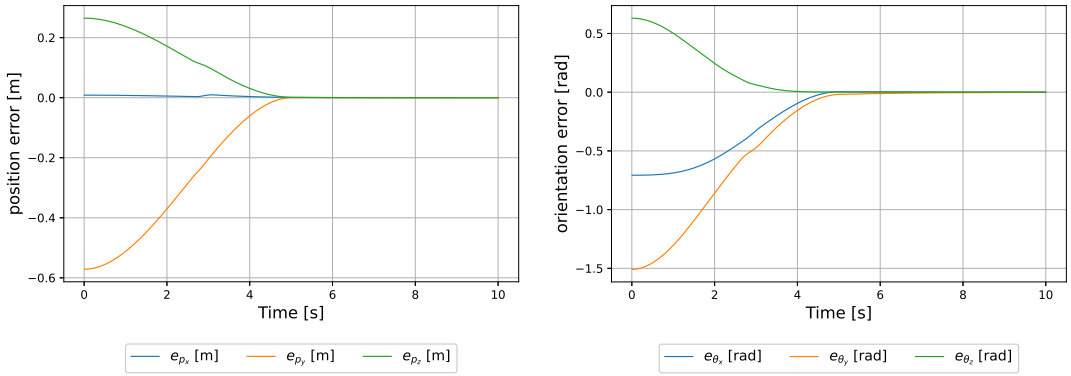

Figure 5.3: Position (left) and orientation (right) errors.

Stack [-1, ['***n3***', True, [100]], ['***n2***', True, [1, 5]]]:

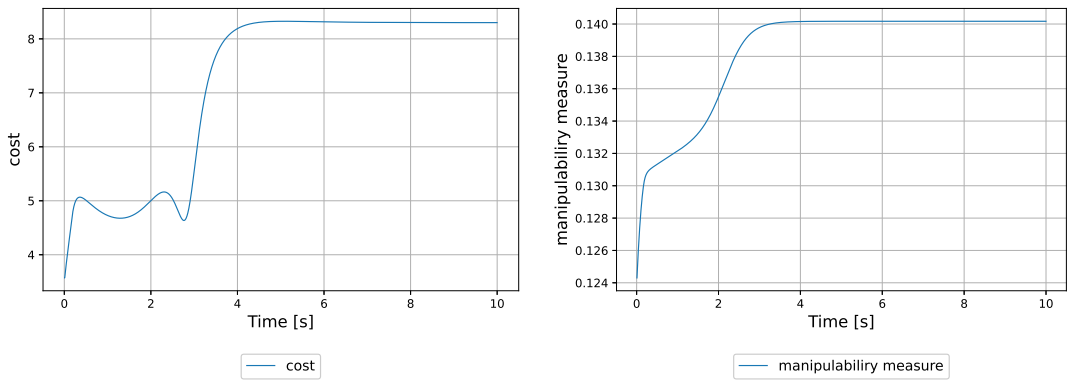

Figure 5.4: Cost (left) and manipulability measure (right) evolution during the simulation time.

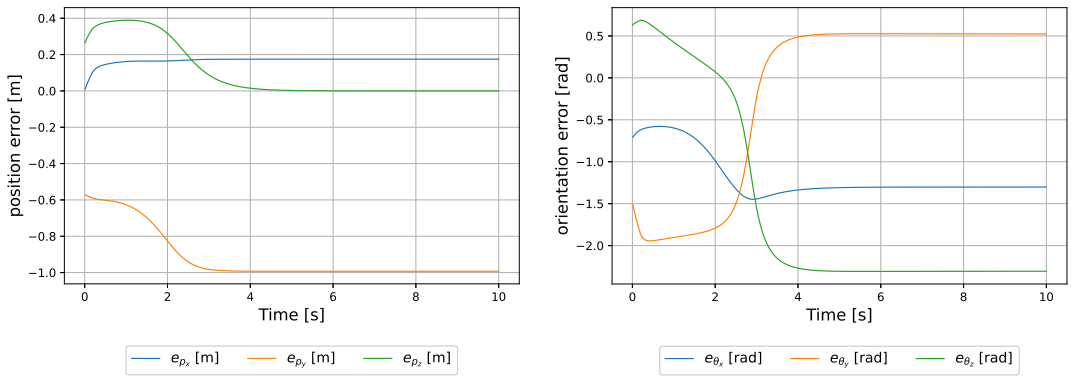

Figure 5.5: Position (left) and orientation (right) errors.

As it is possible to observe in this second case, the arm can not reach the desired pose with that priority order.

5.2.3 Priority of Maximization of Manipulability and distance from M.J.L.

The third and fourth priority position can change depending on the cost and on the target pose. Several simulations were conduced and, it is possible to show that if, proportionally, the importance of manipulability is higher the relate task will be in third position. The same for maximization of distances from mechanical joint limits. The target pose can heavily influence the position of the two tasks too, since n_3 tends to enclose the arm while n_4 tends to extend it. In this way if the arm, for example, needs to satisfy a configuration such that it is almost extended, maximization of distances from mechanical joint limits task will be in third place. So that, third and forth priority position can be swapped depending on the specific situation. However, it can be noticed that in all the conduced simulations, the number of failures (and then higher cost) of the robot due to singularities is lower if n_3 is in third position. This result can be theoretically confirmed since maximize manipulability means escaping from singularities. So, since the order of the stack, have to be fixed for the next phase, *Maximization of the Manipulability measure* task is fixed in third priority position, while *Maximization of distances from Mechanical Joint Limits* task is fixed in forth position.

5.2.4 Best derived Prioritized Order of the Stack of Tasks

In conclusion, the best possible order for the stack of tasks is the one derived trough learning in figure 5.6, and it is valid for both slave and master arm and

for the base.

Obstacle Avoidance
Inverse Kinematic
Maximization of the Manipulability
Maximization of distance from M.J.L.

Figure 5.6: Best stack of tasks.

This order allows the robot to reach the desired end-effector pose without colliding with obstacles along its path. As previously derived in section 5.2.2, it is important that the Maximization of the Manipulability and the Maximization of distance from Mechanical Joint Limits have a lower priority with respect to the Inverse Kinematic task, if the goal of the robot is to reach a desired pose. While as derived in section 5.2.3 the order of the two remaining tasks may depends on the specific case.

5.3 Best Parameters for the Stack of Tasks

In this section, once the best priority order is fixed and then, the *best parameters* can be found through learning with genetic programming. It is possible to randomly initialize the parameters of each task and then allowing them to change during simulation. The selection of the parameters is highly influenced by the cots function (for example if a task is considered through the cost function with relative cost) and it is not possible to find a general solution that satisfies all of them uniquely. So that, some example of learning of parameters are reported with relative cost functions. During the learning processes, the initial robot configuration of the the arms is randomly initialized at the beginning of each single simulation. The base, on the other hand, always starts with pose $\boldsymbol{q} = [0,\ 0,\ 0]$. Nevertheless, the plots reported are obtained testing the best algorithms starting the robot with base configuration

$$\boldsymbol{q}_{base} = [0,\ 0,\ 0]$$

and arms configuration

$$\boldsymbol{q}_{arm} = [\pm 0.403,\ -0.636,\ \pm 0.114,\ 1.432,\ \pm 0.735,\ 1.205,\ \mp 0.269].$$

5.3.1 Right slave arm precision - Empty environment

A first simulation is conduced in an environment with no obstacles in which, the right slave arm has to reach, with its end-effector, the target pose:

$$\boldsymbol{p}_d = [0.7,\ -0.855,\ 0.4,\ 0,\ \frac{2}{3}\pi,\ 0].$$

The cost function is given by the precision only, namely $cost = ||\boldsymbol{e}_{pose}||^2$, the initial population is made by 10 stacks and the algorithms stops after a maximum of 10 generations. Each single simulation lasts for 10 seconds.
At the end of the learning process the best algorithm in terms of cost is:

> [**7.805119815085215e-09**, ['n1', False, [0.6664701348988327, 0.11815516172949232, 1.5449203255977169, 0.6111134559947373]], ['**n2**', **True**, [1.2995040860177244, 3.894005338003036]], ['n3', False, [33.10591416613431]], ['n4', True, [31.063502981128376]]].

As it is possible to observe, the cost based on precision error is really low (almost zero), so that the reached precision is really high.

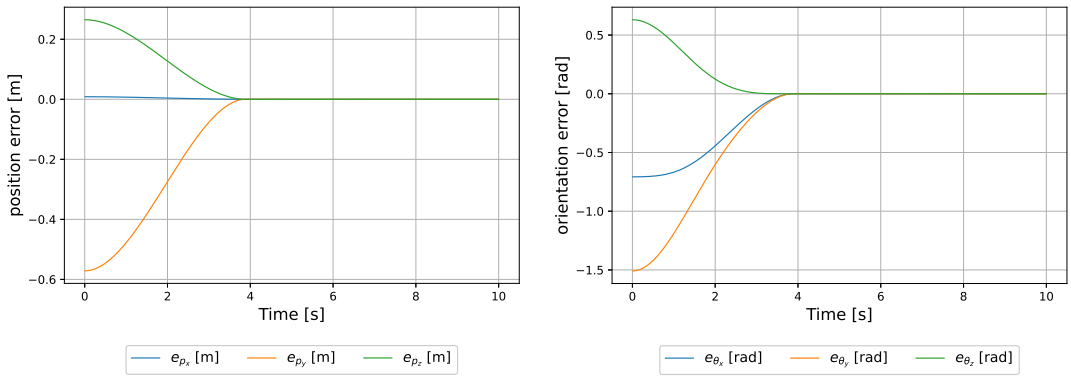

Figure 5.7: Position (left) and orientation (right) errors.

Obstacle avoidance and maximization of manipulability tasks are not active, since they do not play a crucial role in this specific situation and environment. Note that if there are no obstacles, $\boldsymbol{J}_o$ and $\dot{\boldsymbol{q}}_o$ are both null and then they do not act on the joints of the robot and in the null space combination. On the other hand the maximization of distances from mechanical joint limits is active even if it is not crucial for the achievement of the final goal. But, its presence do not influence the inverse kinematic task, allowing the robot to reach an high precision.
In the population of the 10^{th} generation, several algorithms have the same struc-

ture of the best one, with slightly different parameters and a bit higher cost (in the same order, 10^{-9}).

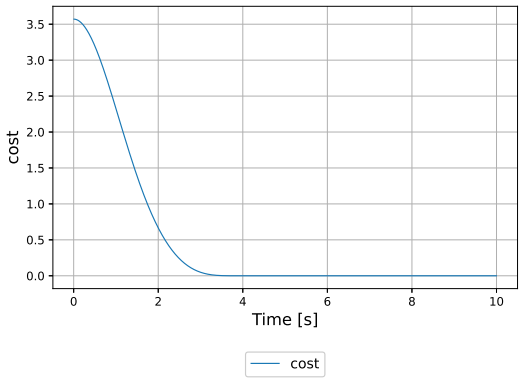

Figure 5.8: Cost evolution during the simulation time.

As it is possible to observe from the plot in figure 5.8, the cost decreases accordingly to the error decreasing.

Precision & time

A second sub-case was exploited about precision, with the introduction of time in the cost function. So, now the cost is given by:

$$cost = 0.5 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.5 \cdot t^2,$$

where t is the time of the simulation. The set-up of the simulations is the same of the previous case but, the single simulation of a stack stops earlier if all the components of the pose error $\boldsymbol{e}_{pose}$ are such that $e_i < 10^{-3}$. Then, a simulation is conduced and the final result is that the best algorithm is:

> [**37.49780059566292**, ’n1’, False, [0.47769720002399096, 0.25710903191981793, 0.12315666941139539, 1.0300102508932005, 6.601423019085237]], ’n2’, True, [1.498185820136733, 0.992253145308607]], ’n3’, True, [32.40971520489533]], ’n4’, False, [98.51371815755876]]].

In this case, it is possible to observe how parameters change for the Inverse Kinematic task. In fact, the trajectory time is really small $t_{traj.} = 0.992253145308607$ s, because if the trajectory is completed in a small amount of time the cost is clearly lower.

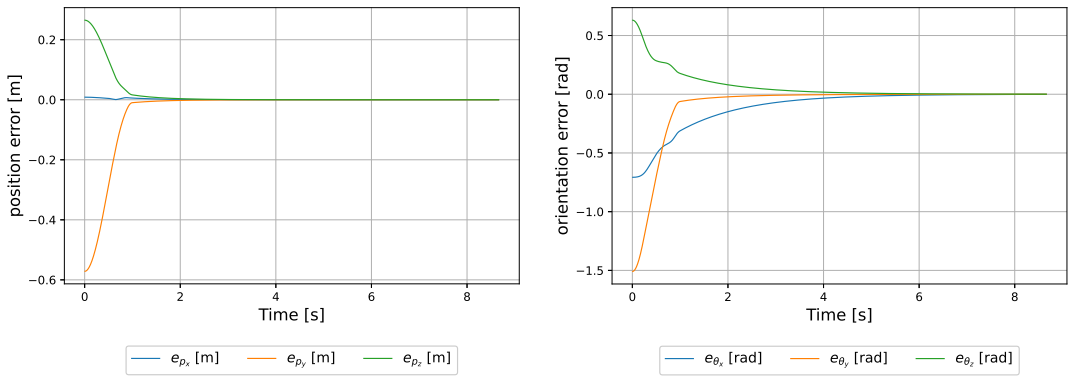

Figure 5.9: Position (left) and orientation (right) errors.

This, because if the target is reached with the desired precision fast, the time is smaller because the arm arrives earlier close to the desired pose. The desired precision is reached in $t = 8.67\ s$, as it is possible to observe in plot in figure 5.10. Note also how, from plots in picture 5.9, the error for position goes to zero faster with respect to the error for orientation.

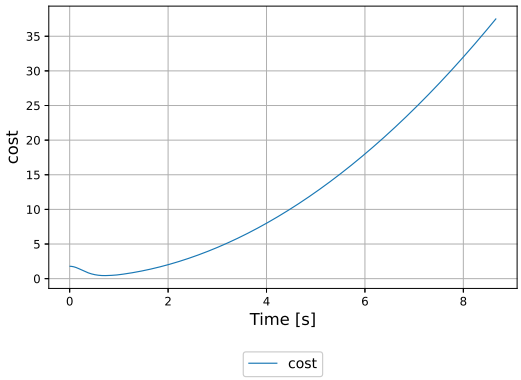

Figure 5.10: Cost evolution during the simulation time.

Looking at the cost, it is possible to observe how, in the beginning phase, it decreases thanks to the higher precision reached. However, after that it starts to grow with a exponential behavior given by the presence of the term t^2 in the cost function. Also in this case, the latest populations have stacks with similar parameters.

5.3.2 Right Master arm precision - Empty environment with a wall

The task, cost function and the initial condition of the robot are the same of section 5.3.1 but, in this case, a wall is placed between the robot and the goal position and the arm is used in master configuration. Right slave arm has to reach the target pose:

$$\boldsymbol{p}_d = [0.7,\ -1.7,\ 1.3,\ 0,\ \frac{2}{3}\pi,\ 0].$$

At the end of the learning process, the best stack is:

> [**1.41820747238802378**, ['**n1**', **True**, [0.5014486982670608, 0.1805397868749832, 0.16454565221094722, 1.1529072927840212, 8.599313395498248]], ['n2', True, [0.9456859434654472, 1.210091975875358]], ['n3', True, [18.65010638992065]], ['n4', False, [11.818643817308317]]].

In this case, it is possible to note how the Obstacle Avoidance task is active, since if it is not the robot will collide against the wall present in this simulation environment. In the final population, several algorithms have parameters similar to the best one. So, the robot achieve the minimum cost, maintaining an equilibrium between distance from the wall and precision. In fact it is possible to note that the cost is substantially increased from the one derive in section 5.3.1, but the robot do not collide, avoiding an even greater cost.

