UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER'S DEGREE IN CONTROL SYSTEMS ENGINEERING

Trajectory Execution for Coordinated Dual-Arm Robotic Systems in Simulation

Supervisor
Prof. Ruggero Carli

Candidate
Alessia Garbo
2118416

Cosupervisors
Prof. Àngel Santamaria Navarro
Prof. Vicenç Puig

ACADEMIC YEAR 2024/2025

October 7, 2025

Abstract

Dual-arm robotic manipulation represents a promising solution for the automation of complex tasks that require coordination, precision, and controlled interaction with the environment. One of the most relevant applications is automated welding, where it is necessary to follow constrained trajectories while constantly maintaining controlled contact with the workpiece.

In this thesis, a dual-arm system is developed and simulated to mimic automated welding operations through precise trajectory tracking. The system is modeled in MATLAB, integrating CasADi for symbolic and numerical optimization and the Robotics System Toolbox for robot modeling and simulation.

Reference trajectories are generated during the path planning phase using the Orientation-Constrained Rapidly exploring Random Tree (OC-RRT) algorithm [1], which enables the creation of kinematically feasible and optimized paths. Trajectory execution is handled by a 12-degree-of-freedom Nonlinear Model Predictive Controller (NMPC), designed to ensure accuracy and stability even under non-ideal conditions. The system is tested in simulation under dynamic disturbances, sensor noise, and in addition with a distance constraint between the end-effectors of the two manipulators.

A comprehensive description of the system architecture, planning strategies, and control techniques is provided, along with a performance analysis in both nominal and perturbed scenarios. The results highlight the robustness and effectiveness of the proposed approach, paving the way for potential future implementations on real robotic manipulators in advanced industrial environments.

Resumen

La manipulación robótica de doble brazo representa una solución prometedora para la automatización de tareas complejas que requieren coordinación, precisión y control de la interacción con el entorno. Una de las aplicaciones más relevantes es la soldadura automática, en la que es necesario seguir trayectorias restringidas manteniendo un contacto controlado y constante con la pieza de trabajo.

En esta tesis se desarrolla y simula un sistema de doble brazo, diseñado para reproducir operaciones de soldadura automatizada mediante un seguimiento preciso de trayectorias. El sistema se modela en el entorno MATLAB, integrando CasADi para la optimización simbólica y numérica, y el Robotics System Toolbox para la modelación y simulación de manipuladores robóticos.

Las trayectorias de referencia se generan en la fase de planificación de caminos mediante el algoritmo Orientation-Constrained Rapidly Exploring Random Tree (OC-RRT) [1], lo que permite la creación de trayectorias cinemáticamente válidas y optimizadas. La ejecución de dichas trayectorias se confía a un controlador predictivo no lineal (NMPC) de 12 grados de libertad, diseñado para garantizar precisión y estabilidad incluso en condiciones no ideales. El sistema se prueba en simulación en presencia de perturbaciones dinámicas, ruido en los sensores y con la inclusión de una restricción de distancia entre los end-effectors de los dos manipuladores.

Se ofrece una descripción completa de la arquitectura del sistema, de las estrategias de planificación y de las técnicas de control empleadas, junto con un análisis del rendimiento en escenarios nominales y perturbados. Los resultados evidencian la robustez y la eficacia del enfoque propuesto, abriendo el camino a futuros desarrollos en manipuladores robóticos reales dentro de entornos industriales avanzados.

Riepilogo

La manipolazione robotica a doppio braccio rappresenta una soluzione promettente per l' automazione di compiti complessi che richiedono coordinazione, precisione e controllo dell' interazione con l' ambiente. Una delle applicazioni più rilevanti è la saldatura automatica, in cui è necessario seguire traiettorie vincolate mantenendo costantemente un contatto controllato con il pezzo da lavorare.

In questa tesi viene sviluppato e simulato un sistema a doppio braccio, progettato per riprodurre operazioni di saldatura automatizzata tramite un preciso tracciamento della traiettoria. Il sistema è modellato in ambiente MATLAB, integrando CasADi per l' ottimizzazione simbolica e numerica e la Robotics System Toolbox per la modellazione e simulazione dei robot manipolatori.

Le traiettorie di riferimento vengono generate nella fase di path planning tramite l' algoritmo OC-RRT (Orientation - Constrained Rapidly Exploring Random Tree) [1], che consente la creazione di percorsi cinematicamente validi e ottimizzati. L' esecuzione delle traiettorie è affidata a un controllore NMPC (Nonlinear Model Predictive Control) a 12 gradi di libertà, progettato per garantire precisione e stabilità anche in condizioni non ideali. Il sistema viene testato in simulazione in presenza di disturbi dinamici, rumore sui sensori e con l'introduzione di un vincolo di distanza tra gli end-effector dei due manipolatori.

Si fornisce una descrizione completa dell' architettura del sistema, delle strategie di pianificazione e delle tecniche di controllo adottate, accompagnata da un' analisi delle prestazioni in scenari sia nominali che perturbati. I risultati evidenziano la robustezza e l' efficacia dell' approccio proposto, ponendo le basi per futuri sviluppi su manipolatori robotici reali in contesti industriali avanzati.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisors, Àngel Santamaria Navarro and Vicenç Puig Cayuela, for their valuable support and great availability. Their attentive guidance and insightful suggestions have been essential to the development of this thesis and have significantly improved its quality, thanks to their deep expertise in control theory and robotics.

Special thanks go to the University of Padua and to my supervisor, Professor Ruggero Carli, for providing me with the opportunity to carry out my final project in a new and international environment, none of this would have been possible without his support.

I am grateful to the Universitat Politècnica de Catalunya for making all of this possible and for giving me the chance to experience months of academic and personal growth.

Finally, a heartfelt thank you goes to my family and closest friends, for their patient and loving support throughout my academic journey. Their presence has been a constant and invaluable source of strength and motivation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

A manipulator is a robotic arm equipped with multiple degree-of-freedom and is a fundamental component of industrial automation. Systems composed of manipulator arms allow repetitive, precise and often dangerous tasks to be performed, improving efficiency, safety and product quality.

The worldwide robot market has grown exponentially in recent years, so much so that the total number of manipulators in factories is estimated to already exceed four million and the number of installations is growing year after year. One of the sectors that benefits most from their use is certainly the manufacturing industry, which has witnessed, and is witnessing, increasing process automation, with a focus on applications that require high precision, repeatability and reduced lead times. Among these, robotic welding represents one of the most critical and complex operations, especially when it involves welding together components with irregular geometries or tight tolerances.

The major manufacturers of manipulative and collaborative robots certainly include the following companies: ABB, KUKA, Comau, Kawasaki, and Universal Robot. Specifically, the use of lightweight robotic arms such as Universal Robot's UR3s, used moreover in this thesis work, has opened up new opportunities to develop flexible, adaptable, high-performance systems. In fact, industrial automation can be argued to be an indispensable key to increasing the productivity and competitiveness of industries.

However, the study of how to achieve optimal and synchronized trajectories among multiple manipulators in a dynamic real-world scenario remains an open challenge. In particular, generating trajectories that simultaneously take into account robot dynamics, the presence of obstacles, mutual collision constraints, joint constraints, and temporal coordination is a complex problem, especially in contexts where accuracy and Model Predictive Control (MPC) are essential. In real-world environments, model-related uncertainties, variations in system parameters and the necessity of real-time adaptation dictate the adoption of advanced and robust algorithms. Furthermore, synchronization between manipulators requires that each robot not only follow its optimal trajectory, but also do so cooperatively, maintaining spatial and temporal coordination with the other. These requirements call for the integration of cutting-edge planning and control approaches.

This thesis arises from a desire to address such a challenge through the development of an advanced planning and control system aimed at improving the time efficiency and quality of the welding process in a collaborative dual-arm robotic system. The trajectory generation for the two manipulators will be handled by a planning algorithm based on Orientation-Constrained Rapidly-exploring Random Trees (OC-RRT), which is capable of producing optimal paths within kinematic and environmental constraints. Subsequently, the combined control of the two UR3s will be managed through a Nonlinear Model Predictive Control (NMPC) controller, designed to operate in real time on a 12 degree-of-freedom system, ensuring accuracy and coordination throughout the welding cycle.

1.2 Project's scope

This research project, which also constitutes this master's thesis, finds its main application in the industrial field, particularly in the manufacturing sector, where robotics is becoming increasingly widespread. This is mainly due to the fact that robotic manipulators can obtain excellent results in terms of both execution time and precision. The ultimate goal of the project is to develop a dual-arm robotic system designed for automated welding of two workpieces together.

The research focuses on the development of a complete set-up consisting of two UR3 (Universal Robots) manipulators, capable of cooperating and executing optimal trajectories. Trajectory planning is performed using the OC-RRT algorithm, which not only generates feasible paths but also incorporates kinematic and environmental constraints. For the control part, an NMPC has been designed to operate on the 12-degree-of-freedom system resulting from the combination of both robots.

The system is entirely developed and tested in a simulated environment, using MATLAB as the main computing platform, alongside advanced toolboxes and libraries such as CasADi for symbolic optimization and YALMIP for solving nonlinear programming problems. It should be noted that the scope of the thesis does not include aspects such as the integration of external sensors, environment perception, or the generation of welding machine code. To achieve the final implementation, a step-by-step approach was adopted: starting from tests on a single 6-degree-of-freedom manipulator, followed by scenarios involving two independent robots, and finally reaching the complete 12 degree-of-freedom coordinated system, thus enabling synchronized operation of the two robotic arms.

In conclusion, although UR3 robots are classified as collaborative (cobots), the developed system does not involve any interaction with human operators. Therefore, in this context, the term "collaborative" should be interpreted as referring to a cooperative multi-robot setup, in this case a dual-arm system, where two manipulators work in a synchronized way to accomplish the task: the welding.

1.3 Objectives

The objective of this thesis is to develop a dual-arm robotic system capable of efficient and accurate welding. This requires maximum coordination between the robots to perform the task accurately.

Before going into the technical details of the methodological choices made, it is necessary to formalise the problem, establish the basic structures to be used and assign the main tasks to each component of the system. To this end, several possible architectural configurations are explored.

A first proposal included a human-machine interaction and a system composed of just two UR3 robotic arms: the operator would perform a manual pre-weld by placing some tack welds, after which one of the two manipulators would transport the two pre-welded pieces to the second arm which was assigned the task of performing the final weld. Then, a second solution, more sophisticated but also more complex, introduced an external object (a box, Figure 1), considered a third arm fixed at a point in space to perform the welding, and envisaged both human-robot collaboration and the coordinated use of the two manipulators: the operator would insert a workpiece into the box, one of the two robots would bring the second workpiece closer, while the other would perform the tack welds, and finally the workpiece would be transported to the stationary welder.

However, although all these proposals have potential, in the end it is decided to adopt a fully automated solution, removing any form of human-robot collaboration. The final system comprises two UR3 manipulator arms and a third fixed welding arm, whose actions are not directly controlled. The two manipulators handle all stages of the process in an autonomous manner: part detection, gripping, trajectory planning and execution, and motion control. It should be mentioning that the focus of this work is exclusively on trajectory planning and control of the dual-arm system.

To this end, the two main techniques used are: trajectory planning using an OC-RRT based algorithm, followed by an NMPC controller operating in real time, which runs in

real time and is applied in a coordinated manner to the two UR3 robotic manipulators. In this way, optimal trajectories are obtained that take into account robot dynamics, collision constraints, joint constraints and kinematic and environmental constraints.

The combined use of OC-RRT and NMPC represents an innovative approach, as path planning and control are usually treated separately. It should also be noted that OC-RRT represents an extension of the classic Rapidly-exploring Random Tree (RRT), while the use of NMPC avoids the linearisation or simplification of the system required by traditional MPC.

The system is developed entirely in a simulated environment and, in addition to the implementation of the techniques described above, the system's performance is analysed, such as tracking accuracy, response to the introduction of disturbances and noise, and robustness to the constraints imposed.

This research is intended to provide a methodological contribution to the design of control and planning architectures that can be extended in the future to real-world or industrial settings where accuracy, reliability, and robotic coordination are critical requirements.

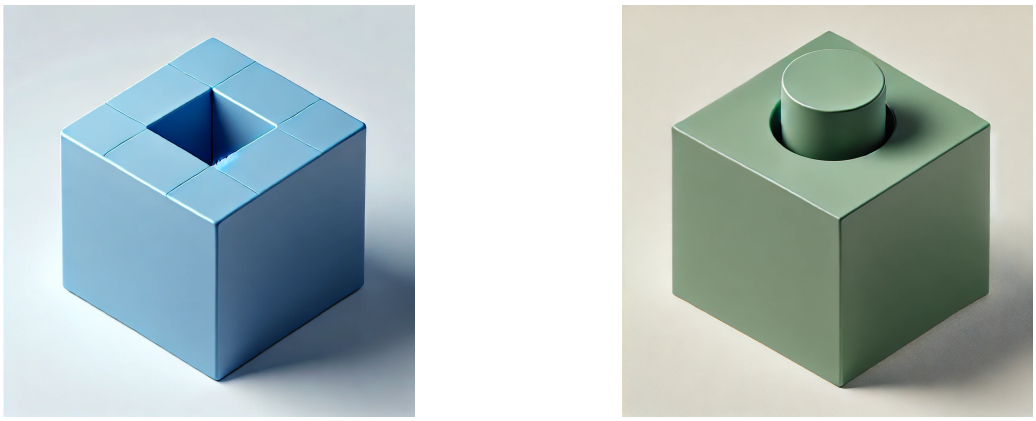


Figure 1: Special feature: Box.

2 Case study

2.1 Scenario and Contest

A dual-arm robot is a system composed by two manipulators, typically mounted on a shared base, capable of operating independently or in a coordinated way in order to perform complex tasks that usually require bimanual dexterity, spatial collaboration, or cooperative objective manipulation. A welding process requires high precision; on the other hand, high precision requires more available time: for this reason, automated welding results to be the key.

The case study here reported deals precisely with a simulation of an automated welding operation conducted by a dual-arm robotic system. Specifically, the interest is in completing this task through the use of two UR3 manipulators from the Danish company Universal Robot. These are used for the purpose of guiding the two workpieces to be welded together along a predefined trajectory in a coordinated manner, while at the same time having to respect the constraints of the system and an additional distance constraint between the end-effectors. This is definitely an industrial scenario, particularly a manufacturing one. With the integration of such a system, it is easy to see how automating complex tasks, such as welding, that require manual dexterity, makes the application context much more advanced and able to achieve more and better results in terms of production.

It is important to remember that this case focuses exclusively on the movement phase, as well as the welding of the workpieces, namely on the planning trajectory and control of the system parts. It is therefore assumed that techniques and data concerning the previous phases are already available: recognition of the workpieces to be picked and their orientation (i.e. detection), picking of the objects (i.e. grasping) and initial positioning of them. In addition, there is no consideration of the welder itself, considered as a fixed third arm external to the system under consideration.

The system is fully developed in a simulation environment and the simulation environment involves the two UR3 to be mounted on a fixed position and cooperating to guide the workpieces along the target trajectory. In particular, the robots are placed close together, with a maximum distance between the two bases set to $41.7\ cm$, on a table where the two pieces to be welded are located. Their initial position considered is therefore a rest position, with end effectors at equal height, each placed in front of the workpiece on which the pick is to be made. The aim is to first plan an optimised trajectory for the

manipulators, more specifically one for each robotic arm. This must be achieved in a coordinated manner. It is known from the literature that coordination of two manipulators during the manipulation and welding of two parts is one of the most interesting challenges in collaborative robotics. To achieve effective coordination, therefore, the trajectories must first be carefully designed. In this thesis, among the several strategies for trajectory planning, two methods will be introduced and developed: firstly, the RRT is presented but then an advance version of it will be chosen for the planning part, namely the OC-RRT. The actual coordination is then implemented through the control strategy, which is why it is important that the desired trajectories are designed in such a way as to facilitate the control action on them, maintaining a certain harmony in the movements of the two manipulators throughout the task. This implies, among other things, ensuring temporal consistency between movements, avoiding abrupt changes in speed or direction to ensure that the robots reach the target effectively.

Only after these trajectories have been defined is it possible to move on to the design and implementation phase of the control system. At this stage, the aim is to translate the desired trajectories into real and synchronized actions performed by the two robots. After trying the MPC architecture, it is chosen to use an NMPC-based strategy that is capable of generating fluid and coordinated movements, respecting constraints. This enables the system to take into account dynamic variations, external disturbances, and potential uncertainties in real time. In this way, both robots are guided along their respective trajectory, while maintaining coordination and avoiding constraint violations.

Focusing therefore on the constraints to be met, there are joint limits, velocity bounds, and workspace constraints to avoid collisions, ensuring the feasibility of the motion and the safety of the interaction. Moreover, an additional distance constraint is added to deal with a more realistic problem, firstly on the trajectory planning part and consequently on the controller part. This last constraint involves a range distance between the two end effectors to be respected: it is requested to be maintained in a range defined as $[0.03, 0.10]\ m$.
Lastly, to ensure the reliability of the system in real-world conditions, it is essential to prove the robustness of the system to external disturbances, modeling uncertainties, and potential deviations in the desired trajectory. The design of the system can be then considered completed once, even in this scenario, it is able to maintain consistent performance.

Going into the details of the assumptions of the model under consideration, it is worth mentioning and emphasising that the physical specifications of the objects to be welded,

such as shape, material, orientation, are not taken into account: they are considered as identical rigid bodies and represented in the simulations as cylinders placed on a table. Furthermore, the simulation environment comprises, besides the two UR3 robots, only the table and the two objects, thus the environment can be considered static and free of further obstacles. This simplification makes it possible to focus exclusively on the kinematic and dynamic aspects of robot manipulation and cooperation. However, it should be remembered that although the environment is simple, the system must ensure no self-collision between arms or between an arm and the object. Finally, as already mentioned, joint limits are imposed in terms of position and velocity, as well as torque on each joint to remain within feasible operational bounds. Recall that the grasping and detection parts are not included in this solution.

A summarizing block diagram representing the workflow and the various strategies implemented for each phase is now presented (Figure 2). By using the developed architecture it is desired to obtain a simulation able to check different features for the system, such as: the quality of the trajectory and of the tracking, the collision avoidance between the robots and with the environment, the behaviour under noises and disturbances, the compliance with the distance constraint, and the tracking errors. The implementation of each block will be described and presented carefully in the subsequent sections.

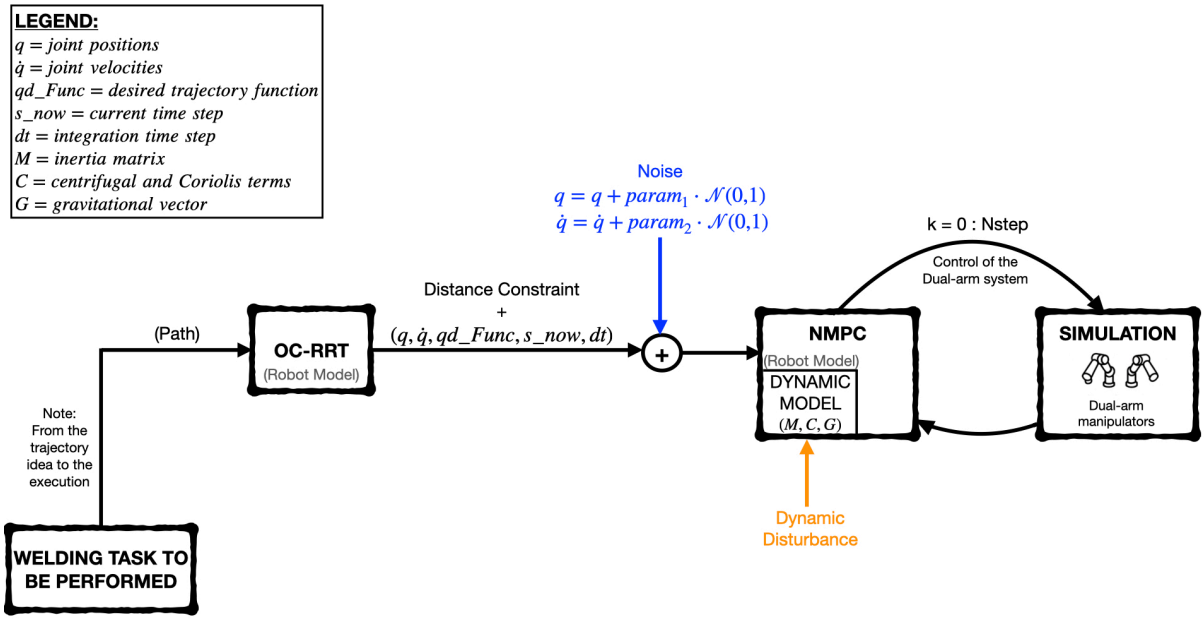


Figure 2: Workflow Block Diagram

2.2 State of the art

The number of dual-arm robots present in industries, as well as in homes, is growing day by day, mainly because their use allows for time optimization and precision that humans often cannot achieve. Indeed, the tasks to be performed are often inspired by human bimanual skills, and the environment in which the robots' movements are planned is often originally designed for human use. Thus, applications range from domestic ones, such as laundry folding, to manufacturing, such as the automated welding case considered here, and even to highly specialized fields such as space missions [2]. Dual-arm robots can also be classified not only on the basis of the task they perform, but also on how they accomplish it: either in a coordinated manner or not.

In general, the trajectory execution for dual-arm systems involves two key aspects: motion planning and control. In the literature, several surveys provide a comprehensive classification of existing techniques for both.

In the context of motion planning for dual arm manipulators, various methods have been proposed, in which they must address high-dimensional spaces, colissions avoidance and kinematic coupling. Reviewing the *Motion Planning for Robotics: A Review for Sampling-based Planners* article [3] it can be stated that the main approaches for it are divided into Global or Local motion planning. While the first calculates an entire path from the starting point to the end point, taking into account the entire known environment, the second calculates movement in real time, considering only the local environment (i.e. close to the robot). Both share some phases, which are:

- Environmental Modeling
- Optimisation Criteria
- Trajectory Optimisation
- Path Search

But they have different approaches. In the Global planner, once the path search phase is reached, three main categories arise:

- Spatial Partitioning Graphs Approach: they divide the space in cell or graph and the main algorithms are called Visibility Graph, Voronoi Graph and Grid-based Graph.

- Decomposition & Sampling-Based Methods: they are based on sampling and the main algorithms comprises Cell Decomposition, RRT and PRM (Probabilistic Roadmap Method).

- Graph-Based Search Algorithms: they are classical algorithm to find path in a graph and algorithms as Dijkstra, A* and D* (which stands for dynamic) are examples of it.

- Bio-Inspired & Meta-Heuristic Optimization: they are algorithms inspired by nature and artificial intelligence such as Simulated Annealing, Genetic Algorithm, Ant Colony Optimisation and Particle Swarm Optimisation.

On the other hand, considering Local planner, reactive planning techniques need to be cited, which are algorithms that react dynamically to the local environment, as for example:

- Artificial Potential Field

- Behavior Decomposition Method

- Case-Based Learning Method

The focus is now moved on the Sampling-based planners (PRM, RRT, RRT*) due to their probabilistic completeness and ability to tackle high-dimensional configuration spaces effectively. Within this family, the RRT excels in speed, efficiency and ease of implementation, while its variants are particularly appealing because of their efficiency in high-dimensional and dynamic environments. The most important are:

- RRT-Connect: improves convergence by growing trees from both the start and goal [4].

- Informed-RRT*: achieve path quality improvements via rewiring mechanisms [5]

- OC-RRT: integrates optimal control within RRT, ensuring dynamically feasible, constraint-compliant trajectories (which denote a crucial advantage in dual-arm scenarios).

Focusing now on coordinated control techniques, the Master-Slave approach stands out, together with an MPC control algorithm that allows the future behaviour of the system to be predicted. This combination minimises errors and improves system stability as it allows to consider the combination of robot and picked object as an ideal rigid

body, and also ensures smooth and coordinated movements of the dual-arm robot [6]. Moreover, other techniques are here reported based on scientific articles:

- PID and impedance control: remain popular for their simplicity and real-time responsiveness [7].
- Hybrid force/motion control: addresses contact-rich tasks but requires accurate force sensing and tuning [8].
- Master–Slave control: enables load sharing between arms, effectively linearizing the coupled system [6, 9].
- Model Predictive Control (MPC): evaluates future behavior while respecting constraints [10].
- Nonlinear MPC (NMPC): extends MPC by handling nonlinear dynamics, constraints, and disturbances in real time, offering ideal predictive coordination for dual-arm tasks [11, 12].
- Advanced recent frameworks: like STORM integrate constraint optimization and robustness for reactive manipulation [13].