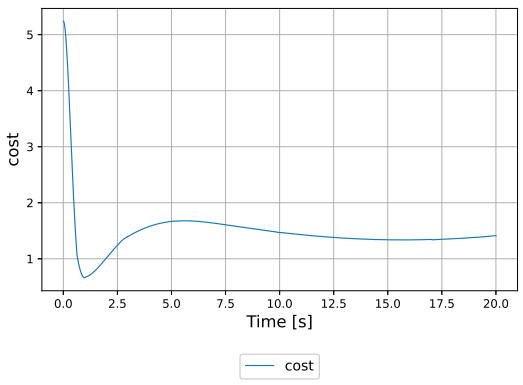

Figure 5.11: Cost evolution during the simulation time.

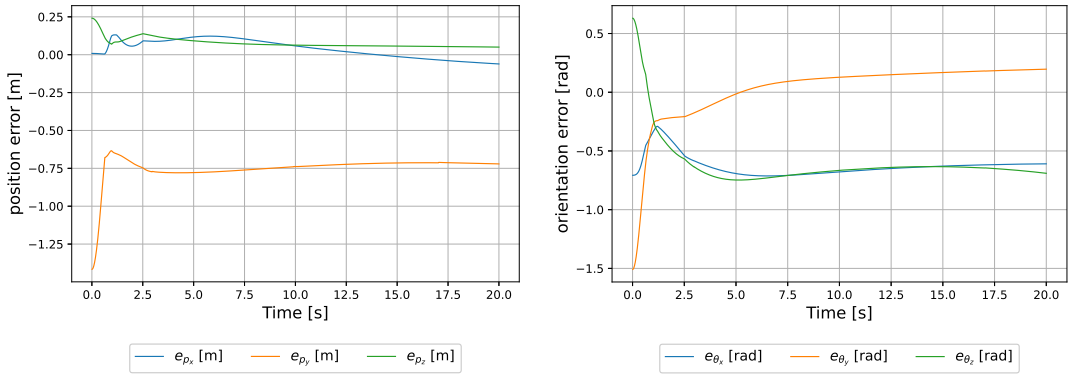

Figure 5.12: Position (left) and orientation (right) errors.

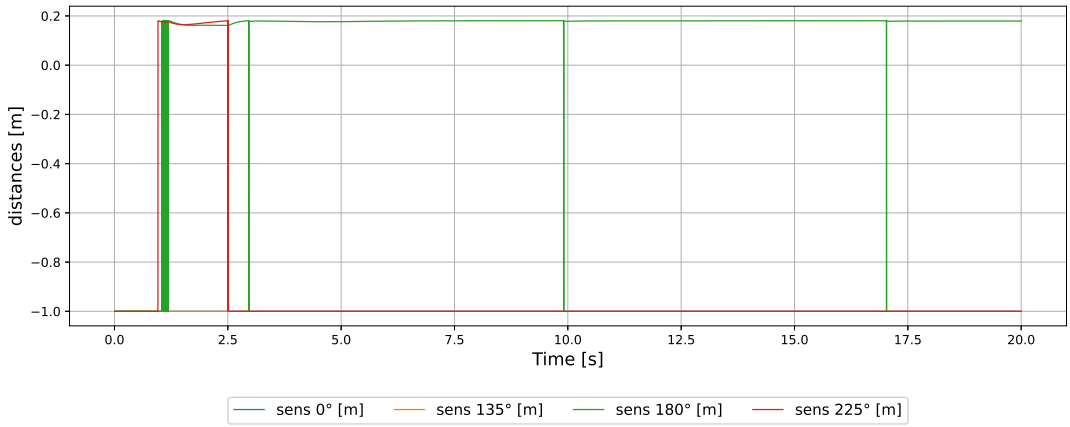

Figure 5.13: Arm 2^{nd} site distances.

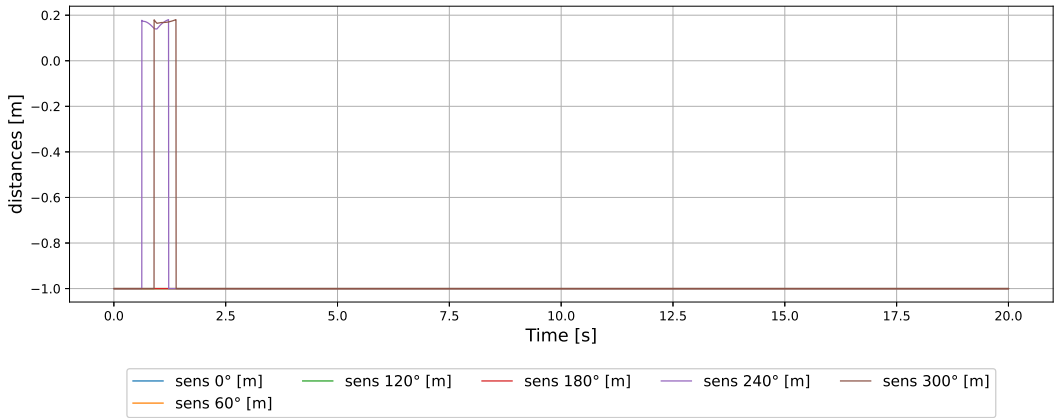

Figure 5.14: Arm 3^{rd} site distances.

It is important to note that the parameters referring to distances in the obstacle avoidance task are all close to the 10% of their potential range, meaning that the

robot tries to activate this task mowing away from the wall as late as possible, in order to reach an higher precision, and then, decrease the cost.

5.3.3 Precision & Distances from Obstacles - Static Obstacles Environment

In this case obstacles were added in the simulation, using the Static Obstacle Environment (section 3.1.2), creating a more complex setting for he simulation. The cost function is given by

$$cost = 0.7 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.3 \cdot \frac{1}{2} \sum_{k=1}^{N} (r_k - d_k)^2,$$

and the right arm, as master, has to reach the target pose expressed in world reference frame:

$$\boldsymbol{p}_d = [3.7,\ 1.7,\ ,1.3,\ 0,\ \frac{2\pi}{3},\ 0].$$

So that, precision and distances from the obstacles have both weight in this simulation. Each single simulation lasts 15 seconds.
Once the learning process is terminated after 10 generations, the best algorithm among the last population of 10 elements is:

> [**4.1968870995830e-03**, ['**n1**', **True**, [0.59093102143878236, 0.20491971819803764 ,0.18371220272697103, 1.4735898852364415, 2.829156571031647]], ['**n2**', **True**, [1.4880297288713509, 3.325501927194323]], ['n3', False, [77.00218578597881]], ['n4', True, [63.36645340681146]]].

However, in the final population, several stacks have close performances and similar parameters.

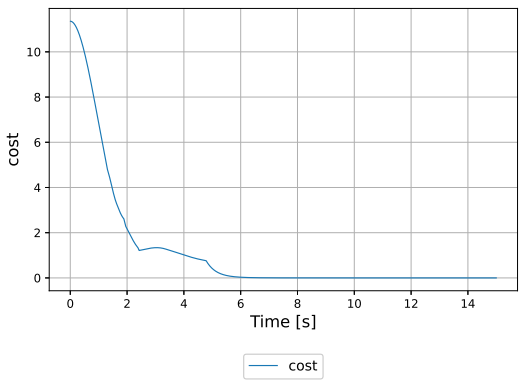

Figure 5.15: Cost evolution during the simulation time.

As it is possible to observe from plot in figure 5.15, the cost decreases while the robot is approaching the desired pose $\boldsymbol{p}_d$ and moves away from obstacles. Following, the metrics of interest are reported. It is possible to observe how the descending behaviour of the cost changes when obstacles are sensed.

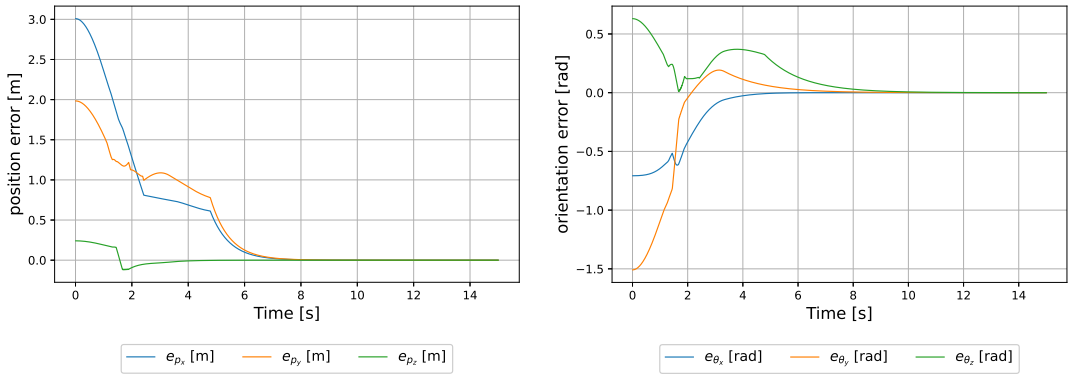

Figure 5.16: Position (left) and orientation (right) errors.

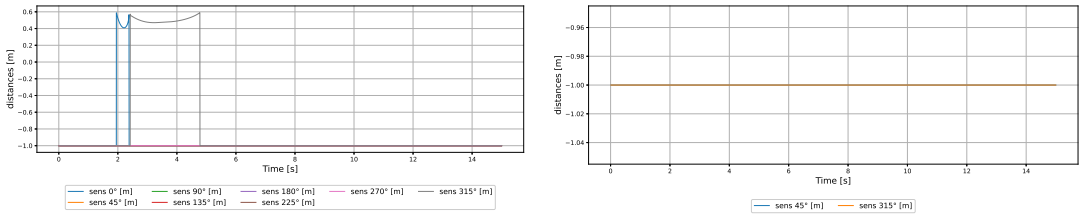

Figure 5.17: Base and arm 1^{st} sites distances.

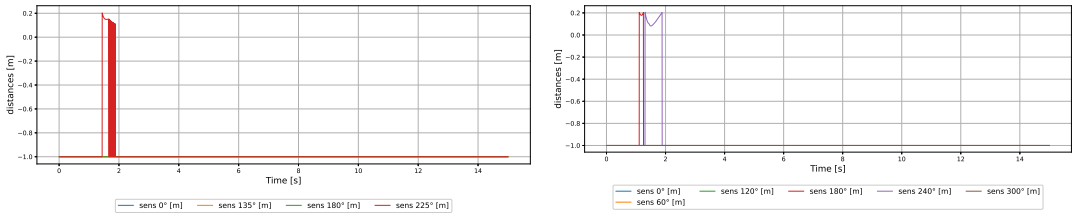

Figure 5.18: Arm 2^{nd} and 3^{rd} sites distances.

5.3.4 Precision & Manipulability - Empty Environment

In this case the cost is given by Precision and Manipulability, in an environment free of obstacles:

$$cost = 0.7 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.3 \cdot \frac{1}{w^2_{manip}}.$$

The slave right arm, has to reach the desired pose

$$\boldsymbol{p}_d = [0.7,\ -1.7,\ 1.3,\ 0,\ \frac{2}{3}\pi,\ 0],$$

the initial population is composed by 10 elements and the algorithm stops after 10 generations. Each simulation lasts 20 seconds.
At the end of the learning process, the best stack is given by:

[**35.590931021438782**, ['**n1**', **True**, [0.65631806501372, 0.3115061982146442, 0.2825299823293798, 1.480115700087536, 1.8182167191099674]], ['n2', True, [1.578200817462834, 7.105678428115425]], ['n3', True, [77.44459222135943]], ['n4', True, [61.15589266102345]]].

However, almost all the elements in the final population show a similar result and similar parameters. Note that n_1 is active, but since there are no obstacles in the environment its contribution is null.

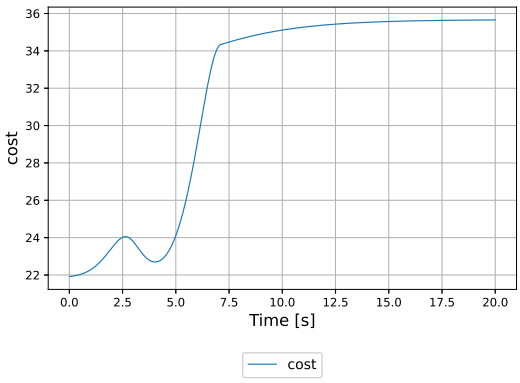

Figure 5.19: Cost evolution during the simulation time.

As it is possible to observe from the plot in figure 5.19 the cost is high, compared with other simulations, since the manipulability is a low number ($\simeq 0.084$) and the cost is given by $cost = \frac{1}{w^2_{manip}}$.

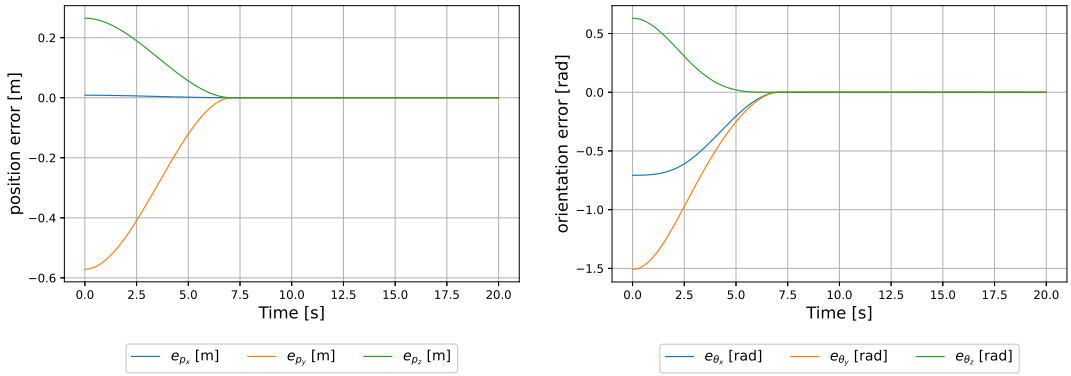

Figure 5.20: Position (left) and orientation (right) errors.

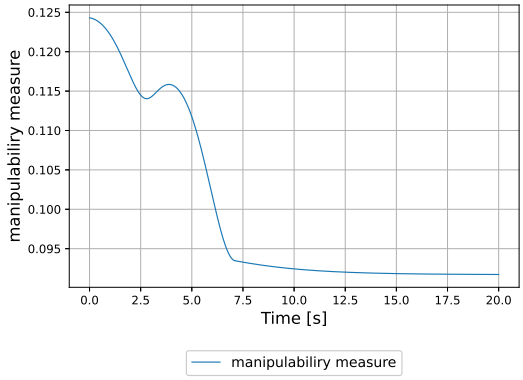

Figure 5.21: Manipulability measure of the right arm.

Master Arm - Obstacles environment

Another simulation of the final stack was conduced with the master arm, adding 3 degrees of freedom to the kinematic chain with target pose:

$$\boldsymbol{p}_d = [3.7,\ 1.7,\ ,1.3,\ 0,\ \frac{2\pi}{3},\ 0].$$

After 15 s the resulting cost is 15.936.

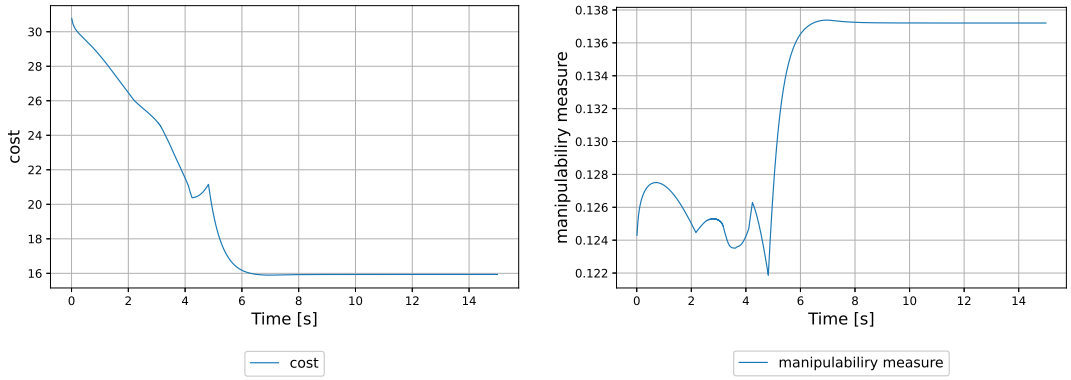

Figure 5.22: Cost (left) and manipulability measure (right) evolution during the simulation time.

As it is possible to observe, the cost is lower with respect to the case in which the arm is used as slave. This happens because, having more degrees of freedom and being the base mobile, the robot is able to reach a configuration which helps to increase the manipulability of the arm, and then reducing the cost, maintaining the precision.

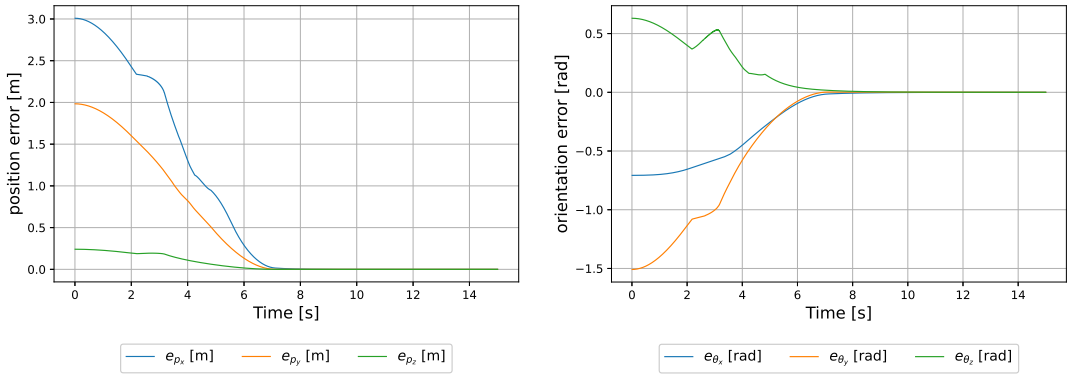

Figure 5.23: Position (left) and orientation (right) errors.

In this second case, the obstacle environment was used to test the effectiveness of the stack in a more general environment, since also the Obstacle Avoidance task is active. Observe that the desired pose is reached, despite the presence of obstacles, avoiding collisions.

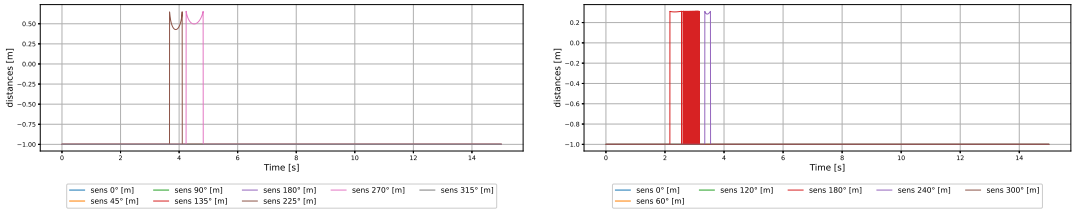

Figure 5.24: Base and arm 3^{rd} sites distances.

5.3.5 Precision & distances from Mechanical Joint Limits - Empty Environment

In this case, the cost function is given by a combination of precision and distances from mechanical joint limits. The arm is used in slave configuration in the empty environment. The right arm have to reach the target pose:

$$\boldsymbol{p}_d = [0.7,\ -1.7,\ 1.3,\ 0,\ \frac{2}{3}\pi,\ 0].$$

The initial population is made of 10 stacks and the algorithm stops after 10 iterations. Each simulation lasts 15 seconds. At the end, all the algorithms present a similar behaviour and similar parameters.

$$cost = 0.7 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.3 \cdot w^2_{m.j.l.}.$$

Once the learning process is terminated the best stack is:

> [**0.0006181198973304119**, ['**n1**', **False**, [0.9482469491528949, 0.3725980375994233, 0.1574773779409989, 1.8133214405465417, 4.426497816500577]], ['n2', True, [0.5235502923757622, 0.9555034931366369]], ['**n3**', **False**, [61.51966907137499]], ['n4', True, [82.17786511382929]]].

Note that, the task n_3 is not active and it is crucial. If it was, the task n_4 couldn't have reached its target of maximizing the distance from mechanical joint limits. n_1 is not active, but the absence of obstacles would have made it a zero contribution task.

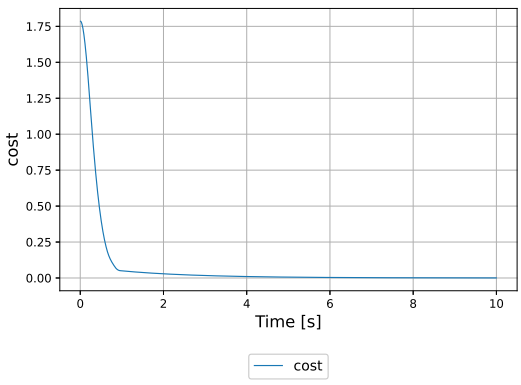

Figure 5.25: Cost evolution during the simulation time.

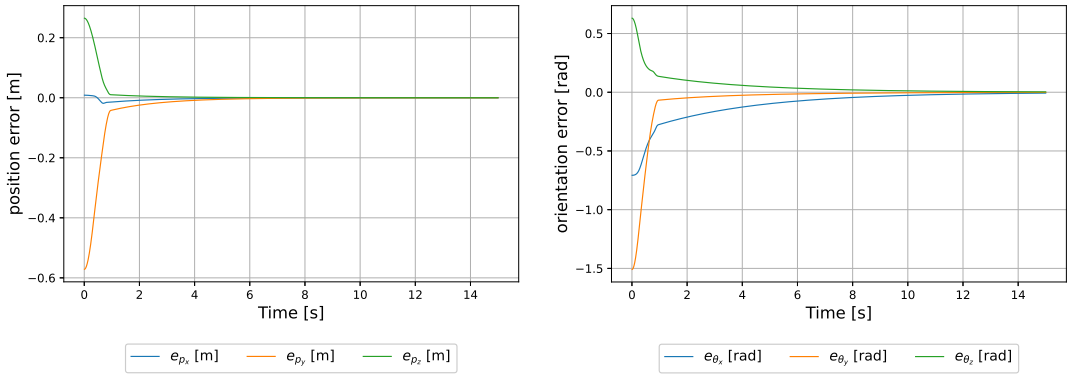

Figure 5.26: Position (left) and orientation (right) errors.

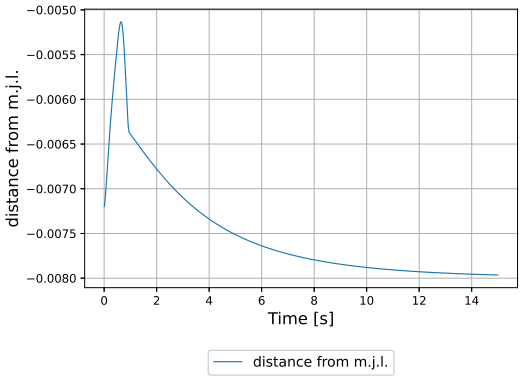

Figure 5.27: Distance from mechanical joint limits measure of the right arm.

Note that the cost is really close to zero, since the precision is really high and the distance from mechanical joint limits is maximized as possible. However, the values of the distance from mechanical joint limits are in the order of 10^{-3}, meaning that it's impact on the cost function is not really effective, until the arm is really close to the desired pose. In fact, comparing the cost in figure 5.25 with the one in 5.8, it is possible to observe a significant change in the slope of the function only once its value is close to zero.

5.4 Manipulability & distances from Mechanical Joint Limits - Empty Environment

In this section the cost function is given by Manipulability and distances from Mechanical Joint Limits only, precision is not considered.

$$cost = 0.5 \cdot \frac{1}{w_{manip}^2} + 0.5 \cdot w_{m.j.l.}^2$$

As it is possible to observe from the cost function, for the robot is not important to reach a certain pose. However, the target pose must be specified since the Inverse Kinematic task is present in the stack. In this simulation the right arm is used as slave, since the base is not implied in none of the two elements of the cost function
At the end of the learning process, 10 generations, with a population made by 10 elements, the best stack is:

[25.591578229522549, ['n1', True, [0.5039302484993167, 0.1574773779409989, 1.022431650776821, 2.193063242854384]], ['**n2**', **False**, [1.7961991584672319, 7.81242649500164]], ['n3', True, [94.73200248683295]], ['n4', True, [73.3497087362359]]]

Note that, n_2 is not active. If it was, it would not have left enough freedom of movement to the arm in order to fully accomplish the n_3 and n_4 tasks.

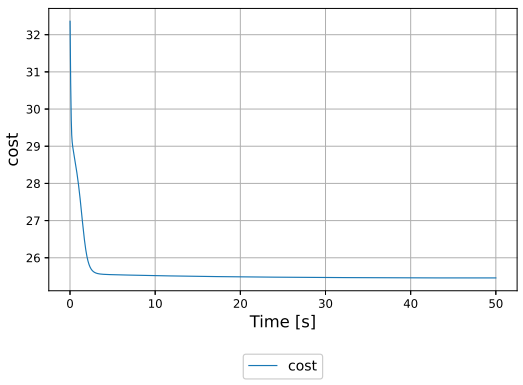

Figure 5.28: Cost evolution during the simulation time.

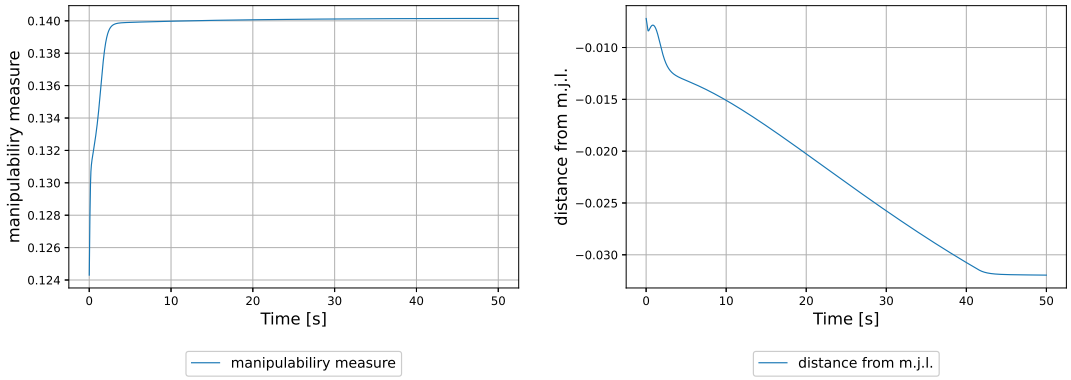

Figure 5.29: Manipulability and distance from mechanical joint limits measures of the right arm.

5.4.1 Precision, Time, Distances from Obstacles, Manipulability & Distances from M.J.L - Static Obstacles Environment

In this case, all the costs are present in the convex combination:

$$cost = 0.4 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.3 \cdot \frac{1}{2}\sum_{k=1}^{N}(r_k - d_k)^2 + 0.1 \cdot \frac{1}{w_{manip}^2} + 0.1 \cdot w_{m.j.l.}^2 + 0.1 \cdot t^2.$$

Each simulation lasts 20 seconds, each population is made of 10 individuals and the algorithm stops after 20 iterations.
The arm, in master configuration, has to reach the final target pose

$$\boldsymbol{p}_d = [3.7,\ 1.7,\ ,1.3,\ 0,\ \frac{2\pi}{3},\ 0],$$

avoid obstacles along the path and attempt to maximize manipulability and distance from mechanical joint limits.
At the end of the leaning process the best stack obtained in the learning process is:

> [**0.000218241783179509**, ['**n1**', **True**, [0.84507865966939866, 0.12213883748029469, 0.10108390180166468, 0.902158147580735, 6.612233865163163]], ['**n2**', **True**, [1.2785646893230078, 6.102977119861568]], ['**n3**', **True**, [78.41294350054314]], ['**n4**', **True**, [35.863477385773024]]]

Note that all the tasks are active. Following all the metric of interest are reported.

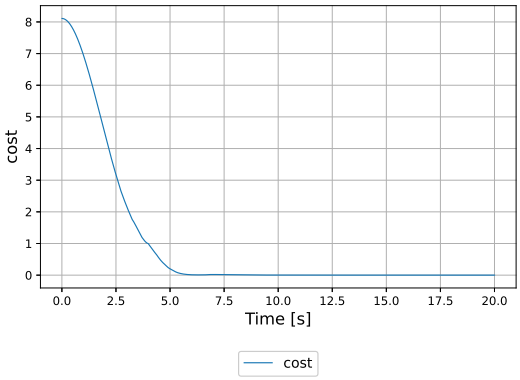

Figure 5.30: Cost evolution during the simulation time.

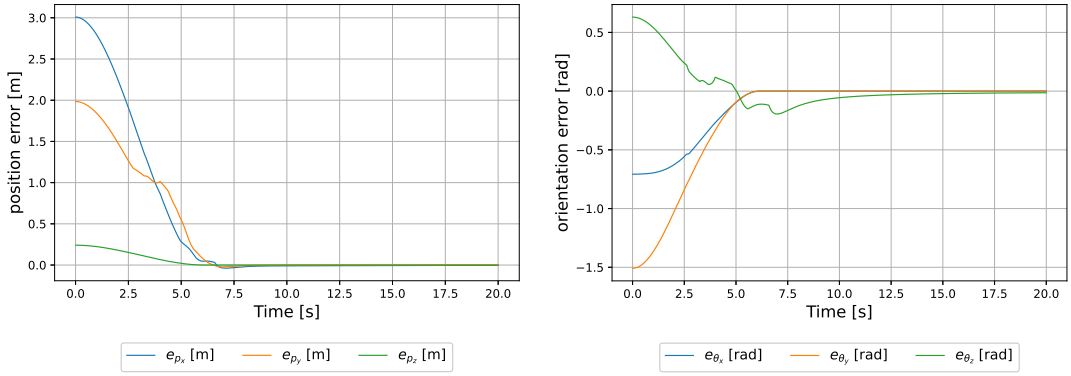

Figure 5.31: Position (left) and orientation (right) errors.

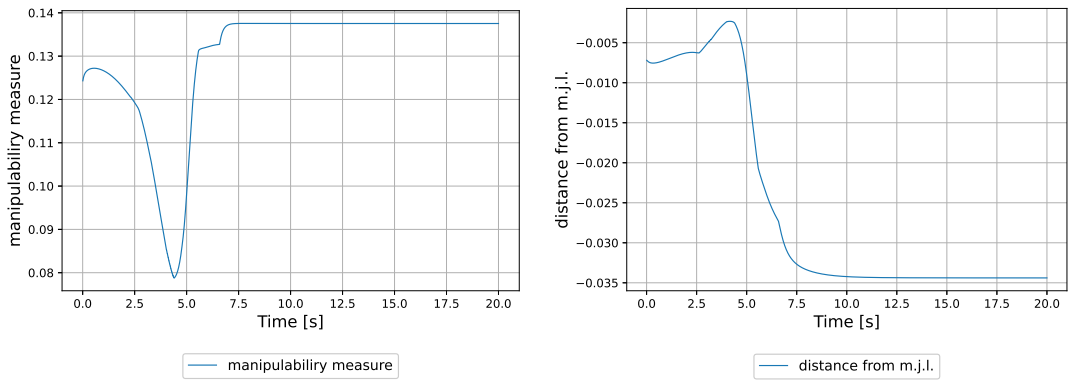

Figure 5.32: Manipulability and distance from mechanical joint limits measures of the right arm.