In addition, there are various other control, modeling, and learning techniques aimed at dual-arm manipulators, and challenges remain to develop increasingly precise and efficient methods that manage system stability and synchronization. The most advanced techniques include visual systems, force sensors, and more recently, adaptive learning and AI integration [8].

The case under consideration concerns the planning and control of an automated welding between two workpieces performed in a coordinated manner, without considering the sorting, identification, location or grasping pose detection part of the various workpieces [14]. Thus, the first thing to be implemented is trajectory planning. In [15], Dual Robot Coordinated Welding Trajectory Planning is discussed and a Master-Slave approach is again proposed to ensure coordination in movement. For trajectory planning, on the other hand, useful interpolation methods are tested to obtain precise trajectories.

Going in more depth on the implementation level, the numerical computing software MATLAB proves to be essential for solving the research problem. In particular, through the installation of the Robotics System Toolbox, which includes algorithms for collision control, path planning, trajectory generation, as well as both forward and inverse kine-

matics and dynamics, it is possible to obtain an initial version of the dual-arm robotic model and a first trajectory planning using inverse kinematics. Therefore, the simulation scenario is completed and the two predefined UR3 models are loaded.

The starting point has thus been reached, but it is certainly not enough. In addition to obtaining a trajectory, it is also important to avoid collisions between robots. The first strategy considered is one that exploits motion planning and execution through an approach that exploits trajectory reservation [16], resulting in an efficient collision free during task execution, although considering asynchronous motion between manipulators. A comparison of different constrained motion planners is presented in [17]. What is important to focus on, however, is the introduction of a new algorithm, namely OC-RRT: the main objective is to generate collision-free paths between the initial and final configurations while respecting the orientation constraints of the end-effectors. This algorithm is characterised by efficiency and accuracy and is a combination of sampling-based planning methods and inverse kinematics. Finally, another interesting feature is the B-spline curve, which can improve the quality of the initial path planning, transforming it into a collision-free and smooth trajectory that respects the kinematic constraints of the robot [1]. Thus, the trajectory planning part is concluded: It is decided to use the innovative OC-RRT algorithm, which also takes into account the dynamics of the system under consideration and in which constraints can be added to the system. A new feature is then added to this same algorithm, namely the interpolation of optimized paths, with the application of spline, in order to obtain a continuous, smooth trajectory.

The next step is to achieve coordination between the two manipulators and, above all, to study the most efficient control method for the case under investigation. As presented in [9], it is possible to implement coordinated control for the two robotic arms based on an exact linearization of the system and output decoupling through appropriate nonlinear feedback, so that the two arms can transfer an object along the desired trajectory while holding it at the two ends. Howerver, over the years, more advanced control techniques have emerged. In fact, as shown in [10], model-based control, data-driven models that use data collected during operation, and NMPC are innovatively integrated, thus improving tracking performance and achieving greater speed and accuracy. The most technical and main aspects of NMPC are detailed in [11], and this paper also serves as basis for [12], which presents the Model Predictive Path-Following Control algorithm (MPFC). This algorithm relies on the receding horizon principle to provide real-time control capable of respecting joint, velocity, and workspace con-

straints; moreover, it can react to external disturbances or uncertainties and facilitates coordination between the two robotic arms without imposing rigid timing, resulting in a robust control strategy.

The strategy used takes into account all these theoretical notions and is based on the approach used in [18]: a combination of RRT* as global planner, Spline interpolation to obtain continuous path, and finally an MPC as local planner. In the case under consideration, it is decided to replace the RRT* with an OC-RRT as global planner, maintain interpolation via Spline and finally replace the linear MPC with an NMPC. The implementation exploits both Yalmip, a MATLAB-based toolbox for formulating and solving symbolic optimisation problems, and CasADi (Computer Algebra for Automatic Differentiation), a library for numerical optimisation and symbolic computation that is very good for non-linear optimisation and high-performance numerical algorithms. The entire system is formulated step by step, starting with a single-arm and using RRT, Spline and MPC and then gradually extending it to the case under study: double-arm with OC-RRT, Spline and 12 degree-of-freedom NMPC, where additionally a constraint on the distance between the two end-effectors and robustness to noise and dynamic disturbances are added.

As a final step, a possible future modification to the implemented control structure is also analysed. Exploiting the notions of [13] a further improvement of the strategy could be to add even more constraints to the model and have these constraints directly included in the cost function. Using the STORM framework, one can include desired robot behaviours or constraints in a batched manner by simply using a weighted sum and still obtain robustness of the MPC (or NMPC) to which is added the flexibility to encode different desired behaviours or constraints.

To summarise, it can be said that the study and development of robot-dual arms are very important. These manipulators are a key technology for the future of industrial robotics and beyond: they are increasingly efficient at performing tasks requiring skills that are closer and closer to those of humans with time and accuracy that humans can hardly achieve. Despite progress in recent years, however, there are still many challenges concerning effective coordination, safety and robustness of the systems in real, dynamic environments. This is why artificial intelligence has also started to be integrated into the most innovative solutions.

2.3 Theoretical Notions

2.3.1 Basic Concepts of Rapidly-exploring Random Trees (RRT)

The first problem to be addressed and considered in this thesis is path planning: the robot must be able to move from point A to point B while avoiding environmental obstacles and, if present, potential collisions with other robots. In the case under consideration, the OC-RRT algorithm will be used, which is a novel version of the RRT algorithm. For this reason, it is important to introduce the basic concepts of RRT, an algorithm that is able to create a graph and find a path for the robot, but be aware that this path is not necessarily optimal. To ensure an optimal solution, it is recommended to use a modified version of this algorithm called RRT* [19, 20]. RRT is a very popular motion planning algorithm, especially for complex, high-dimensional systems, such as robotic arms. It incrementally builds a tree to quickly explore the configuration space. Specifically, what the RRT algorithm does is build a tree using random sampling in the search space. To do this, an initial state z_{init} is defined, which is the starting state from which the tree begins to expand, as well as z_{goal}, the target state. The goal is to find a path between these two points by expanding the tree iteratively. Now let us define the configuration space Z. At each iteration, a random sample z_{rand} is selected: if this lies within a region free of obstacles, the algorithm searches for $z_{nearest}$, the nearest node that respects the parameter ρ, which is the incremental distance. At this point, the tree expands by connecting z_{rand} to $z_{nearest}$. If, however, z_{rand} is not accessible from $z_{nearest}$, the algorithm returns a new node , z_{new} and connects , z_{new} to $z_{nearest}$. This process continues until a configuration near the goal is found or, alternatively, until a fixed number of iterations has been reached or a predefined time limit has expired. Node expansion process is described in Figure 3 and the pseudocode for RRT is presented below:

Code 1: RRT Algorithm, T=(V,E) <–RRT(z_init)

```
1  T <-- InitializeTree();
2
3  T <-- InsertNode(NULL, z_init, T);
4
5  for i=0 to i=N do
6  z_rand <-- Sample(i);
7  z_nearest <-- Nearest(T, z_rand);
8  (z_new, U_new) <-- Steer(z_nearest, z_rand);
9
10  if Obstaclefree(z_new) then
```

```
11   T <-- InsertNode(z_min, z_new, T);
12
13   return T
```

where the function *Steer* is the one that provides a control input $u[0, T]$ which drives the system from $z(0) = z_{rand}$ to $z(T) = z_{nearest}$ along the path $z : [0, T]$.
Therefore, the procedure can be summarised in four main points:

1. Generate a random point in the space.
2. Find the nearest node in the tree.
3. Extend the tree towards the random point.
4. If the extension is valid (no collision), add the new node.

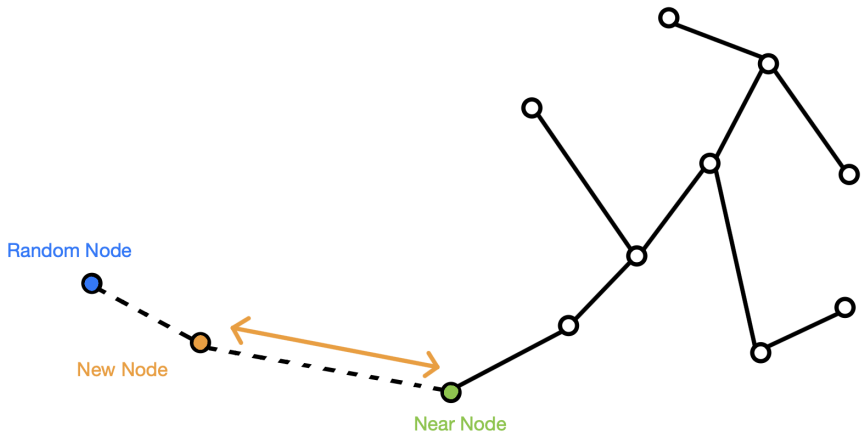


Figure 3: Node Expansion Process.

Hence, it is concluded that RRT is an efficient, flexible and simple solution to solve the path planning problem.

2.3.2 Basic concepts of Model Predictive Control (MPC)

Regarding control techniques, in this case a NMPC is employed, which represents an extension of the MPC framework to systems exhibiting nonlinear dynamics. Unlike

classical MPC, which relies on linear models and optimization problems (generally quadratic ones), NMPC manages nonlinear models and solves nonlinear programming problems at each time step, offering greater accuracy and flexibility at the expense of higher computational complexity. For this reason, the following subsection presents some fundamental concepts of MPC along with a brief description of its key principles and control scheme.

Essentially, MPC is an optimal control strategy that utilizes the mathematical model of the system to predict future behavior starting from the current state over a specified prediction horizon P, composed of a certain number of future time steps. A cost function is defined to penalize deviations from the desired state while considering possible constraints. At each time instant, a real-time optimization problem is solved to determine the optimal sequence of control inputs. The updated state is then acquired at the subsequent step, and the process is repeated continuously. This iterative cycle ensures responsiveness and adaptability to changing conditions. Figure 4 illustrates the MPC-based control scheme.

Similarly, just as NMPC is an extension of MPC, MPC itself can be seen as an evolution of the optimal Linear Quadratic Regulator (LQR) [21, 22], where physical constraints related to system inputs, states, and/or outputs are also taken into account. This addition requires solving the optimization problem online, which demands high performance numerical solvers specifically tailored to the problem.

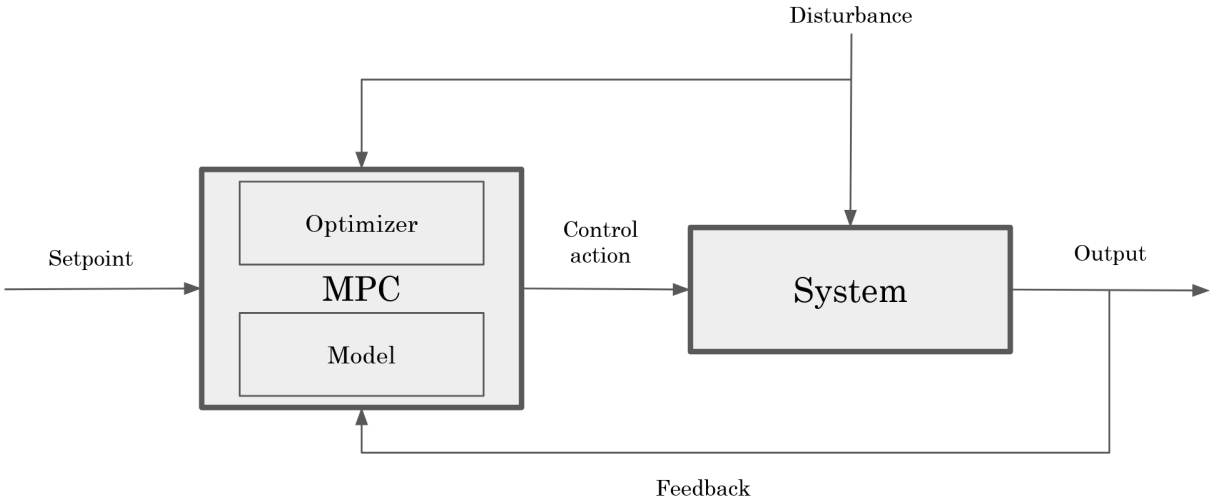


Figure 4: Simple MPC scheme.