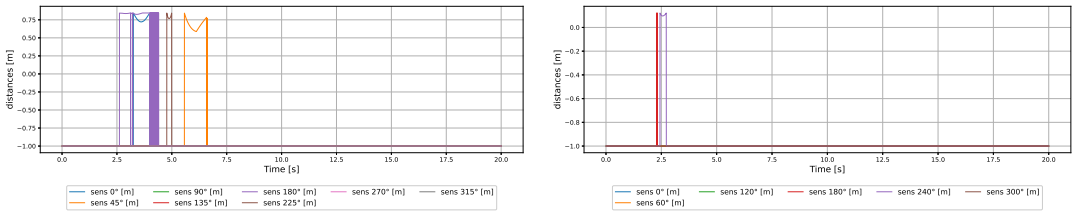

Figure 5.33: Base and arm 1^{st} sites distances.

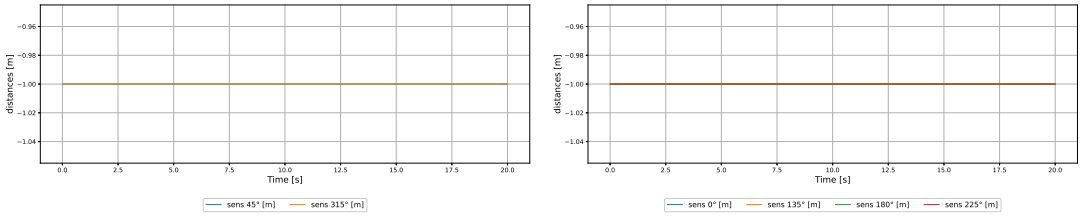

Figure 5.34: Arm 2^{nd} and 3^{rd} sites distances.

The robot accomplish its tasks moving away from obstacles, reaching the desired pose and maximizing the manipulability measure. However, the distances from mechanical joint limits can not be maximized, since there is not enough freedom of movement to solve this task. Note that, the cost related to this last task is 0.1, which is lower with respect precision and distances from obstacles, but it is the same of the cost related to maximization of manipulability measure. But, in this case, the cost of the two tasks have different weights in the fitness measure since, in general, $\frac{1}{w_{manip}^2} \gg w_{m.j.l.}^2$. Considering what has just been said, the algorithm acts as expected, without maximizing the last measure and giving priority to other tasks.

5.4.2 Precision & Distances from Obstacles - Dynamic Obstacles Environment

In this case, a dynamic obstacle environment is tested in order to prove that the robot is able to learn how to escape from dynamic obstacles in the environment. The cost is given by:

$$cost = 0.7 \cdot ||\boldsymbol{e}_{pose}||^2 + 0.3 \cdot \frac{1}{2} \sum_{k=1}^{N} (r_k - d_k)^2.$$

The objective of this simulation, is to reach a desired pose with the end-effector of the right master arm avoiding obstacles. After that, a dynamic obstacle comes

close to the robot simulating an human operator which interacts with the end-effector. The target pose is given by

$$\boldsymbol{p}_d = [3,\ 1.7,\ 1.3,\ 0,\ \frac{2\pi}{3},\ 0].$$

Once the pose is reached with all the error elements of $\boldsymbol{e}_{pose}$ with modulus lower than 10^{-3}, the dynamic obstacle moves toward the robot with velocity 0.5 m/s along the y axis. Then, once the dynamic obstacle has covered 3 m, it inverts its velocity moving away from the robot at -0.5 m/s.
Each simulation lasts 50 seconds, each population is made by 10 stacks and the algorithm stops after 10 iterations.
At the end of the learning process, the best performing stack is:

[**6.66477497111878e-05**, ['n1', True, [1.1172054449654453, 0.11857644031248402, 0.10164344404011497, 1.7260753881272786, 1.3855131187054726]], ['n2', True, [1.2551133685916285, 3.6090185464932354]], ['n3', True, [35.30699434742218]], ['n4', False, [15.001361835437589]]]

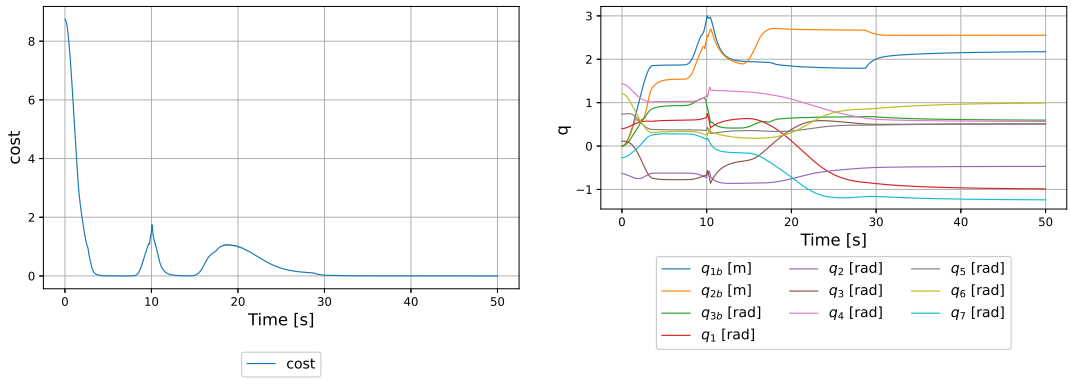

Figure 5.35: Cost evolution (left) during the simulation time and joint positions q (right) in configuration space.

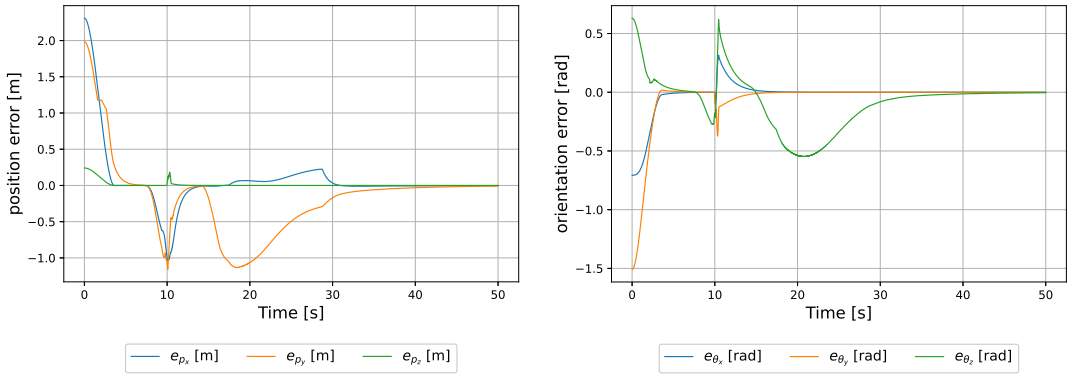

Figure 5.36: Position (left) and orientation (right) errors

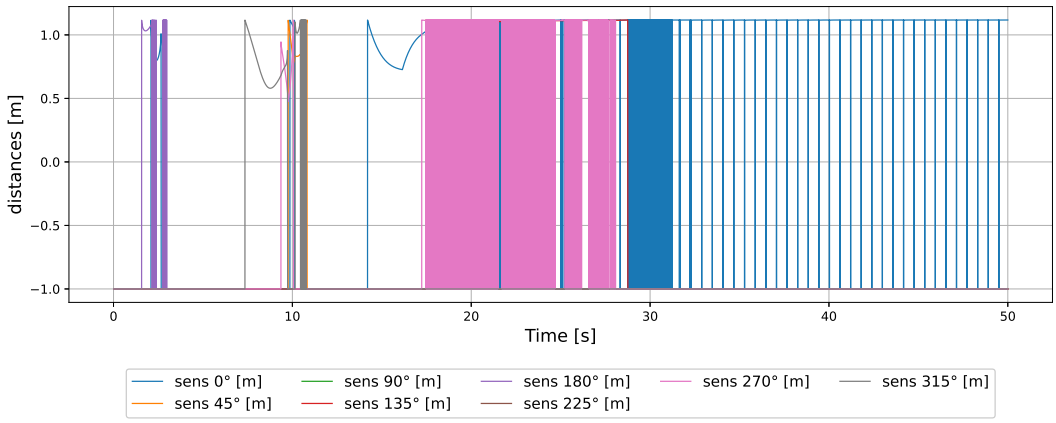

Figure 5.37: Base site distances.

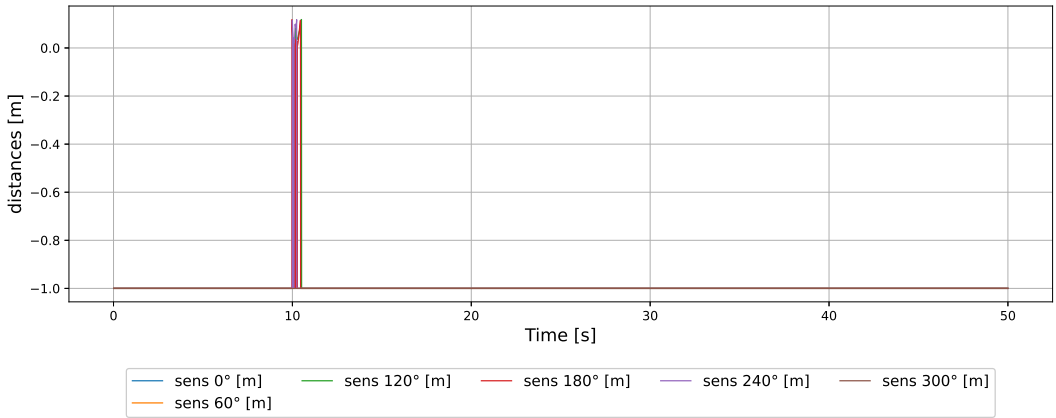

Figure 5.38: Arm 3^{rd} site distances.

Observe how it is possible to see from plots the behaviour of the robot in a dynamic environment. First of all the robot reaches the desired pose dealing

with fixed obstacles along its path. Once the robot has reached the desired pose with the specified accuracy, the dynamic obstacle starts to move towards the robot. It is possible to see from plot in figure 5.37 that at almost 8 s, the base sensor placed at 315° sens the obstacle coming. The the robot starts to move in order to avoid it, misleading the target pose in order to avoid collisions. Then, the obstacles moves away and the robot quickly return to its target pose. Later, the dynamic obstacle returns in its original position and, after that, moves again towards the robot. This time, the robot is in a different configuration when the obstacle is sensed (see plot of positions in configuration space in figure 5.35) and, the reaction is different since the base active different sensors. The base is moved further away with respect to the first time and then also the return in the desire position is slower. Finally the obstacle stays in its original position and the robot is allowed to stay in its desired pose and reach an higher accuracy. Note, that the trajectory time for the obstacle avoidance task is not high. This because, if it had been high the robot would have not enough time to escape from the dynamic obstacle before a collision.
So that, it is demonstrated through simulation, that the developed algorithm makes the robot capable of reacting in dynamic changing environments, avoiding collisions.

5.5 Robustness of the algorithm to useless tasks

In this section it is demonstrated that if a task is not useful, given a cost function which must be minimized, it is still present in the stack at the end of the genetic operations but, it is deactivated. For this purpose, a completely useless task is used to prove this fact (section 3.7.2). The simulation starts with 10 elements in the initial population, and ends after 10 generations of stacks. The goal of the robot is to reach the target pose

$$[1.7,\ -0.7,\ 1.3,\ 0,\ \frac{2\pi}{3},\ 0]$$

with the right arm considered as master arm. The cost function is given by a weighted combination of all the costs that are related to the useful tasks, namely distances from obstacles, precision, manipulability measure and distances from mechanical joint limits. All this components are equally weighted with $\alpha = 0.25$ each. The useless task, is the only element of the stack that is allowed to start in a random position in the priority order, and that can change its priority during

mutation or crossover process. This last feature is only present in this simulation and helps to prove the robustness of the algorithm. Following, the best obtained result is reported

> [4.998323960911508, 'n1', True, [0.34507865966939866, 0.10108390180166468, 0.902158147580735, 6.612233865163163]], ['n2', True, [1.2785646893230078, 6.102977119861568]], ['n5', **False**, [0.3497087362359]],['n3', True , [78.41294350054314]], ['n4', True, [35.863477385773024]]]

As it is possible to note, the task was deactivated, and then it does not give any contribution to the final velocities imposed on the robot joints. All the others tasks are active.
This result proves that the algorithm is robust to useless tasks for the given cost function, deactivating them if not needed.

Chapter 6

Test of the learned stack of tasks

In this section the algorithms founded through learning in chapter 5 are tested, executing a pick and place duty with the robot. The choices are two, as shown in section 3.10, for the same operation. One of the arms can be considered as master, or both can be considered as slave with the base moving in the environment as master. In both cases the environment is the same showed in section 3.1.3. The task is considered to be successfully computed if the object is placed on the corresponding pedestal.
In both cases, the base of the robot starts in pose $\boldsymbol{p} = [0, , 0, 0]$ and the object is placed in position $\boldsymbol{x}_{pick} = [2, -3, 1]\ m$. The place position is in the opposite side of the room $\boldsymbol{x}_{place} = [-2, 3, 1]\ m$. Since the stack in section 5.4.2 was the best solution founded in a dynamic environment, it was used in both the cases due to the presence of dynamic and not dynamic obstacles.
In these simulations only full dynamic environment is used, in order to show the performances of the robot with the learned stack in a real situation.

6.1 Base Master

In this first simulation, the base can move as master reaching the pick-up point for the object.
The initial pose of the base is $\boldsymbol{p} = [0, 0, 0]$ and the robot have to reach the pose, defined in world frame,

$$\boldsymbol{p}_d = [-2, 2.3, -\frac{\pi}{2}]$$

to be in the right position. However, the base is allowed to reach $\boldsymbol{p}_d$ with a an error of 5 cm in position and 0.1 rad in orientation, since the arms would be able to pick up the object also with this slight imprecision.

After that, the base is stopped and the arms reach the handle of the pot. The target poses, expressed with respect to the base frame, are

$$\boldsymbol{p}_{left} = [0.7 + e_x,\ 0.2 + e_y,\ 0.25,\ -\frac{\pi}{2},\ \pi + 0.01,\ 0 + e_z]$$

and

$$\boldsymbol{p}_{right} = [0.7 + e_x,\ -0.2 + e_y,\ 0.25,\ \frac{\pi}{2},\ \pi + 0.01,\ 0 + e_z]$$

where $\boldsymbol{e}_{pose} = [e_x,\ e_y,\ e_{\theta_z}]$ is the error pose of the base. In this way the robot reaches exactly the target pose in order to pick up the pot.
Once both the arms are in position, the grippers are closed and then the pot is grabbed. This operation lasts for 1 s. After that, the arms move up of 0.05 m in order to take away the pot from the pedestal.
While the grippers remain closed, the base moves and reaches the target pose, defined in world frame,

$$\boldsymbol{p}_d = [-2,\ 2.3,\ -\pi].$$

In this point, the grippers open and the object is places on the pedestal.
Along its path, the robot avoids static and dynamic obstacles, as showed in figure 3.1.4. Following, the results obtained for each phase are reported.

Movement of the base to the pick position

First, the base moves in order to reach the target pose with the accepted precision. From plots, it is possible to note that this operation takes about 11 s.

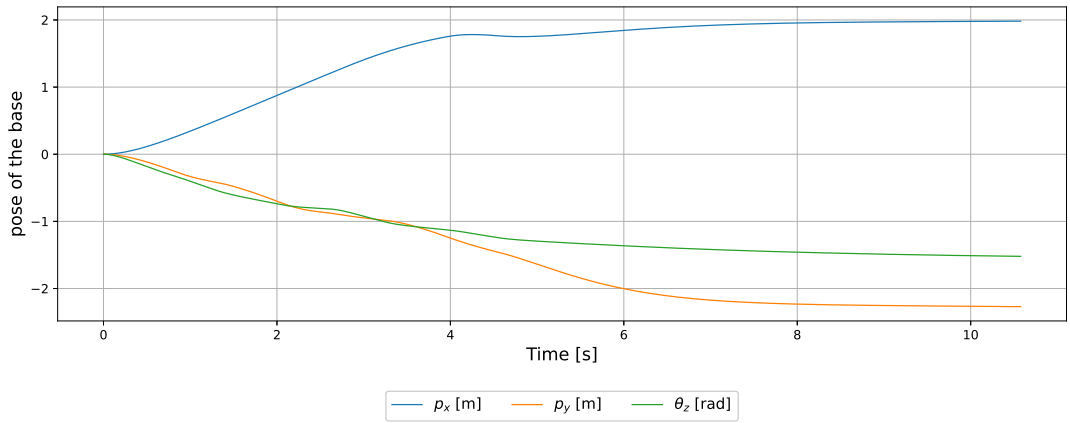

Figure 6.1: Pose of the base.

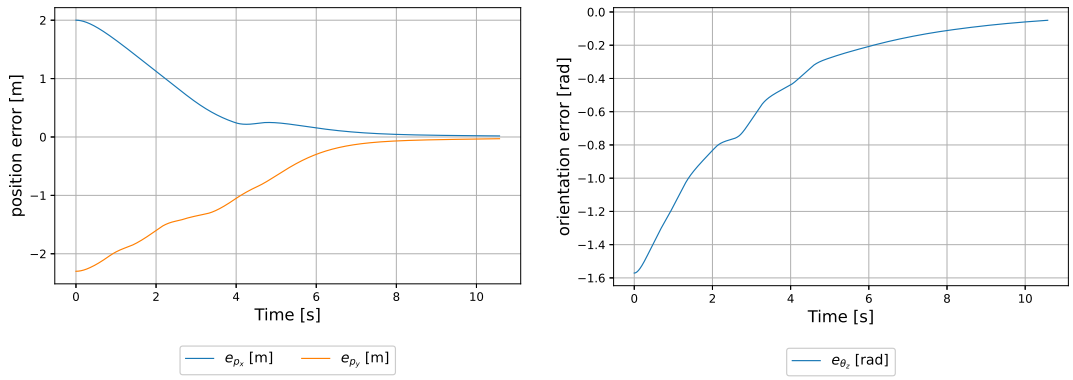

Figure 6.2: Base linear velocities (left) and base angular velocity (right).

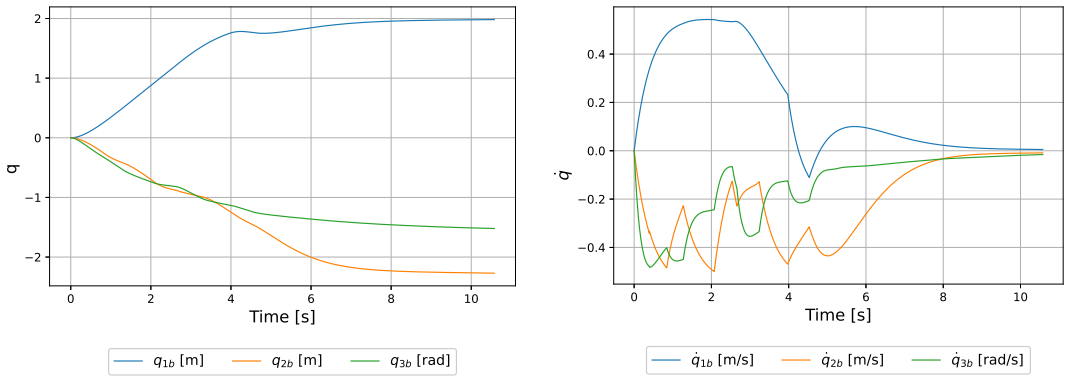

Figure 6.3: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

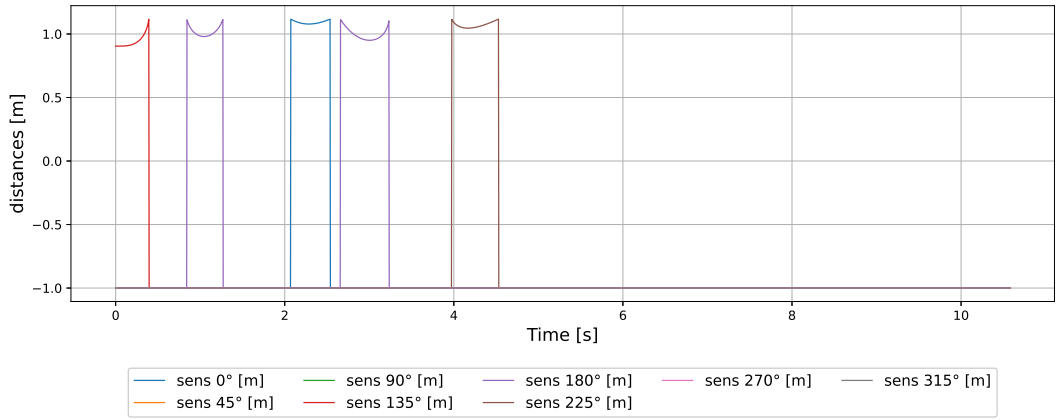

Figure 6.4: Distances of the base sensors.

From plots in figure 6.2 and figure 6.3 it is possible to observe the reaction of the robot to the presence of fixed obstacles along the path. While in figure 6.4 it is possible to observe when and from which sensor the obstacles are detected. The

robot moves away quickly, since the trajectory time for the obstacle avoidance task is $t_{traj} \simeq 1.4\ s$.

Movement of the arms to pick the pot

Once the robot's base is in position, the arms start to move toward the desired pose, taking into account the inaccuracy of the base. First the result for the right arm are reported.

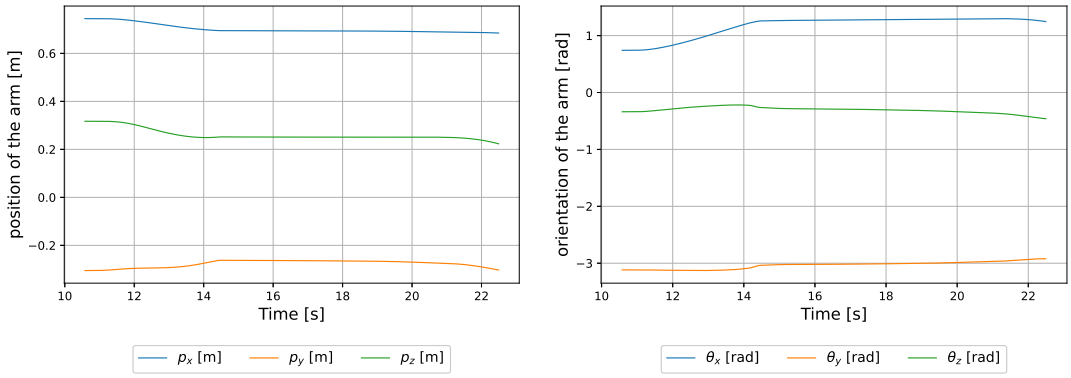

Figure 6.5: End-effector position (left) and orientation (right).

Note how the arm reaches the desired pose at almost 14 s from the beginning of the simulation.

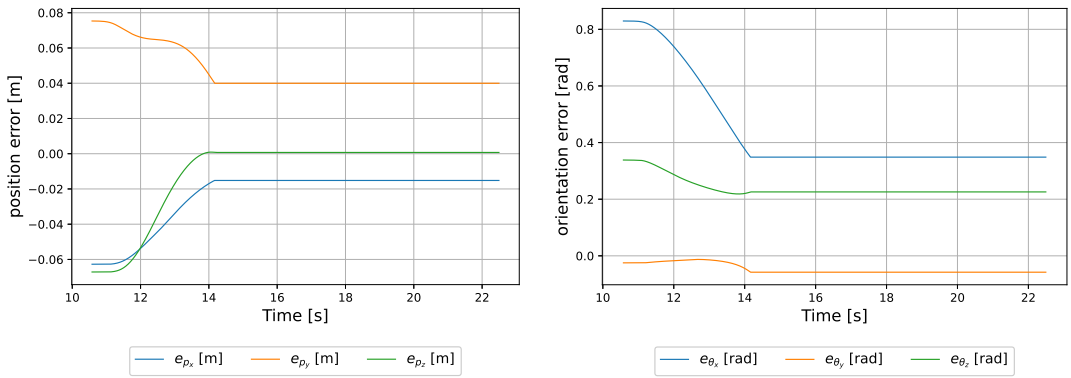

Figure 6.6: Position (left) and orientation (right) errors.

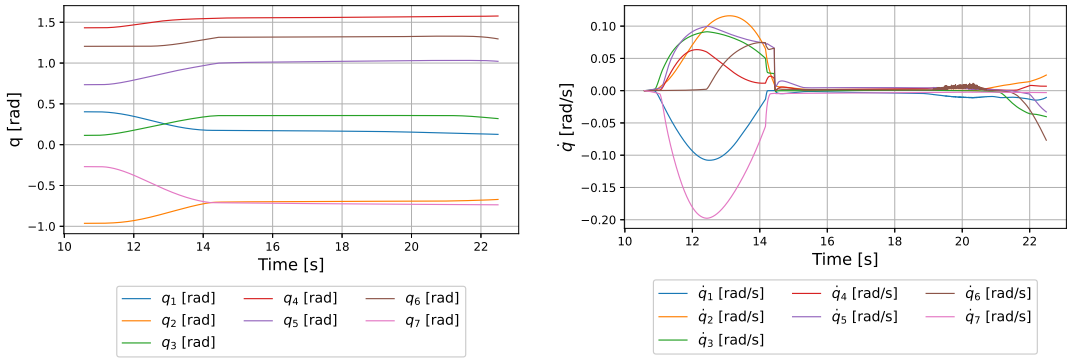

Figure 6.7: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

It is possible to see from plots in figures 6.7 and 6.8 that after the target pose is reached, the arm keeps moving in order to increase manipulability, since the task is active in the stack.

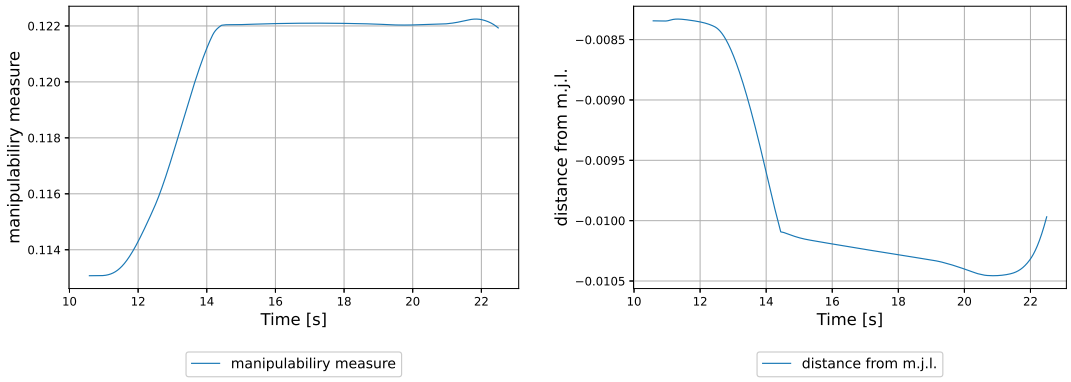

Figure 6.8: Manipulability and distance from mechanical joint limits measures of the right arm.

Then, plots about left arm are reported. First of all it is possible to note that the left arm requires more time to reach the target pose, reached at almost 23 s, since the arm start farther with respect to the right arm from the relative targets.

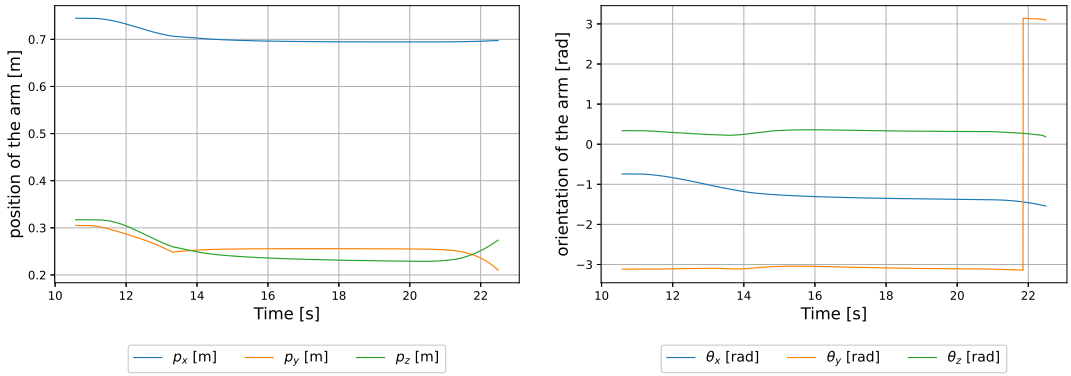

Figure 6.9: End-effector position (left) and orientation (right).

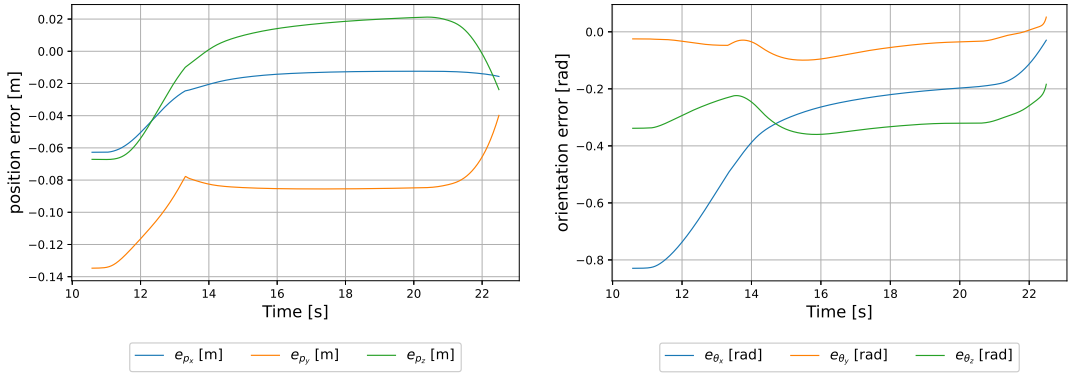

Figure 6.10: Position (left) and orientation (right) errors.