In general, an MPC optimization problem can be formulated as follows:

$$\min_{\mathbf{u}} \quad \sum_{k=0}^{N_p-1} \left(\|\mathbf{q}_{k+1} - \mathbf{q}_{\text{ref}}\|^2_{\mathbf{Q}} + \|\mathbf{u}_k - \mathbf{u}_{\text{ref}}\|^2_{\mathbf{R}} \right) + \|\mathbf{q}_{N_p} - \mathbf{q}_{\text{ref}}\|^2_{\mathbf{P}} \,,$$

subject to:

$$\begin{aligned}
&\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k), \quad k = 0, \ldots, N_p - 1 \quad \text{(System dynamics)} \\
&\mathbf{u}_{\text{min}} \leq \mathbf{u}_k \leq \mathbf{u}_{\text{max}}, \quad k = 0, \ldots, N_c - 1 \quad \text{(Input constraints)} \\
&\mathbf{q}_{\text{min}} \leq \mathbf{q}_k \leq \mathbf{q}_{\text{max}}, \quad k = 1, \ldots, N_p \quad \text{(State constraints)} \\
&\mathbf{q}_0 = \mathbf{q}_{\text{feedback}} \quad \text{(Feedback condition)}
\end{aligned} \tag{1}$$

Where q_{ref} is the desired reference state, Np is the number of time steps of the prediction horizon and finally Q and R are the weights of the control objectives in the cost function in quadratic form. Looking now more specifically at the problem, it can be seen that at each time instant, the system, considering a future prediction horizon, calculates the optimal control inputs u to achieve q_{ref}. In order to do this, the cost function is optimised, which represents how far away from the desired state it is. During the optimisation, the dynamic rules of the system, the constraints to be met and the feedback provided by sensors or state estimators are also taken into account for each step k of the prediction horizon. Looking at the cost function, one can also see an additional term, the last one, a terminal cost; in the same way, further constraints can be added to the optimisation problem formulation. Recall that the process is repeated at each time step, each time analysing an entire prediction horizon along the trajectory, so that the control inputs are continuously adapted optimally. Figure 5 is a complete example of an MPC for a one-input, one-output system. In the upper part, the past and future trajectories are shown, and it can be seen that the state actually follows the reference r within a finite time horizon P. In the lower part of the figure, on the other hand, the trajectory of the input is shown: specifically, the commands applied in the past are shown in black, and those that will produce the estimated evolution visible in the upper part are shown in white.

Finally, it should be noted that, as it is often necessary to work with complex dynamics, such as in the case of robots, it is necessary to use NMPC. However, if NMPC is applied to a linear model, an MPC will still be obtained.

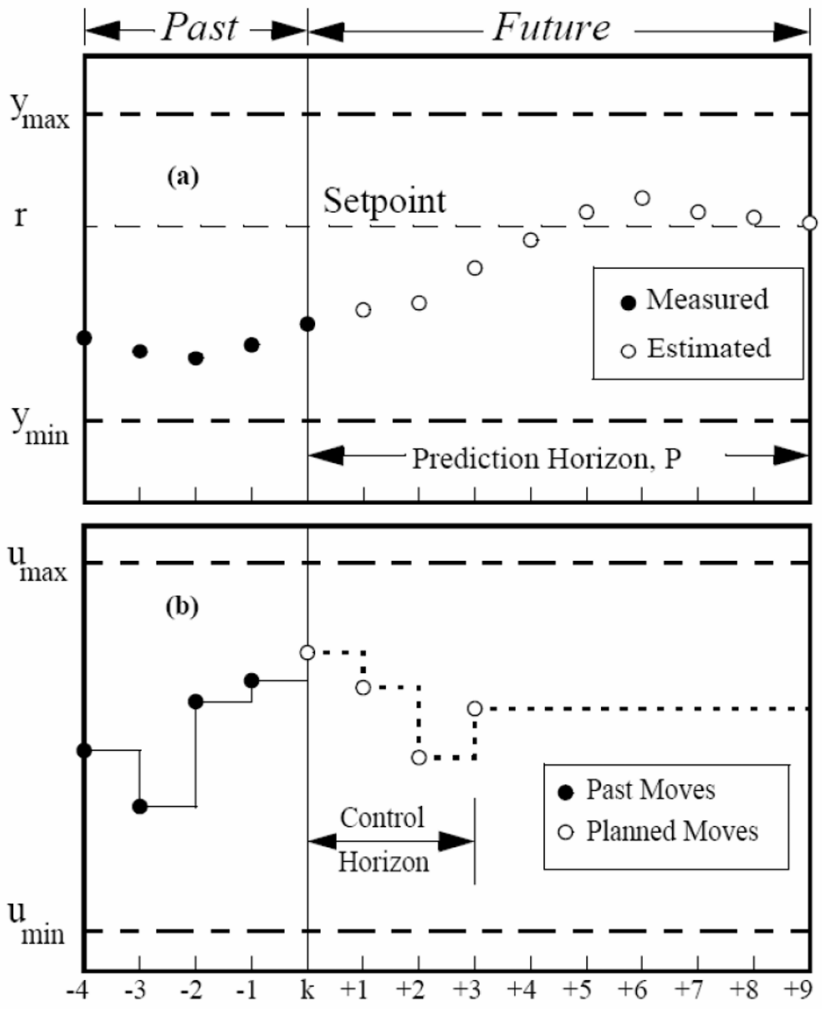


Figure 5: Operational principles of Model Predictive Control. Source: [23]

2.4 Main Software Tools and Simulation Setup

The project is developed and tested entirely with the programming and numeric computing platform MATLAB [24]. The two UR3 robotic arms used are described in a universal manner via an official *.urdf* file that contains values such as masses, inertia, links' lengths and other characteristics of the robots. They are then loaded into the static simulation scenario each attached to a separate base, and oriented to grasp the respective object. Subsequently, various codes are implemented, as well as many auxiliary functions: using a step-by-step methodology, the best possible solution for this system has been found, adding from time to time details aimed at reaching continuous implementation improvement.

As another thing, it should be noted that the use of the Robotics System Toolbox is essential [25]: from the simple loading of the robots, it proved to be an essential tool

for modelling manipulators, calculating direct and inverse kinematics, and working on trajectories. When the time to set up the control problem arrives, two other tools are essential. Firstly, Yalmip, a MATLAB toolbox for formulating optimisation problems using symbolic variables, is introduced [26]. Thanks to this toolbox, simulation times can be reduced. Unfortunately, however, one limitation found is that it is not able to treat the system under consideration using its complete non-linear dynamics, but only under certain model approximations that make it linear. To avoid this problem, therefore, CasAdi was introduced, which is an external library with a MATLAB interface for symbolic numerical optimisation and optimal control, especially oriented towards Predictive Control (also non-linear) and non-linear optimisation for complex dynamic models [27]. It is used to define the dynamics of the model and complete the solution to the proposed problem using *'ipopt'* [28] as optimisation solver.

The following setup not only aims at verifying and demonstrating the robustness of the system and enables the analysis of trajectory accuracy, synchronisation between robots, control robustness and compliance with constraints, but also to achieve good computational efficiency.

Finally, to conclude, it is interesting to mention that the choice of a case study based on automated welding with dual-arm robot manipulators stems from two main reasons: firstly, it has direct industrial applicability as welding is a key process in many production chains. On the other hand, it presents a technical complexity not to be underestimated since the simultaneous coordination of two arms, the planning of smooth trajectories and the implementation of predictive control represent a multidisciplinary challenge involving robotics, optimisation and system dynamics.

3 Design

3.1 Setup & Initialization

After having presented in detail the objective in Subsection 1.3, the case study in Section 2, and the most advanced techniques available today for this type of problem in Subsection 2.2, it is now time to go into more technical detail of the design and methodological choices made.

Recall that the operational scenario consists of two robotic manipulator arms and a third, fixed welding arm placed at a specific location within the environment. The welding arm's actions are not actively controlled and are therefore not taken into account in this study. Instead, the focus is placed entirely on the coordination and control of the two UR3 manipulators.

To begin with, it is important to load, or eventually create from scratch, the robot model. Hence, the Robotics System Toolbox, already presented previously, is employed and it allows to build test scenarios and use reference examples to validate common industrial robotics applications. It also features a library of commercially available industrial robot models that can be imported, visualized, simulated, and used with reference applications.

A very first implementation is the one in which the robot is created from zero: it is composed by three links plus the end-effector, and three joints that are revolute except for the last one, which is the one of the end-effector that is fixed. To obtain it, a rigid body object need to be create together with a joint, which need to be assigned to the rigid body. After that, a rigid body tree can be defined along with a base coordinate frame to attach bodies to, indeed the first body is added to the tree. Then, it's time to create the second body link, define its property, attach it to the first rigid body and finally specify the transformation relative to the previous body frame. Lastly, using the same procedure the end-effector is added as well. In conclusion it is necessary to generate a configuration for the robot and after that the visualization of it appears as in Figure 6, where each link is represented by a blue line, while the joints, the base, and the end-effector are also displayed.

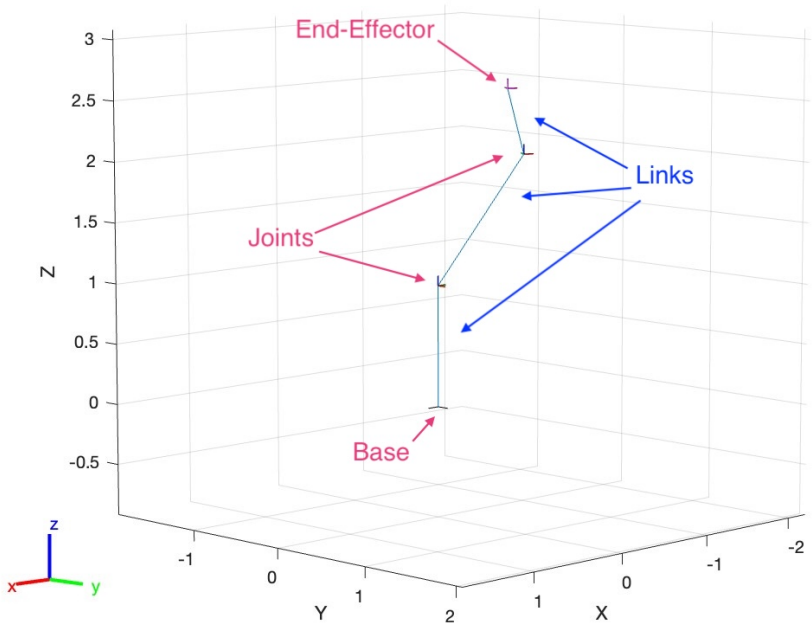


Figure 6: A first very simple visualization.

However, since this result is definitely not enough to carry out the desired implementation, it is decided to load predefined robot models. Robots with 6 degree-of-freedom, which describe the number of independent movements the arm can make, are now of interest, especially the ones in which the degree-of-freedom (DOFs) refer to movements in rotational joints. Therefore the focus is on the anthropomorphic robotic arm, a type of industrial robot whose structure and movements mimic the characteristics and functions of a human arm.

The first question to be addressed is therefore which robot model to choose in order to get to the heart of the implementation. It is decided to opt for the manipulator arm model UR3, from Universal Robot [29], which visualization is reported in Figure 7. This is an excellent choice as it was created specifically for light assembly tasks and automated workbench scenarios. It weighs $11\ kg$, has a payload of $3\ kg$ and is excellent for applications requiring 6 degree-of-freedom. In addition to its technical characteristics, the choice of the UR3 is also motivated by its availability in the university's laboratories, a factor that paves the way for future experimental implementations in real environments. In any case, several other models can be selected, such as the UR5 [30], ABB's IRB 120 [31] or even the KUKA LBR iiwa 14 [32]. These three types of robots

also allow for complex spatial movements, enabling the robots to reach a large working area and adapt to various handling tasks, mimic as well the movement capabilities of the human arm.