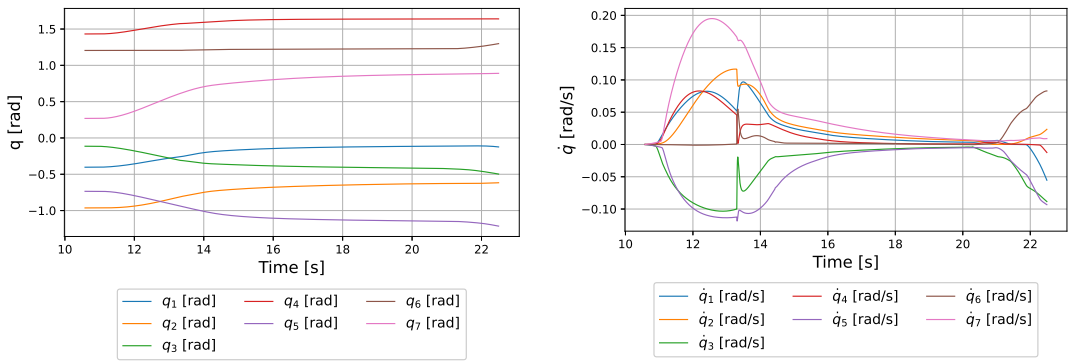

Figure 6.11: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

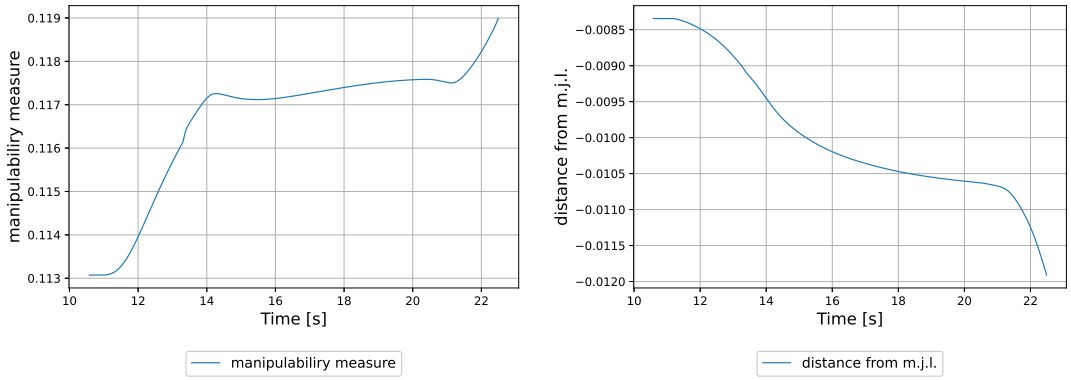

Figure 6.12: Manipulability and distance from mechanical joint limits measures of the left arm.

Also in this case, once the arm is close to the target pose, it starts to augment its manipulability.
Once both the arms are in position with the desired precision, the grippers are closed and the arms get up of 0.05 m in order to raise the pot from the pedestal. Those two phases are not reported with plots since they are not relevant. However this phase takes approximately 7 seconds.

Movement of the base to the place position

Once the pot is lifted, the base starts to move toward the place pose.

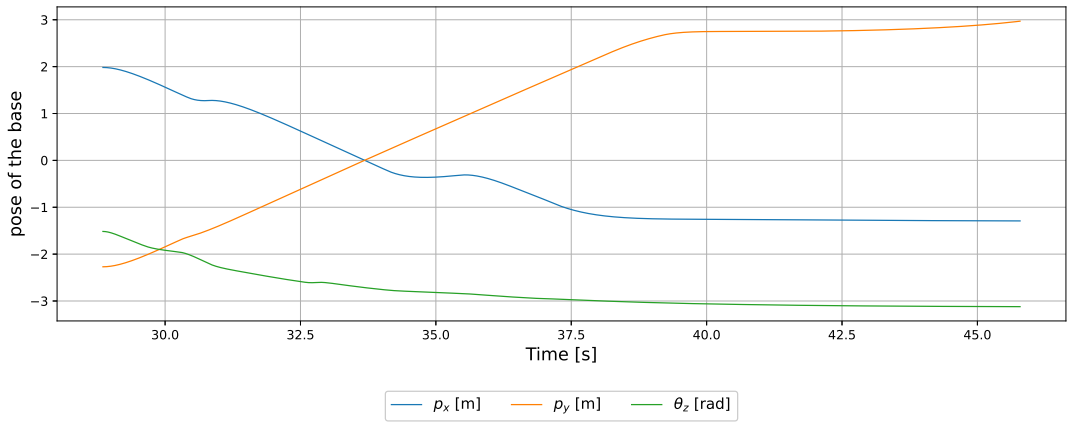

Figure 6.13: Pose of the base.

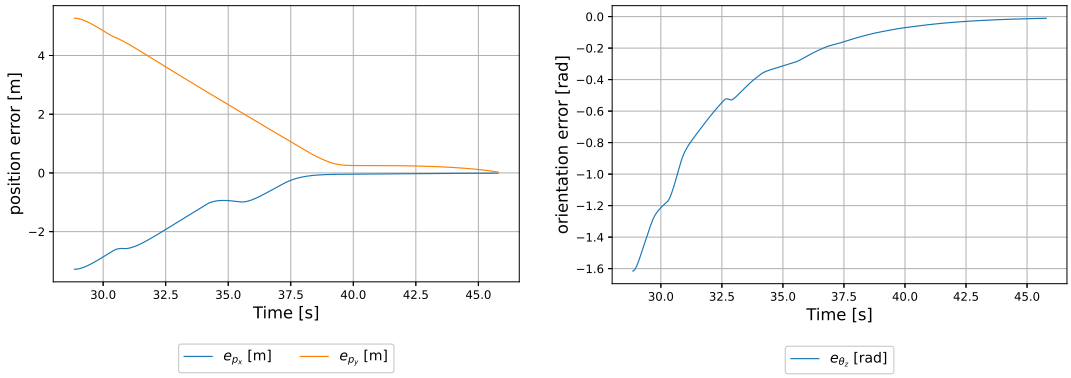

Figure 6.14: Base linear velocities (left) and base angular velocity (right).

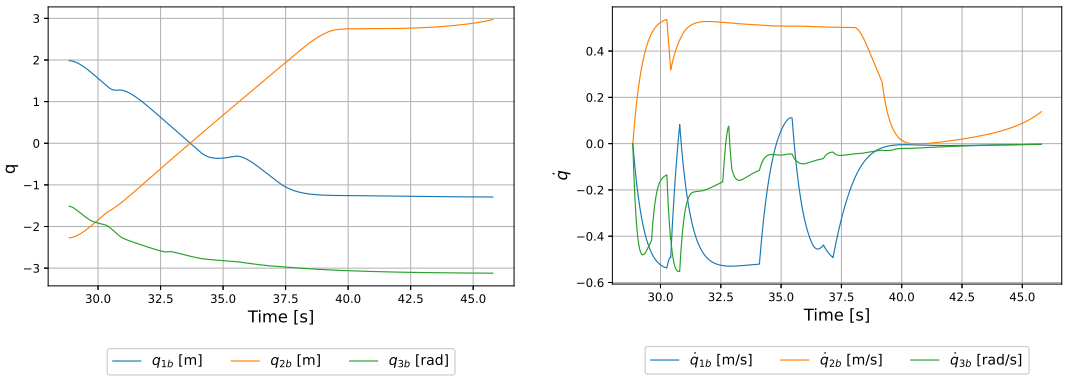

Figure 6.15: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

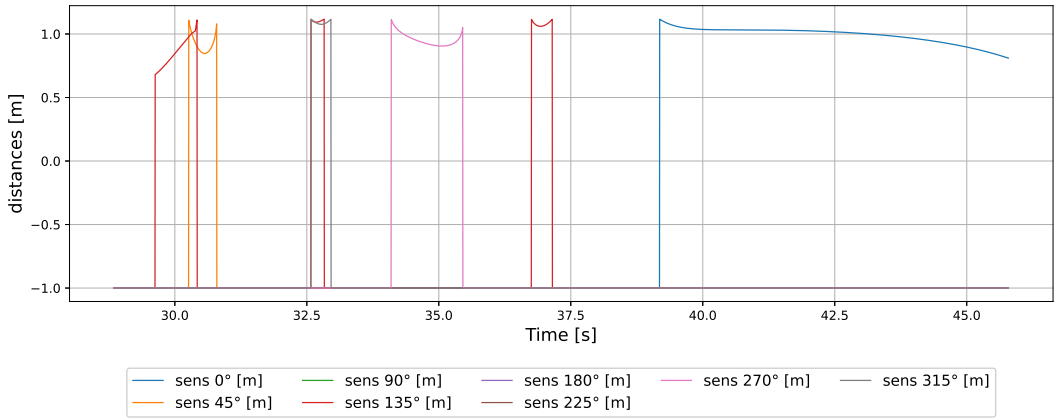

Figure 6.16: Distances of the base sensors.

With the pots in its hands, the robot reaches the desired place pose for the base after 46 s from the beginning of the simulation. It is possible to observe how it avoids the static obstacles in the first phase of the path, escaping in a short time.

However, when it is close to the target pose, the dynamic obstacle arrives and pushes the robot away from it. In fact, it is possible to note how the convergence slope of the error in position changes, and slow down its behaviour, in figure 6.14 when the dynamic obstacle is sensed. This happens because the contribution of the obstacle avoidance task is no longer null and, it uses degrees of freedom to escape from the obstacle, removing them from the inverse kinematic task.
Once the pose is reached with the desired accuracy, the grippers open and the object is on the place position and the simulation stops.

Pick & place images sequence

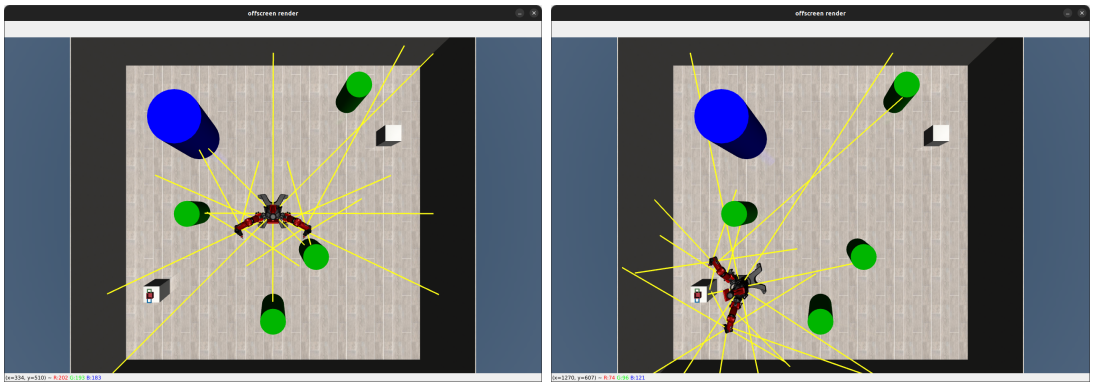

Figure 6.17: Robot moving toward the desired pick pose.

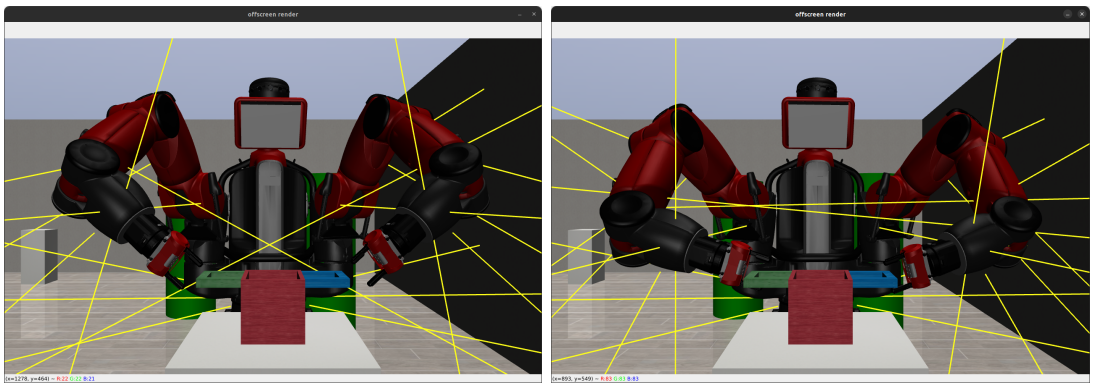

Figure 6.18: Picking of the pot, once the robot is in position.

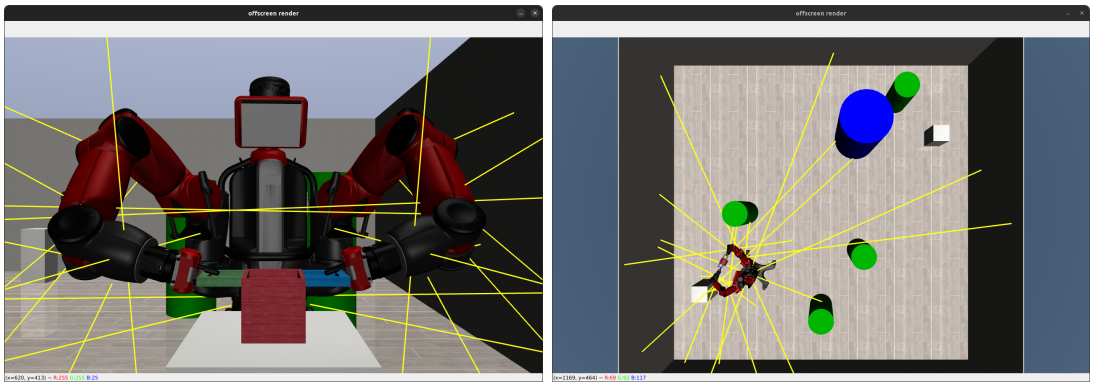

Figure 6.19: Once the grippers are closed, the robot moves toward the place pose.

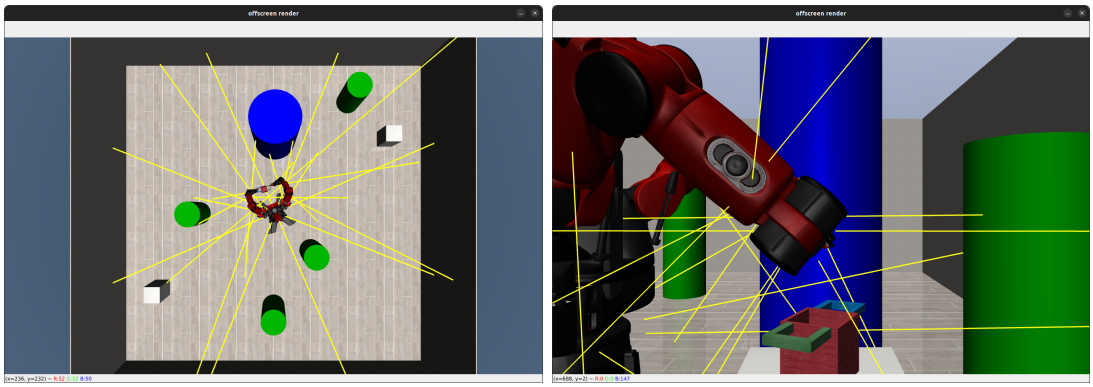

Figure 6.20: The robot moves between the obstacles and once the place pose is reached the pot is left.

6.2 Arm Master

In this case one arm (left) moves as master trough the target pick pose where, an object is placed. The pot used in the previous chapter was substituted with a cylindrical can, so only one hand is required to pick up the object. The other arm is not used. The environment is the same used with two slave arms (section 3.1.3).

The desired pose, expressed with respect to the world reference frame, is

$$\boldsymbol{p}_d = [2,\ -3,\ 1.15,\ \frac{\pi}{2},\ 0,\ -\frac{\pi}{2}].$$

However, in order to pick the object with a small velocity and higher precision, the robot first reaches a pose close to the desired one, in particular $\boldsymbol{p}_{d_{first}} = [2,\ -2.9,\ 1.15,\ \frac{\pi}{2},\ 0,\ -\frac{\pi}{2}]$, and then it moves toward $\boldsymbol{p}_d$.

After that, the gripper is closed for 1 s and the object is slightly lifted in order to not collide with the table. Then, once the object is picked and lifted, the master

arm moves toward the target place pose:

$$\boldsymbol{p}_d = [-2,\ 3,\ 1.15,\ 0,\ -\frac{\pi}{2} + 0.01,\ 0].$$

Once that this desired pose is reached, the gripper opens and the arm moves away. Then, the task is successfully completed.
During the simulation time, the robot deals with dynamic and static obstacles as showed in section 3.1.3. In this simulation, the robot is allowed to perform with a precision of 0.02 m for positions and 0.1 rad for orientations.

Movement of the robot to $p_{d_{first}}$

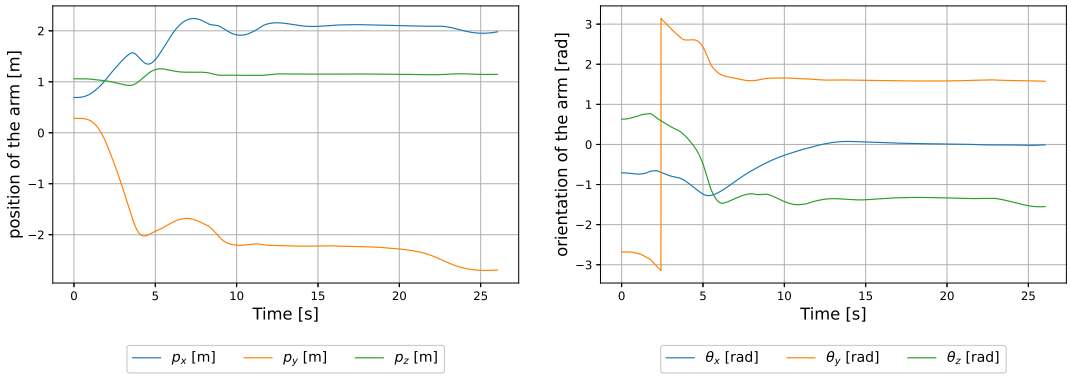

Figure 6.21: End-effector position (left) and orientation (right).

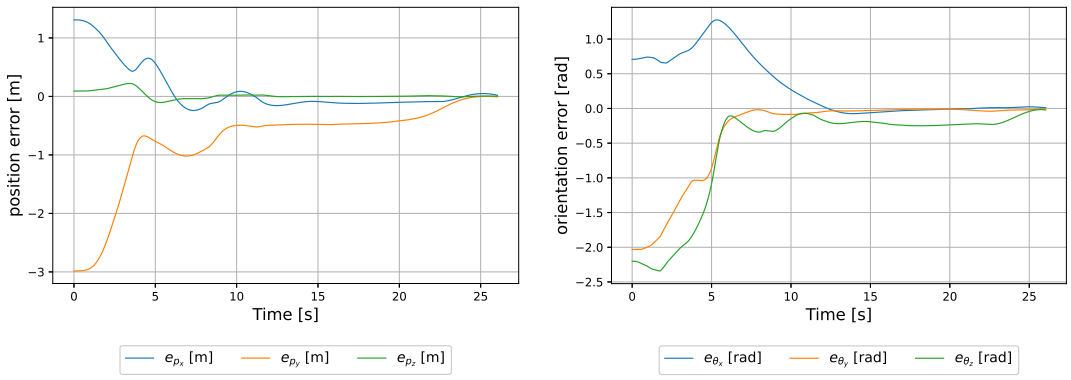

Figure 6.22: Position (left) and orientation (right) errors.

Looking at the plots it is possible to observe that the arm reaches the desired pose really late, despite of the specified trajectory time. This happens not only for the presence of dynamics but, in particular, for the presence of obstacles since the robot spends time to escape from them.

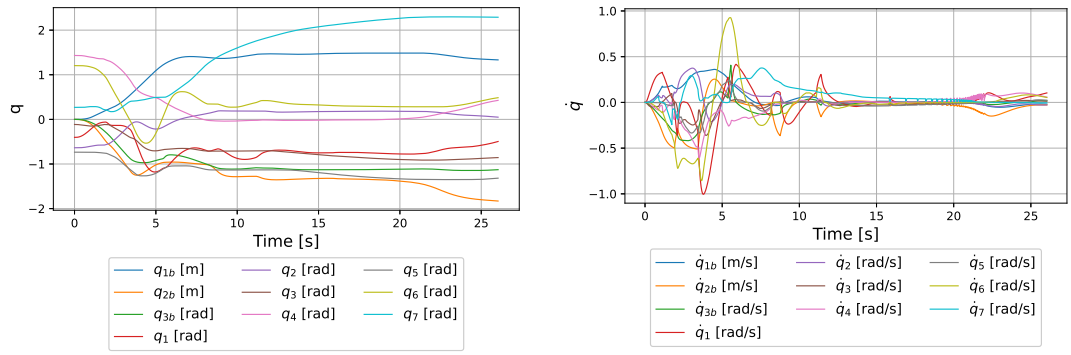

Figure 6.23: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Observing the velocities of the joints (figure 6.23) and the manipulability plot (figure 6.25) it is possible to observe how the robot attempts to maximize the cited measure once the target is almost reached.

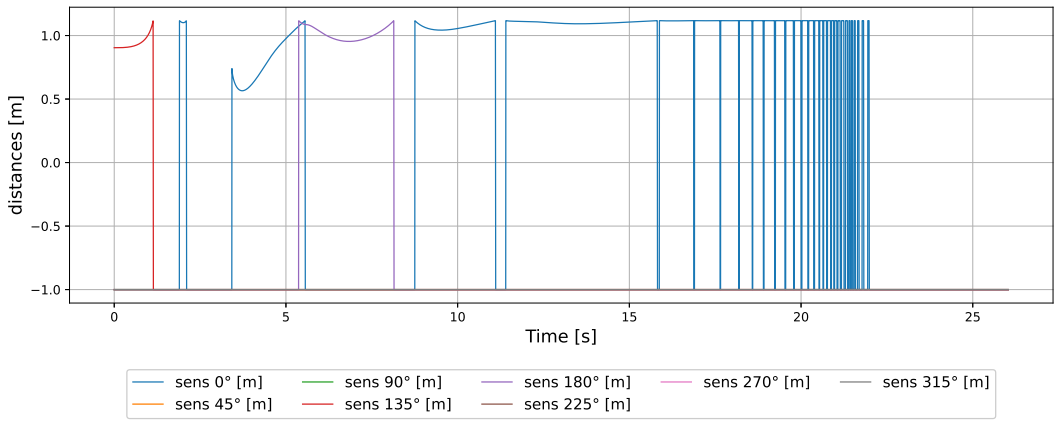

Figure 6.24: Base sensors site.

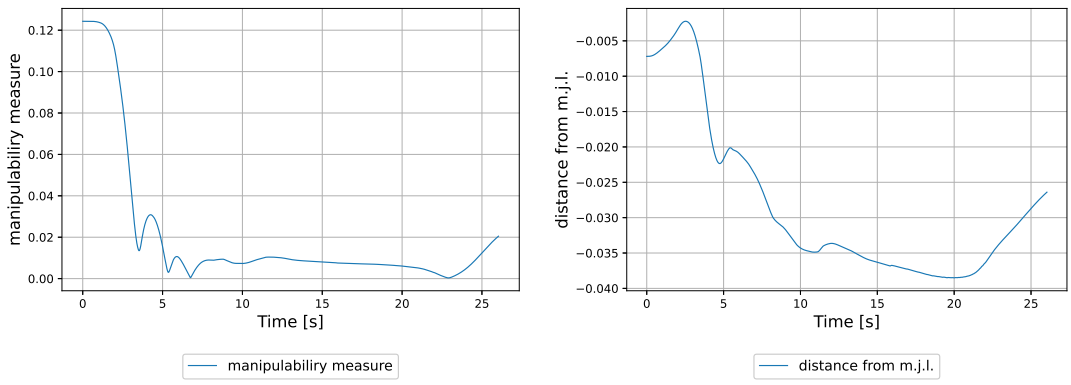

Figure 6.25: Manipulability and distance from mechanical joint limits measures of the left arm.

Movement of the robot to the pick p_d

Once the robot is in $\boldsymbol{p}_{d_{first}}$ it reaches the desired final target pose $\boldsymbol{p}_d$ in order to pick the object. The distance to cover is not long, but this process requires 3.5 s, namely the time indicated t_{traj} for the inverse kinematic task.

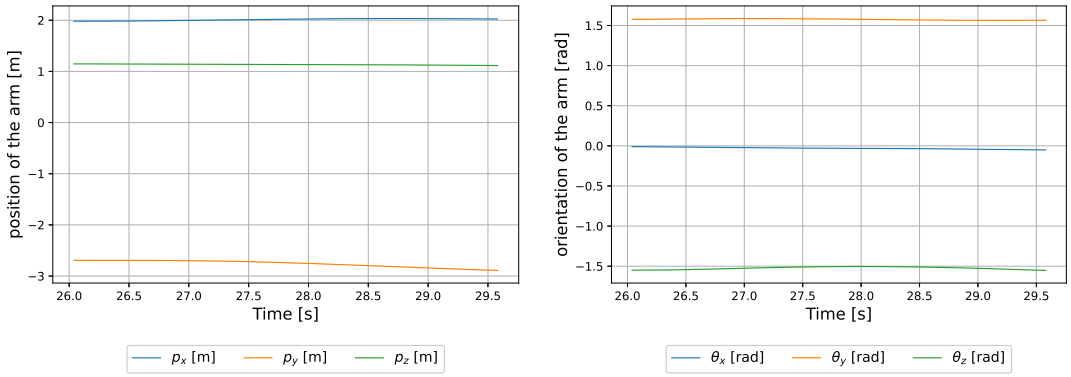

Figure 6.26: End-effector position (left) and orientation (right).

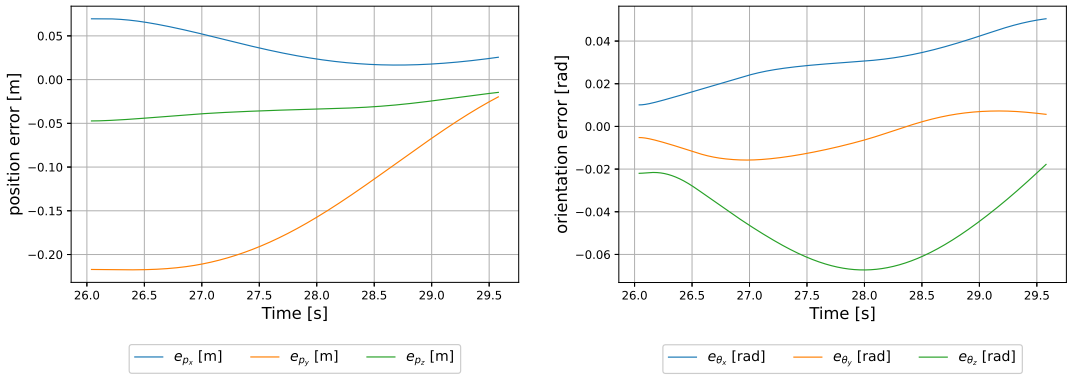

Figure 6.27: Position (left) and orientation (right) errors.

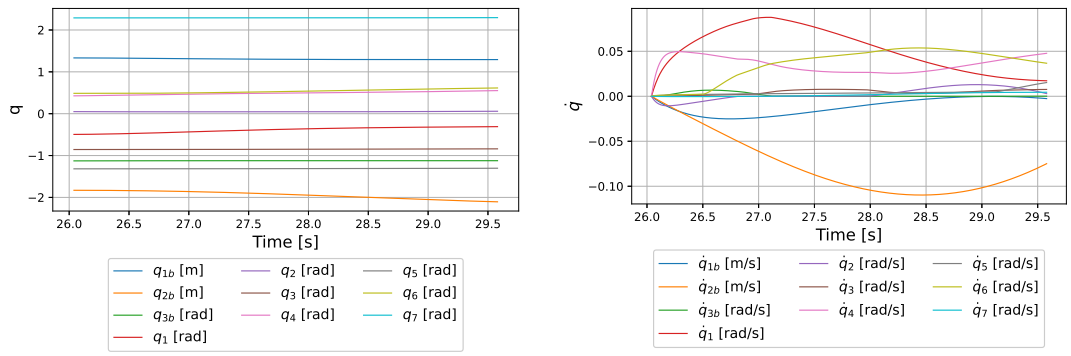

Figure 6.28: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

Movement of the robot to the pose p_d

Once the object is picked and lifted, it is possible for the robot to reach the desired target pose in order to place the object on the second table and, complete its duty.

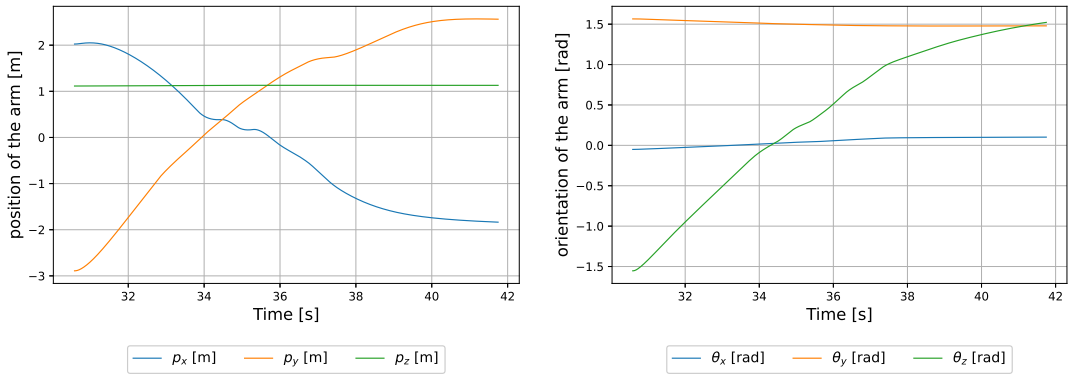

Figure 6.29: End-effector position (left) and orientation (right).

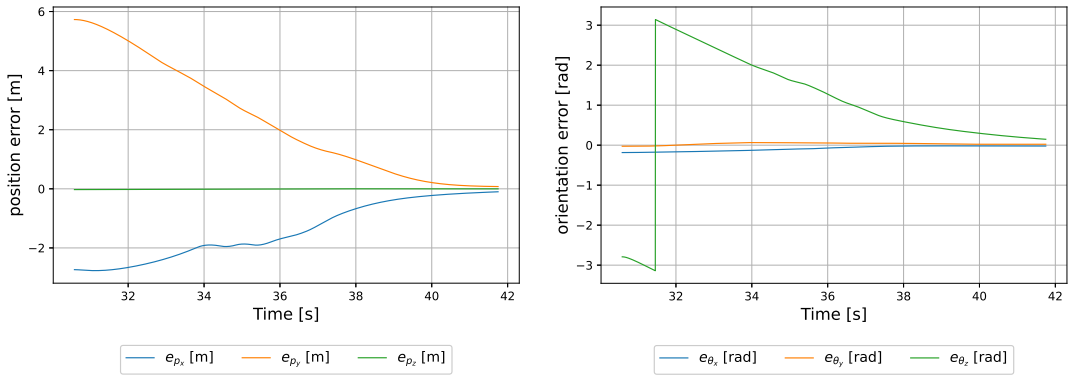

Figure 6.30: Position (left) and orientation (right) errors.

The final pose is reached with enough precision from the robot and then it stops. It is possible to see how it is influenced by obstacles, including the dynamic one which is present for a longer period of time as sensed obstacle in figure 6.32 due to its movement.

Pick & place images sequence

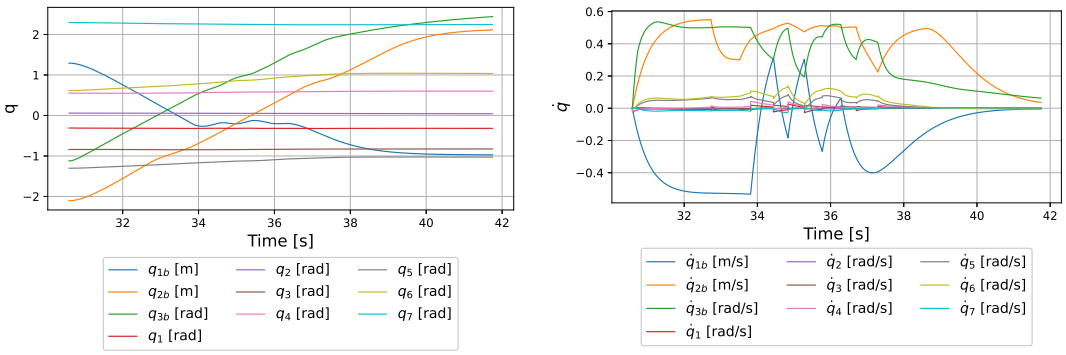

Figure 6.31: Position of the joints q (left) and velocity of the joints $\dot{q}$ (right) in configuration space.