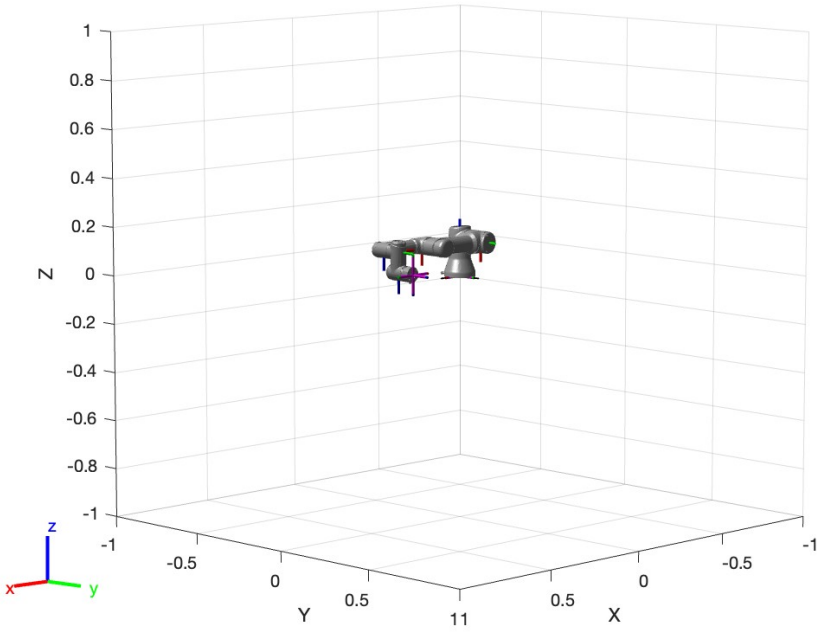


Figure 7: Load Predefined Model: UR3 Robotic Arm

After obtaining a robot in the scene, an initial test of planning a reaching trajectory with kinematic constraints is carried out. In order to develop a reaching motion for the robot under consideration there are some key steps to follow:

- **Robot Model Import & Setup:** Load the *.urdf* model and a start configuration.
- **Target Object Definition:** Define a cup with specific dimensions and position, which represent a workpiece.
- **Inverse Kinematics Calculation:** To get the joint configurations based on the desired end-effector position.
- **Visualization of the Robot & Target Object:** Display the robot and the cup in 3D.
- **Visulization of the trajectory:** Visualize the robot executing the trajectory and the trajectory itself followed by the robot.

and what is finally obtain is a good realization of this first motion for the manipulator arm. Figure 8 shows the UR3 after having performed a parabolic trajectory (depicted by the green line) while also carrying the workpiece (the orange object).

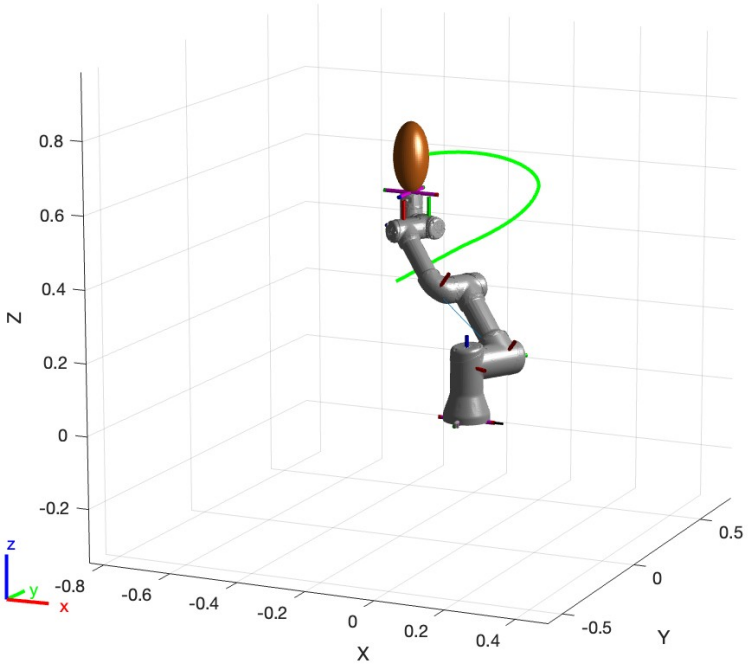


Figure 8: First Simple Parabolic Trajectory.

However, the fact is that the solution to be achieved with this work involves two robotic arms. An initial implementation tested was to use two arms built on the same base. In this way, by again applying a parabolic type trajectory, based simply on inverse kinematics, to the system consisting of two robotic arms, the result presented in Figure 9 are obtained. It can be seen that the two end-effectors follow two identical but shifted trajectories (red and green line) maintaining a constant off-set between the two end-effectors. A note with respect to trajectories is that they exploit waypoints, a common approach when it is desired for the robot to follow a complex and fluid trajectory that cannot be easily described by only a few discrete points. In other words, waypoints are specific positions in space that the robot must reach sequentially to complete a path: even if the start and end points of the trajectory are known, intermediate points are given along it in order to facilitate the robot's movement.

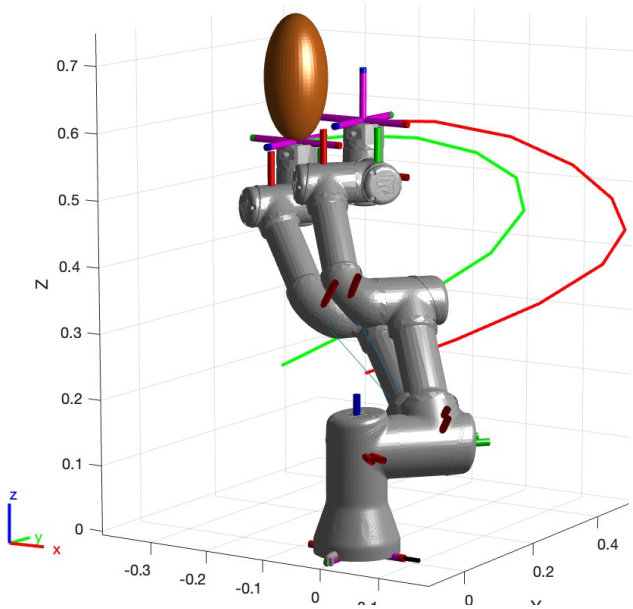


Figure 9: Two arms with the same base executing coordinated trajectories.

However, in order to avoid having too many collision problems between the arms from the very beginning, it is decided that the best idea is to keep the two manipulator arms separate, namely on two separate bases, and then to develop a 12 degree-of-freedom controller to treat them as a single robot. In conclusion, the framework used from now on is the one of Figure 10 in which the two UR3 are visualized, each one with its base.

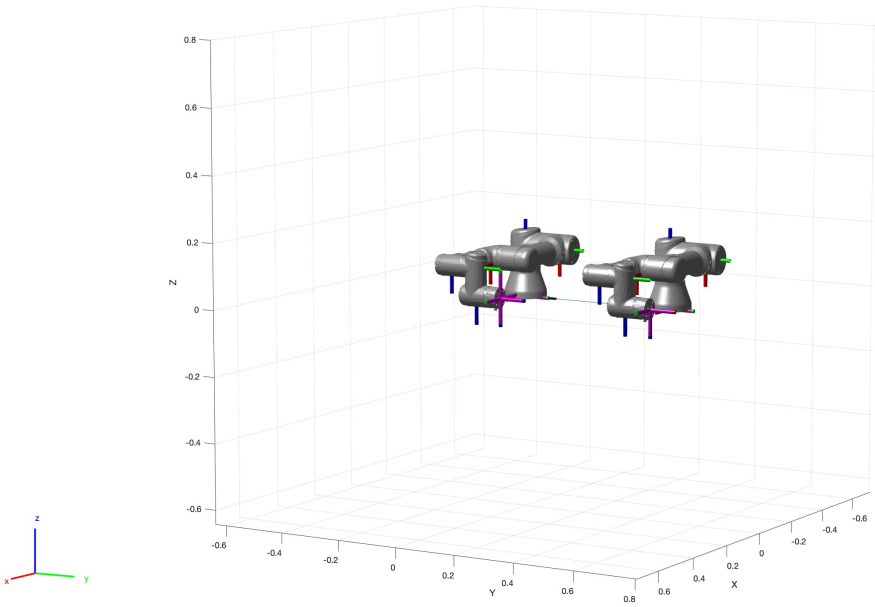


Figure 10: Two robots with two different bases.

3.2 Trajectory Planning

Next comes the part where the focus is on trajectory planning, and in this subsection the methods used are analysed. Trajectory planning is a very important step in robotic manipulation, especially in cooperative multi-arm systems where coordination and accuracy are essential. The goal is to calculate feasible and optimised paths for end-effectors that respect geometric constraints, kinematic limits and task requirements.

In this study, two approaches are explored: first using the RRT, which is effective in high-dimensional spaces and cluttered environments, and then moving on to a more sophisticated optimisation-based strategy, namely the OC-RRT, an improved extension of the previous approach. These methods make it possible not only to generate collision-free paths, but also to integrate additional cost functions and constraints to produce the desired trajectories. The combination of planning and control is designed to ensure synchronised motion and adherence to the desired trajectory, leading the way to robust and reliable dual-arm manipulation.

Theoretical notions, implementations and results obtained using both strategies will be presented below, and it is also important to specify that the end-effector distance constraint will be added only in the case of OC-RRT since in the end it will be the algorithm selected.

3.2.1 Rapidly-exploring Random Tree (RRT)

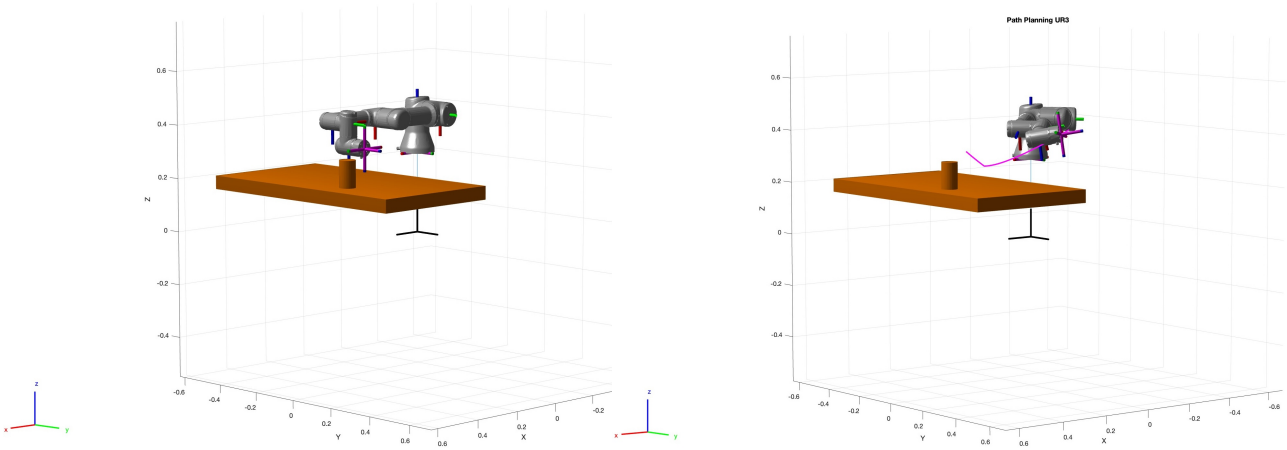
As has been mentioned, the first trajectories created are obtained by means of RRT, an algorithm whose basics have already been presented in Subsubsection 2.3.1. The complexity of the implementations and simulations will increase step by step, in fact the first trajectory with RRT presented is the one for a single robotic arm.

After loading the UR3 robotic model, the table and the corresponding object on which the robot has to perform the pick action are created in order to make the robot's workspace real. In addition, a collision check with the outside world is implemented to avoid problems. Next, the initial configuration is defined, which is calculated using the MATLAB function *homeConfiguration*(), and the final one is set as:

$$\text{goalConfig} = \begin{bmatrix} 0.8 & -0.1 & 0.2 & -1.5 & 0.2 & 1.2 \end{bmatrix} \tag{2}$$

It is now time to talk about the planner setup: using MATLAB's Robotics System Tool-

box object called *manipulatorRRT*() and its *plan*() function, the trajectory is planned, taking the robot model and environment into account. In addition, the maximum distance allowed to connect two nodes is set to 0.3 (the larger the values the faster it is, but also the less accurate) and the collision validation step is set to 0.1 (the smaller the more accurate, but the smaller the slower). Finally, once the output is obtained with the main nodes found by the RRT, the trajectory is interpolated exploiting the *interpolate*(*rrt*,*path*) MATLAB's function, which generates intermediate points along the path to obtain an even smoother trajectory. The results then are presented in Figure 11: in particular, Figure 11a represents the UR3 robot in its initial configuration, while in Figure 11b the pink line shows the trajectory performed to reach the final configuration, the desired one (equation (2)).