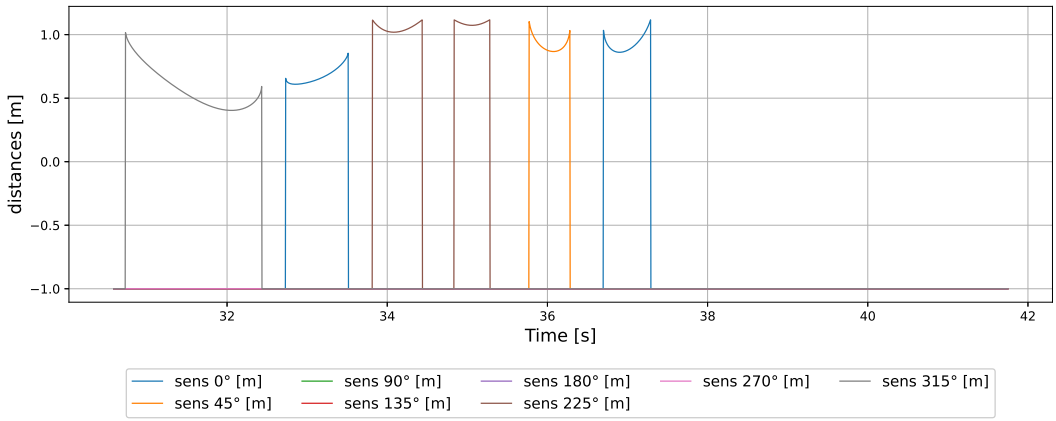

Figure 6.32: Base sensors site.

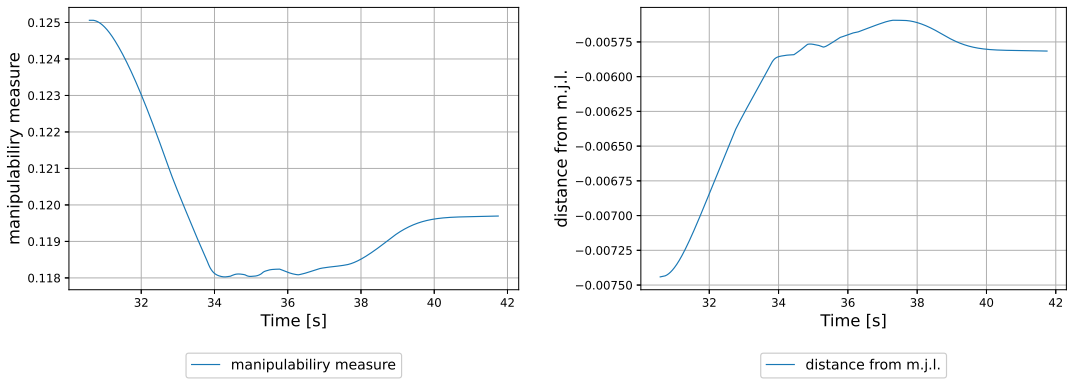

Figure 6.33: Manipulability and distance from mechanical joint limits measures of the left arm.

Once the pose is reached the end-effector opens up and the object is on the table. The task is successfully completed in about 42 s.

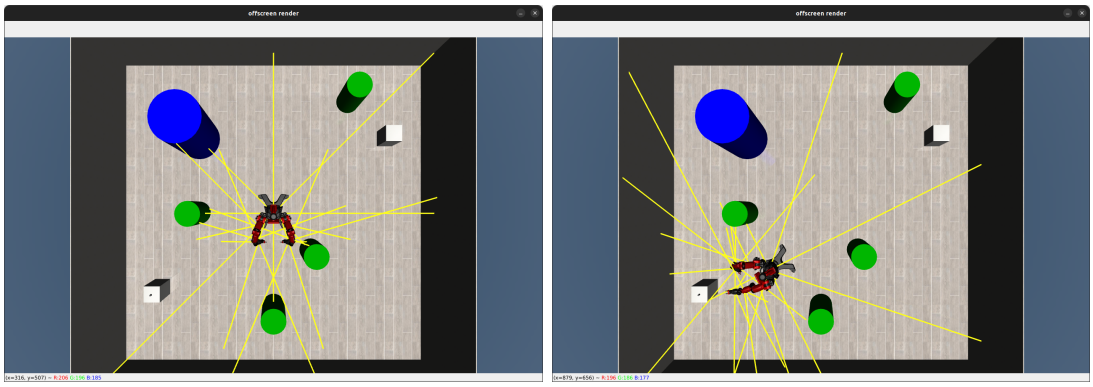

Figure 6.34: Robot moving toward the desired pick pose.

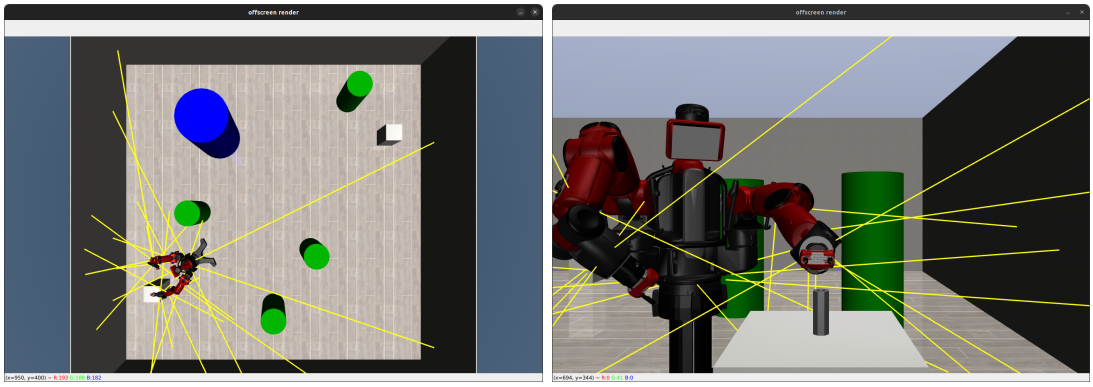

Figure 6.35: Then, the robot approaches the object once it is close enough to it.

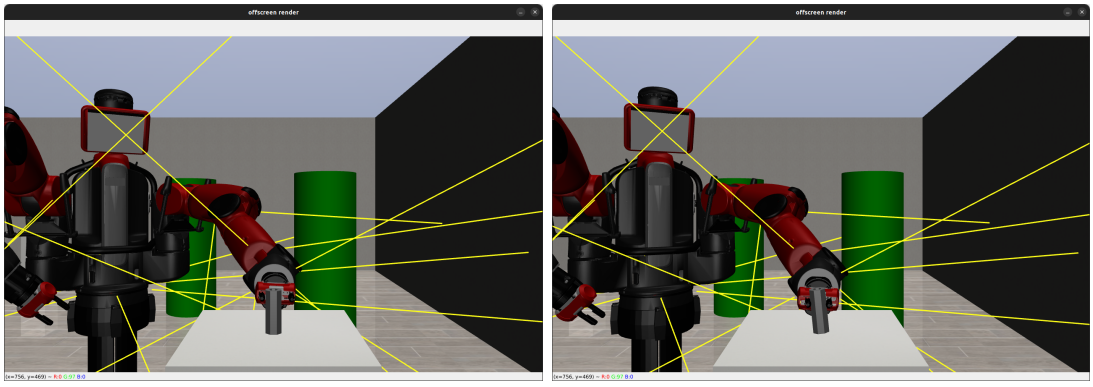

Figure 6.36: Once the gripper is closed, the robot moves toward the place pose.

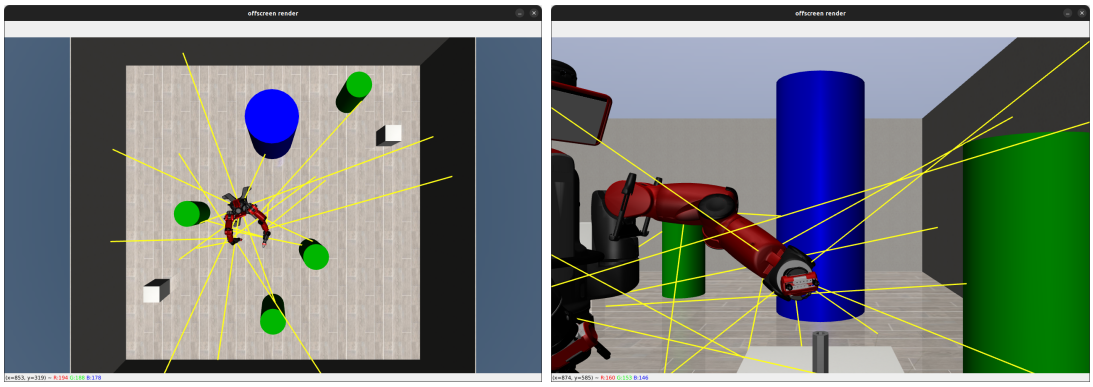

Figure 6.37: The robot moves between the obstacles and once the place pose is reached the object is left.

Chapter 7

Conclusions

7.1 Conclusions

In conclusion it is possible to see how the algorithm is able to find an optimal solution for a specific task even in unpredictable dynamically changing environments, describing the wanted performances with a cost function which must be minimized. So that, the initial objective is reached.
The order of the stack can be extended in all the cases once it is derived through learning but, the parameters of the tasks are specific for a certain fitness measure. This underlines how the result is heavily dependent on the cost function that can be defined by the user, adapting the same algorithm to a different situation and different requirements.
Despite the learning phase was handled in environments in which dynamics are not present, in order to make the simulation lighter, the derived parameters, once they are learned, work well also in the case of a full dynamics environment.
The used libraries, in Python programming language of RoboSuite and MuJoCo, revealed themselves as powerful tools for the learning development trough simulation, also with redundant robots.

7.2 Future developments

Building on the findings of this study, future research could explore a camera system that may be implemented to work in place or in collaboration with the range finder sensors for the obstacle recognition. This will allow a better understanding of distances or possible impacts with the trajectories of the robot, allowing the creation of combined trajectory between the Inverse Kinematic and

Obstacle Avoidance task. Also, other genetic programming techniques could be implemented in the algorithm, like different genetic operations to generate an offspring or variable length stacks.
To address the limitations identified in this research, subsequent studies should focus on the possibility to learn and test stacks on a real robot in real environment. This would help to understand the behaviour of the algorithm in the real world and its adaptability to true duties.
Furthermore, some aspects could be extended in order to obtain better performances. For example, if a sensor on the arm sens an obstacle, the whole kinematic chain between the robot base and the sensor itself moves, changing the end-effector orientation and position, while could be enough to slightly move the base of the robot in a certain direction. Finally, a real-time plotting system could be implemented in order to see data and performances while the simulation is still running, without waiting until the end of the simulation.

References

[1] Yuke Zhu et al., *Robosuite: A Modular Simulation Framework and Benchmark for Robot Learning*, 2020 https://robosuite.ai/

[2] Todorov et al., *MuJoCo: A physics engine for model-based control*, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems

[3] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo: *Robotics Modelling, Planning and Control*, Springer-Verlag London Limited 2010.

[4] Signe Moe1, Gianluca Antonelli, Andrew R. Teel, Kristin Y. Pettersen, Johannes Schrimpf: *Set-Based Tasks within the Singularity-Robust Multiple Task-Priority Inverse Kinematics Framework: General Formulation, Stability Analysis, and Experimental Results*, Front. Robot. AI, 18 April 2016 Sec. Robotic Control Systems Volume 3 - 2016

[5] Pietro Falco and Ciro Natale, *Low-level flexible planning for mobile manipulators: A distributed perception approach*, 2014

[6] John R. Koza and Riccardo Poli, *A Genetic Programming Tutorial*, 2003

[7] Rethink Robotics. sdk-docs. GitHub. Accessed, 2024. https://github.com/RethinkRobotics/sdk-docs/blob/master/README.md.

[8] S. Stavridis, P. Falco and Z. Doulgeri, *Pick-and-place in dynamic environments with a mobile dual-arm robot equipped with distributed distance sensors*, 2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids), Munich, Germany, 2021, pp. 76-82, doi: 10.1109

[9] Matteo Iovino, Jonathan Styrud, Pietro Falco, Christian Smith, *Learning behavior trees with genetic programming in unpredictable environments*, 2021 IEEE International Conference on Robotics and Automation (ICRA)

[10] Kévin Dufour, Wael Suleiman. *On Maximizing Manipulability Index while Solving a Kinematics Task*, Journal of Intelligent and Robotic Systems, 2020

[11] T.L. Harman and Carol Fairchild, *INTRODUCTION TO BAXTER*, Updated 2/08/2016

[12] Akshay Kumar, Ashwin Sahasrabudhe, Chaitanya Perugu, Sanjuksha Nirgude, Aakash Murugan, *Kinematics & Dynamics Library for Baxter Arm*

[13] R.L. Williams II, *Baxter Humanoid Robot Kinematics*, Internet Publication, https://www.ohio.edu/mechanical-faculty/williams/html/pdf/BaxterKinematics.pdf, April 2017

[14] Mostafa BagheriMiroslav et al., *Multivariable Extremum Seeking for Joint-Space Trajectory Optimization of a High-Degrees-of-Freedom Robot*

[15] Ròbert Krasnansky et al., *Reference trajectory tracking for a multi-DOF robot arm*, Archives of Control Sciences Volume 25(LXI), 2015 No. 4, pages 513–527

[16] L. E. Kavraki, P. Svestka, J. . -C. Latombe and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, in IEEE Transactions on Robotics and Automation

[17] Steven M. Lavalle and James Kuffner, *Rapidly-Exploring Random Trees: Progress and Prospects*, January 2000

[18] Sachin Chitta et al., *Mobile Manipulation in Unstructured Environments*, June 2012 IEEE Robotics & Automation Magazine

[19] M. C. Sinclair and S. H. Shami, *Evolving simple software agents: comparing genetic algorithm and genetic programming performance*, Second International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications, Glasgow, UK, 1997

[20] Olivier Stasse, Adrien Escande, Nicolas Mansard, Sylvain Miossec, Paul Evrard, et al.. *Real-time (self)-collision avoidance task on a HRP-2 humanoid robot*, ICRA'2008

[21] Z. Ju, C. Yang and H. Ma, *Kinematics modeling and experimental verification of baxter robot*, Proceedings of the 33rd Chinese Control Conference, Nanjing, China, 2014

[22] M. Bagheri, M. Krstić, Peiman Naseradinmousavi, *Multivariable Extremum Seeking for Joint-Space Trajectory Optimization of a High-Degrees-of-Freedom Robot*, Journal of Dynamic Systems Measurement, and Control August 2018

[23] Jesse HavilandJesse HavilandPeter Ian CorkePeter Ian Corke, *Maximising Manipulability During Resolved-Rate Motion Control*, February 2020

[24] P. Corke and J. Haviland, *Not your grandmother's toolbox – the Robotics Toolbox reinvented for Python*, 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021

[25] "Robotics Knowledgebase", *Robotics Knowledgebase*, https://roboticsknowledgebase.com/

[26] M. Galrinho, C. R. Rojas and H. Hjalmarsson, *Parametric Identification Using Weighted Null-Space Fitting*, in IEEE Transactions on Automatic Control, vol. 64, no. 7, pp. 2798-2813, July 2019

[27] Nakamura Y, Hanafusa H, Yoshikawa T., *Task-Priority Based Redundancy Control of Robot Manipulators*. The International Journal of Robotics Research. 1987