(a) Starting point. (b) Final point.

Figure 11: Path Planning.

The approach just described is then applied to a configuration that now includes two robotic arms. The main implemented changes to take into account are that, in addition to loading two UR3 manipulators, a second workpiece is added to the previously used environment. Next we introduce the second initial position, still set via *homeConfiguration*() and the second final position for the new robot:

$$\text{goalConfig2} = \begin{bmatrix} 0.8 & -0.1 & 0.2 & -1.5 & 0.2 & 1.2 \end{bmatrix} \tag{3}$$

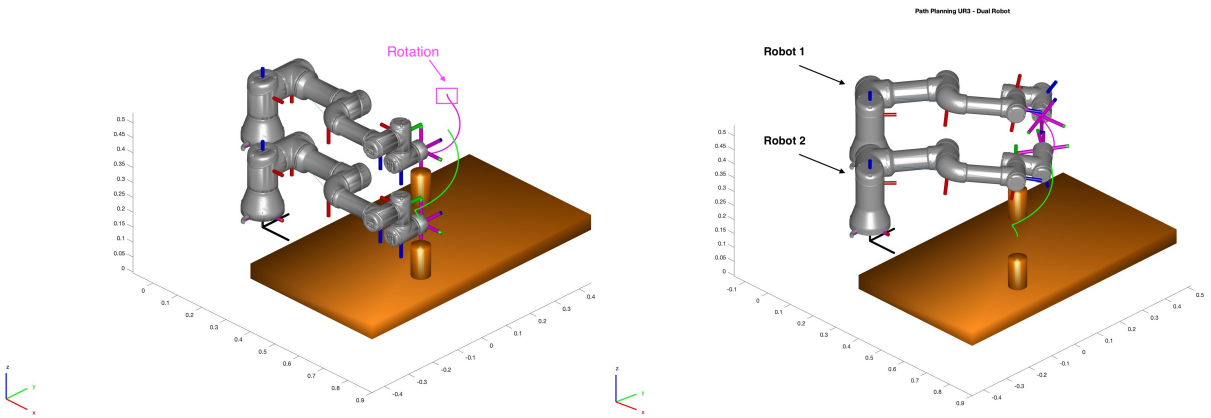
Note that in order for the end-effectors to keep the two workpieces close to each other and aligned, it is necessary to change the end position of the penultimate joint of the

first robot from goalConfig1 = [0.8 -0.1 0.2 -1.5 0.2 1.2] to:

$$\text{goalConfig1} = \begin{bmatrix} 0.8 & -0.1 & 0.2 & -1.5 & \mathbf{0.2 + pi} & 1.2 \end{bmatrix} \tag{4}$$

so that its end-effector is rotated by 180 degrees. As far as planner initialisation and trajectory planning are concerned, nothing changes, but the approach used considers double separate planning, thus two independent paths are constructed. As done previously, they are interpolated and a step synchronisation action is used to achieve coordination between the two robots. What is obtained by following this procedure can be observed in Figure 12 where again the trajectories followed by the end-effector of each robot are represented by pink and green lines. Notice that a rotation of the penultimate joint of the first rotobot is added in order to reach a final configuration that permits to have the end-effectors of the system facing each other.

Although good results are obtained in terms of both computational efficiency and performance, one cannot be satisfied with this method. In fact, one of the major limitations of the RRT algorithm is that it does not guarantee an optimal solution. In addition, it must be recalled that RRT only explores the space of configurations, also known as $C - space$, without taking into account any limits or constraints specific to the robot, which on the contrary are very important in this study.



(a) Starting point. (b) Final point.

Figure 12: Path Planning.

3.2.2 Orientation-Constrained Rapidly-exploring Random Tree (OC-RRT)

The next step involves the use of one of the multiple variants of RRT, namely the OC-RRT. While RRT limits itself to finding a kinematically valid path in environments with (or without) obstacles operating only in configuration space, on the other hand OC-RRT aims to generate compatible and optimised trajectories not only from a cost point of view, but also by integrating an optimal control problem that takes into account and guarantees collision-free, geometric constraints, joint constraints and additional constraints as desired in the planning phase. In this way, optimised trajectories are obtained, although on the other hand solving this problem is more computationally expensive. It can therefore be said that OC-RRT is a tool that combines random exploration with optimisation, thus offering more realistic solutions suitable for real physical systems. OC-RRT therefore appears to be very powerful in contests where precision and motion reactivity are essential, these properties are extremely important in the context of an automated robotic welding performed by two robots.

The implementation of this new algorithm consists of two main phases:

- Path Generation
- Optimal Control

The first phase can be related to RRT planning, in that an initial kinematically valid path is generated between the initial and final configuration: this takes into account obstacles in the environment but not the constraints of the system and operates uniquely in configuration space. Subsequently, the second phase aims to refine the trajectory: each intermediate configuration of the path is optimised locally, so as to satisfy the constraints. In particular, the optimisation problem is defined for each waypoint in the path and the objective function is based on the distance to the desired path.

The OC-RRT algorithm just presented has several advantages as it has modularity, since constraints can be easily extended, and good continuity, due to optimisation and interpolation that avoid abrupt jumps between two successive configurations. Moreover, although it is computationally more expensive, it is still a quick alternative to RRT. In conclusion, OC-RRT proves to be a robust and efficient solution for the case under consideration, which allows a better path quality, which is shorter and smoother. The block diagram of Figure 13 serves as a summary for this new path planning algorithm.

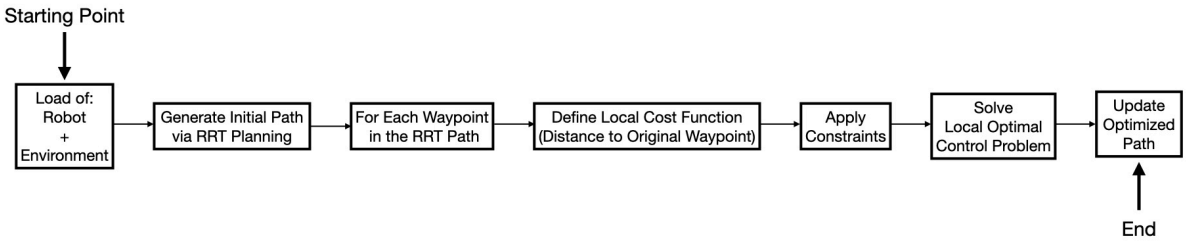
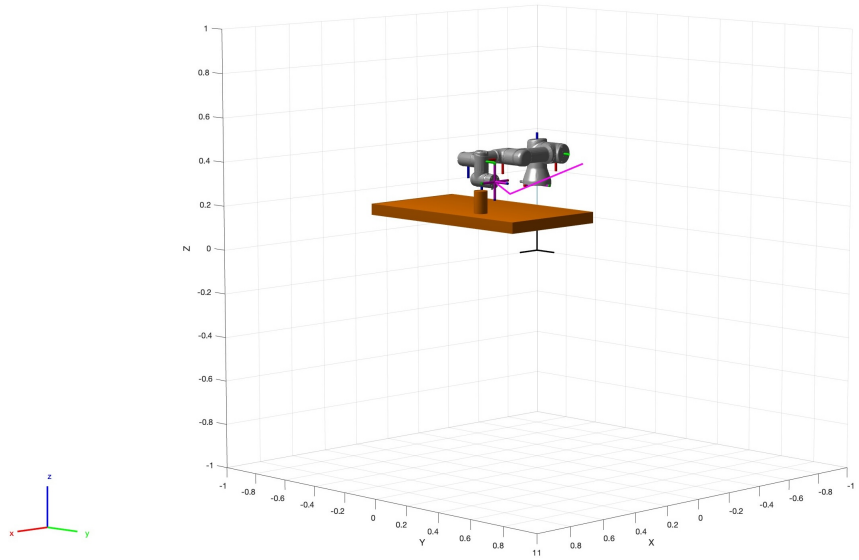


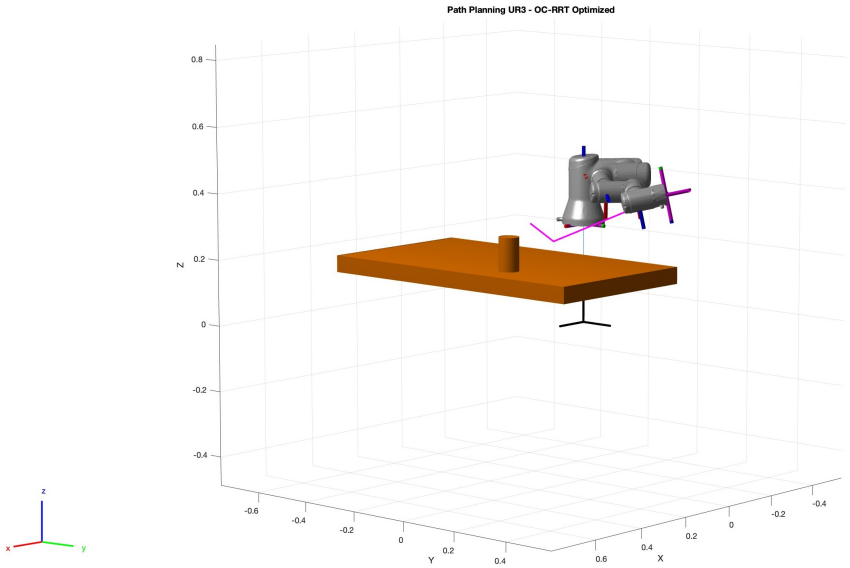
Figure 13: OC-RRT Block Diagram.

As before, the first test with this new algorithm is carried out using a single robotic arm. After loading the robot and the environment into the MATLAB script, defining the initial and final configuration and initialising the planner, two additional functions are defined. The first is aimed at optimising the path using optimal control, it is called *optimisePath_OCeasy*() and takes the path and the robot as input. It is responsible for iterating over each waypoint and it optimises motion between points. Here the cost function of the problem is presented and designed to penalize the deviation of the current state from the planned path, by minimizing the norm between the current state and the corresponding target point. Then the function *fmincon*() is used to optimise each intermediate configuration of the path. Note that *fmincon* is a MATLAB function for constrained optimisation and its purpose is precisely to minimise a constrained cost function. At this point the second function developed comes into play, which is called *robotDynamicsConstraint_OCeasy*(), and takes the current configuration and the robot as input. Here, the basic constraints of the system are defined, which for the moment are limited to only joint velocity limits. Each intermediate path configuration is then optimised locally using *fmincon*() in order to satisfy additional constraints.

The results obtained are those of Figure 14. Note how the environment has not been changed (and will not be changed for subsequent cases either) and how the trajectory of the end effector, identified by the pink line, although very similar to the case with RRT, leads to simulation results that enjoy greater speed in the trajectory movement. Moreover the trajectory result to be smoother and thus improvements with respect to the previous case are obtained.



(a) Starting point.

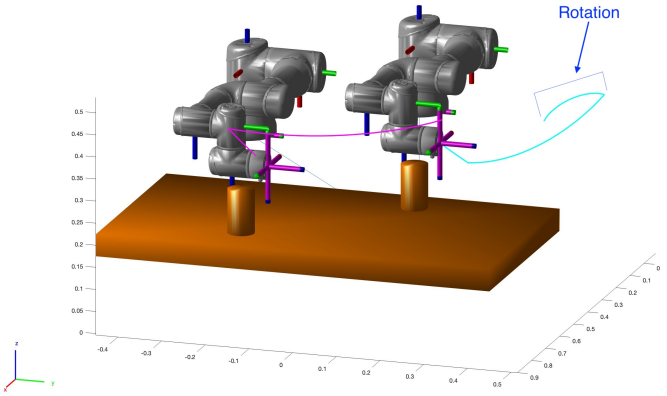


(b) Final point.

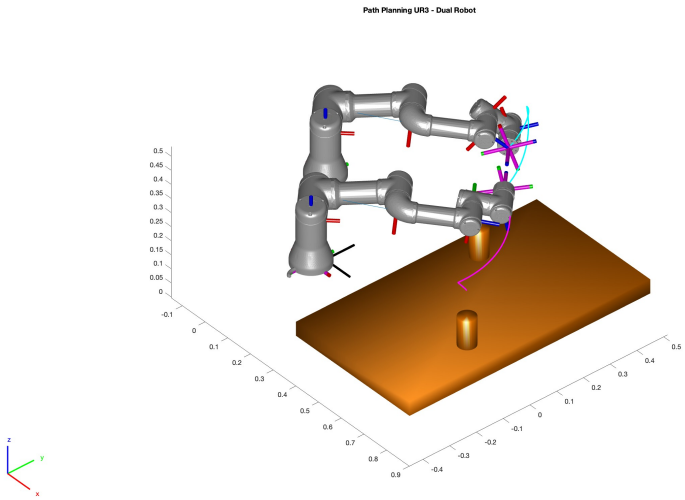
Figure 14: Path Planning.

Finally, the case of OC-RRT applied to two manipulator arms is considered. Once again, the robot and the environment are loaded, the initial and final configurations of both robots are defined and a first general path is found using the function *manipulatorRRT*(), one for each robot. Then the function *optimisePath_OCRRT2*() is used to optimise these paths, which is nothing more than a variant of the previous *optimisePath_OCeasy*() and which exploits *robotDynamicsConstraint_OCRRT2*(), itself a variant of *robotDynamicsConstraint_OCeasy*(). The procedure therefore remains the same as long as it is consistently adapted to the two robots. The results are those of Figure 15: it is again con-

cluded, looking also at the simulation, that the trajectories are executed more smoothly and without any jumps. The two trajectories are reported with a pink and with a light blue lines in Figure 15 and, as in the previous case where two robots are considered, the rotation of the penultimate joint has been added to the trajectory of the first arm to be able to mantain the two workpiece aligned, and this addition can be recognize at the end of the light blue trajectory.



(a) Starting point.



(b) Final point.

Figure 15: Path Planning.

The path planning phase is therefore successfully completed through the implementation and validation of the OC-RRT method. A feasible and optimised path for each manipulator is now available. However, the ability to follow these trajectories in a precise and synchronised manner depends entirely on the design of an appropriate control

strategy. For this reason, the next step of the project focuses on the development and validation of a control architecture capable of transforming planned trajectories into coordinated and dynamically feasible movements of the robotic arms.

3.3 Control Architecture

Once the trajectory planning phase has been completed, it is time to talk about the design and the subsequent validation of the control architecture. Control plays a key role: it is responsible for ensuring accurate execution of reference paths by the robotic manipulators. This requires not only accurate tracking, but also respect for joint limits, collision avoidance and smooth, coordinated movement.

The proposed control architecture, applied to one or two UR3 robots, is based on predictive strategies, aimed at anticipating the future behaviour of the system. Based on these predictions, the controller optimises the actions to be performed in order to reach the goal while respecting the imposed constraints. This predictive behaviour is particularly valuable in multi-robot systems, where interactions between them must be carefully kept under control to avoid conflicts and maintain task coherence, working in a coordinated manner towards the desired objective.

The proposed control strategy can be divided into two distinct phases:

- Model Predictive Control (MPC)
- Nonlinear Model Predictive Control

A first implementation based on MPC provides an initial way of managing the constraints and optimising the system's behaviour. From this formulation, a more advanced version is then developed, i.e. a predictive control scheme of the non-linear model, resulting in an NMPC, which exploits a more accurate representation of the system to further improve performance, especially in terms of tracking accuracy, responsiveness to disturbances and robustness of coordination.

3.3.1 Model Predictive Control (MPC)

An overview of model predictive control can be found in Subsubsection 2.3.2, where the main theoretical concepts of this approach are explained. However, it is useful to recall some concepts since this strategy provides the basis for the implementation of the control architecture ultimately used. MPC formulates the control task as a finite-horizon optimal control problem, solved at every step. At each iteration, a cost function

of the type:

$$J = \sum_{k=0}^{N-1} [(q_k - q_d)' \, Q \, (q_k - q_d) + u_k' \, R \, u_k] \tag{5}$$

is minimized under system dynamics and constraints, yelding then to an optimal sequence of control inputs. In particular q_k is the predicted configuration at future step k while q_d is the desired one, Q and R are positive definite weighting matrices and N is the predictive horizon. Moreover it is important to note that the constraints are usually enforced through linear inequality constraints. Finally only the first input is applied and the process is repeated at the next time step using updated system states.

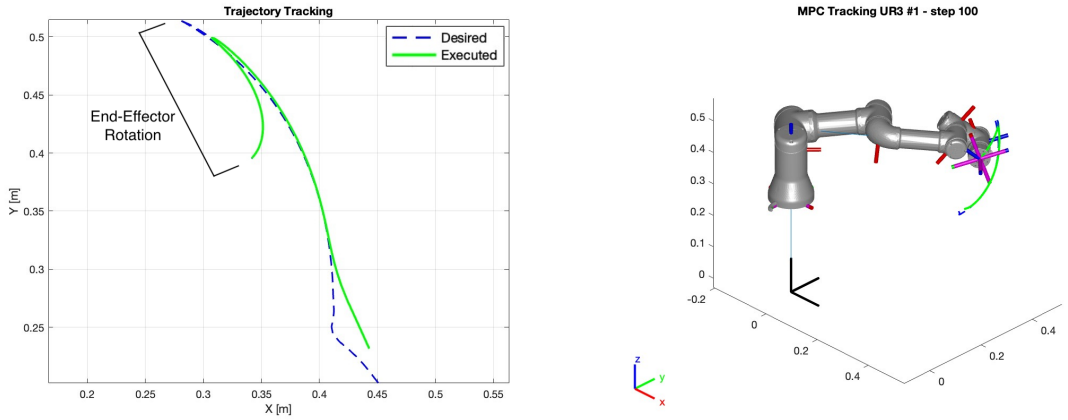
Although great results and computational efficiency can be achieved with the MPC, this modern control strategy requires the dynamic system to be approximated by a linear model. For this reason, for the UR3 robotic arm, the dynamic system is approximated by a discrete-time kinematic model. The control input is assumed to be the joint velocity vector $\dot{q}$ and torques and inertias are neglected. The evolution of the joint positions is then governed by:

$$q_{i+1} = q_i + \dot{q}_i \cdot dt \tag{6}$$

where q_i is the joint configuration at time step i, $\dot{q}_i$ is the joint velocity and dt is the time discretization step.

This technique is now presented applied to the case where a single UR3 arm is taken into account. First of all, the trajectory obtained by the OC-RRT algorithm (Subsubsection 3.2.2) is loaded into the MATLAB main script. However, since it is composed of discrete joint sequences and therefore of points that are not continuous in time, an interpolation is carried out by using the *spline*() function to transform this sequence of discrete points into a continuous trajectory. The next step involves the implementation of the MPC function, named *mpcUR3Step*(), which receives as input the current state, the desired function, the position along the path, and the time step. In its implementation the predictive horizon is set to 10, which means that it looks at 10 steps into the future; the weights matrices are set to $Q = eye(6)$ and $R = 0.01 \cdot eye(6)$; the cost function is created and the joint velocity $\dot{q}$ is optimised using *fmincon*() respecting the velocity limits, set to $-\pi \leq \dot{q}_k \leq \pi$; finally the next state is calculated. Concerning the cost function, called *totalCost*, it has the task of updating the state q, calculating the error with respect to the objective $qd\_Func(s)$ (i.e. the desired function), and accumulating the

cost by penalising the tracking error with Q and the velocity with R. The final goal is to track well and smoothly without too much acceleration or velocity. The results are given below in Figure 16 and it can be said looking at it that the goal has been achieved: the robot performs a trajectory (the green line in Figure 16) that is very close to the desired ones (the blue dotted lines in Figure 16).



(a) Trajectory Tracking. (b) 3D Trajectory Tracking.

Figure 16: Single Arm: Trajectory Tracking.

This can easily be extended to the case where the MPC algorithm is used to control two independent robotic arms: it is sufficient to also load the second trajectory, and apply the just presented *mpcUR3Step*() function twice to obtain the trajectory tracking of Figure 17, here the 3D plot is not reported since is not helpful in terms of comprehension. Good results are achieved confirming the fact that both of the robots follow the given trajectories: the light blue and the pink dotted lines are the reference trajectories while the blu and red lines depicted the executed trajectories as reported in the legend of Figure 17. In particular, it can be seen that the greatest error occurs at the beginning, but then the MPC is able to correct itself and follow the reference almost completely. Moreover, an initial error can be tolerated, since for the first part of the trajectory of R1, namely until the end-effector of robot 1 starts to rotate due to the rotation of joint 5, complete coordination between the two manipulator arms is not necessary.

Notice that the different phases of the trajectory for robot 1 are defined in Figure 17 and are assumed to be valid for the whole implementation.

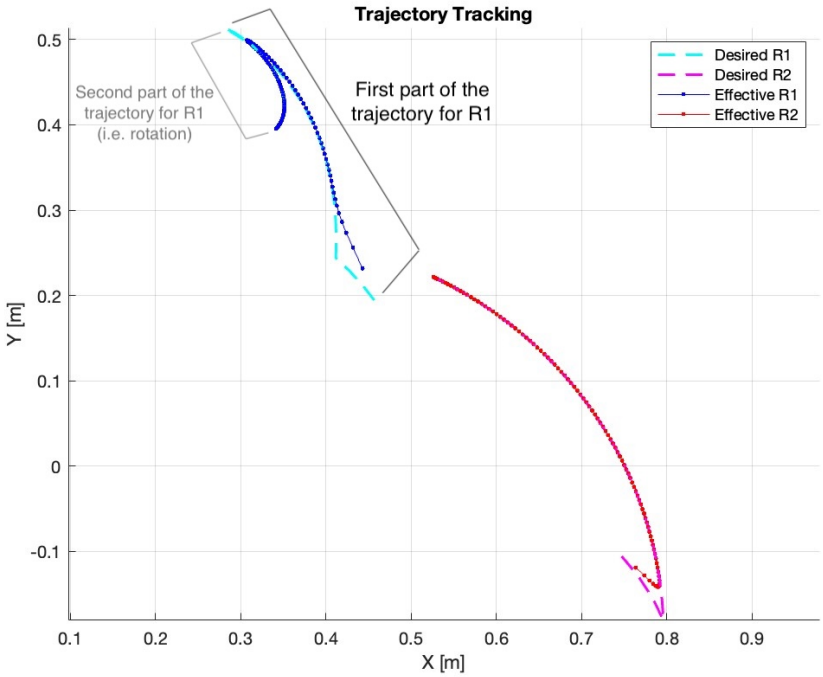


Figure 17: Two Independent Arms: Trajectory Tracking

It is now necessary to extend this approach to the case where an MPC with 12 degree-of-freedom is used. Compared to the previous case, small changes have to be made: first of all in the main script the *qd_Func*, namely the desired function, represents the concatenated vector of 12 joints (6+6). Moreover the vector of states passed as input to the MPC function is defined as a combined state vector $q\_combined$ of dimension 12x1. As far as the MPC function is concerned, taking into consideration the previous cases, it is sufficient to change the various degree-of-freedom defining the variables, changing 6 with 12. After making these adjustments, the results of Figure 18 are obtained: again a good trajectory tracking can be observed, with the greatest error at the beginning of the trajectories for both manipulators; however, the same considerations of before hold.

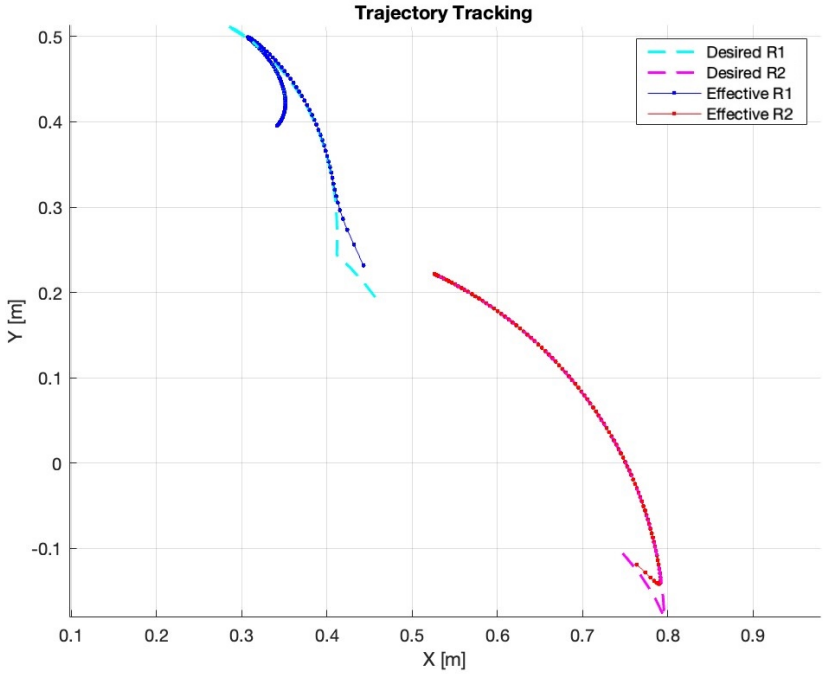


Figure 18: Two Arms and 12 DOFs: Trajectory Tracking

This is not the end of the MPC implementation with 12DOFs. It is true that the latest proposed results are satisfactory, but it should be mentioned that a high simulation time is required to achieve them. A development has therefore been thought out which, while maintaining good performance, is also able to do so in a reasonable time. In fact, this is when the MATLAB Yalmip toolbox is exploited.

YALMIP (i.e. Yet Another LMI Parser) is a modeling toolbox developed in MATLAB, specifically designed for the symbolic formulation of mathematical optimization problems. It acts as an intermediate abstraction layer between the mathematical definition of a problem and its numerical solution through external solvers. This enables the user to express optimization models in a readable and intuitive form, closely aligned with standard mathematical notation.
The modeling process in YALMIP is structured around three main stages:

- Symbolic definition: decision variables are defined using *sdpvar* objects, which are symbolic variables. Algebraic expressions such as objectives and constraints are then formulated using these variables.

- Problem canonicalization: once the model is defined, YALMIP automatically transforms it into a solver-compatible standard form. During this step, YALMIP per-

forms a structural analysis to determine properties such as convexity, linearity, or the presence of integer variables.

- Solver interfacing: YALMIP generates the numerical representation of the problem and interfaces with external solvers via appropriate APIs (Application Programming Interfaces). Once solved, the numerical results are mapped back to the symbolic variables using functions such as *value*(x).

This toolbox is capable of modeling a wide range of optimization problems, including linear or quadratic programming, and constrained linear optimization. It supports integration with both open-source and commercial otimization solvers, and in this work, the open-source solver *quadprog* is used.

A generic problem formulated by using YALMIP has the following structure:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \dots, m \\ & h_j(x) = 0, \quad j = 1, \dots, p \\ & x \in \mathcal{X} \subseteq \mathbb{R}^n \end{aligned} \tag{7}$$

where x are the decision variables, namely the symbolic ones defined by *sdpvar*, $f(x)$ is the cost function and $g_i(x)$, $h_j(x)$ are the inequality and equality constraints respectively. In the context of MPC, YALMIP proves to be a particularly convenient and flexible tool. Thanks to its symbolic structure, it allows for an intuitive formulation of the optimization problem, and constraints can be easily added, which is especially advantageous when dealing with complex robotic systems.

In order to take advantage of this toolbox, some modifications to the MPC function must be made. Code 2 shows the definition of the sdpvars, of the constraints and of the settings of the new solver. Ultimately, the results of Figure 19 are obtained, which conclude the study of control through MPC. Not only the time required for the simulation is greatly reduced, but improvements are also noted in terms of performance: the initial error is reduced and tracking is even more accurate along the entire trajectory.

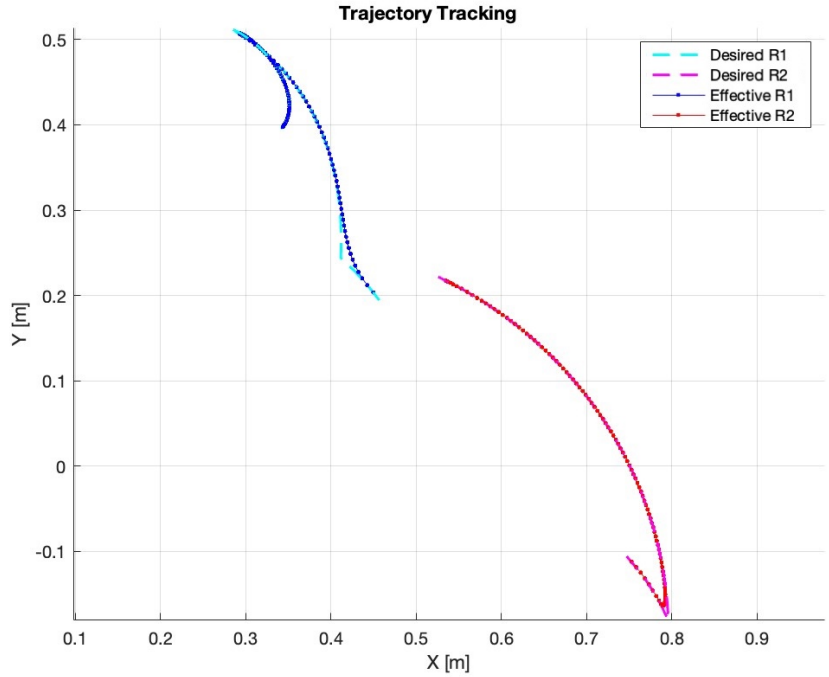


Figure 19: Two Arms and 12DOFs YALMIP: Trajectory Tracking.

Code 2: MPC function with 12 DOFs Exploiting Yalmip

```
function [qNext,qDotOpt]=mpcUR3Step12DOF_YALMIP(q,qdFunc,s_now,dt)

% Define: Predictive Horizon N=10, DOF=12 and q of dimension 12x1
% Define the initial State qdot0: qdot0=zeros(DOF, 1)
% Define weight matrices: Q=eye(DOF) and R=0.01 * eye(DOF)

% Optimization variable
u = sdpvar(DOF, N);

% Objective Function:
J = J + (q_next-qd)' * Q * (q_next-qd) + u(:,i)' * R * u(:,i);

% Constraint:
constraints = [u >= -pi*ones(DOF, N), u <= pi*ones(DOF, N)];

% Resolution of the optimization Problem
optimize(constraints, J, options);

```

```
% Optimal Solution:
qDotOpt = value(u);

% Update qNext as:
qNext=q+dt*qDotOpt(:,1)
```

In summary, the approach that exploits the MPC algorithm to control the system, consisting first of a single UR3 robot, and then of two manipulator arms, with the aim of performing coordinated movement and following reference trajectories as closely as possible, has several advantages. The most important are the smoothness of the trajectory and the constraint handling, to which are added, thanks to the last case, fast simulation times and excellent trajectory tracking. On the other hand, however, one must remember that an approximation of the model is being used, in particular a discrete linear dynamic, which is essential for applying MPC but which does not allow sufficiently realistic control for the case under consideration. For this reason, the next step will focus on a variant of predictive control, namely Nonlinear MPC.

3.3.2 Nonlinear Model Predictive Control (NMPC)

To overcome the limitations of linear MPC, a Nonlinear MPC approach is adopted. NMPC is an extension of the classical MPC just seen, in which instead of using a linear model of the system, the full nonlinear dynamic model is used, allowing more accurate predictions and better performance, especially in the context of coordinated dual-arm manipulation. The NMPC solves a constrained optimisation problem at each time instant over a finite horizon, but using the true nonlinear equations of the system. Thus, the nonlinear dynamics of each UR3 manipulator are modelled using a function of the form:

$$\dot{x} = f(x, u)$$

where x represents the state vector and u the control inputs. The NMPC formulation keep the same basic structure as MPC but replaces the linear system model with a nonlinear one, then the cost function is similarly defined:

$$\int_t^{t+T} (q(\tau) - q_d(\tau))' \, Q \, (q(\tau) - q_d(\tau)) + u(\tau)' \, R \, u(\tau) \quad d\tau \tag{8}$$

which, in practical applications, is discretized to allow implementation using standard numerical optimization software. This NMPC scheme will be the core of the proposed control architecture, providing a flexible and powerful tool for managing complex dual-arm coordination tasks in real-time simulation environments.

Before proceeding with the actual implementation of the control architecture just presented, the focus is put on the choice of modelling and optimisation environment, which is crucial in order to achieve high computational performance and good management of dynamic models. In the case of MPC, the YALMIP modelling toolbox of MATLAB was used. However, for NMPC applications in the robotic field, where problems have to be solved in real time, CasADi offers superior performance, greater flexibility in modelling dynamics and precise and efficient automatic derivation. CasADi is precisely an open-source framework designed for the symbolic formulation of nonlinear optimisation problems, with a particular focus on dynamic systems and optimal control. It is accessible via an interface in MATLAB and offers powerful tools for the automatic generation of derivatives and the efficient construction of models to be solved by external solvers, thus being well-suited for applications in nonlinear optimal control where repeated evaluation of gradients and Jacobians is required for solver efficiency. Once again, symbolic objects are included, with CasADi implementing two types of them:

- SX (Symbolic Expression): suitable for small models or those with a completely symbolic structure.
- MX (Matrix Expression): more flexible and scalable, they allow the construction of complex expressions and the efficient manipulation of dynamic systems and sparse matrices.

Symbolic modelling is therefore one of the main features of the framework, making it possible to express complex mathematical expressions in a compact and computationally efficient manner and thus generate symbolic functions. Furthermore, CasADi benefits from a robust and extremely efficient system for automatic differentiation, enabling the fast and accurate calculation of gradients, Jacobians and Hessians, which are fundamental elements in nonlinear optimisation problems. Finally, it offers a direct interface with numerous solvers for nonlinear problems, including *ipopt*, *worhp*, *snopt* and many others, making it possible to solve nonlinear optimisation programmes (NLP), differential-algebraic equations (DAE) and optimal control problems (OCP). A typical problem solvable in CasADi takes the same form as equation (7) defined previously,

which involves decision variables, objective function, and equality and inequality constraints.

The combination of flexible symbolic modelling, numerical efficiency and compatibility with advanced solvers makes CasADi an ideal tool for control and optimisation scenarios in advanced robotics, and is also particularly suitable for real-time implementations, where the rapidity and reliability of calculating derivatives is an essential requirement. For these reasons, CasADi is therefore adopted as the main tool for the developement of the control startegy based on Nonlinear Model Predictive Control in the case of study presented in this thesis.

In this case as well, the step-by-step implementation strategy is followed. Therefore, the first scenario considered involves a system composed of a single UR3 robotic arm. The approach is based on a full dynamic model of the robot, generated from the data contained in the *.urdf* file, and on an online finite-horizon optimization, solved using the CasADi toolbox and the *'ipopt'* solver. Obviously, the first step is to load the UR3 manipulator model and the reference trajectory computed via OC-RRT. It is important to recall that this trajectory consists of a sequence of joint configurations, therefore, interpolation using the *spline*() function is required to obtain a continuous trajectory. The next step is the creation of the dynamic model of an n-joint robot, which in this case has $n = 6$.

The robot dynamics can be expressed by the following equation:

$$M(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau \tag{9}$$

where:

- q is the vector of joint angular positions.
- $\dot{q}$ is the vector of angular velocities.
- $\ddot{q}$ is the vector of accellerations.
- $M(q)$ is the inertia matrix.
- $C(q, \dot{q})$ are the centrifugal and Coriolis terms.
- G(q) is the vector of the gravitational forces.

- τ are the torques applied to the joints.

CasADi then implements these terms in symbolic form so that they can then be easily used in the formulation of the optimisation problem. The dynamics is then obtained via Lagrange's method, namely starting from the Lagrangian:

$$L(q, \dot{q}) = T - V \tag{10}$$

where T is the kinetic energy calculated considering the translational and rotational contributions of each link, and V is the gravitational potential energy. In order to compute the kinetic and potential energy of each link the Jacobians must be used, specifically:

$$v_i = J_{v_i}\, \dot{q} \quad \text{Center of mass velocity} \tag{11}$$

$$\omega_i = J_{w_i}\, \dot{q} \quad \text{Angular velocity} \tag{12}$$

Thus, the kinetic energy for each link is:

$$T_i = \frac{1}{2} m_i v_i^T + \frac{1}{2} \omega_i^T I_i \omega_i \tag{13}$$

while, the gravitational potential energy is:

$$V_i = m_i g^T p_{com_i} \tag{14}$$

where p_{com_i} identifies the position of the centre of mass.
Adding together now the kinetic and potential energies of each link, one can construct the Lagrangian of equation (10) and by applying the Lagrange's formula:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = \tau \tag{15}$$

a system of non-linear equations is obtained which can be rewritten in the form of equation (9).

As far as implementation in MATLAB is concerned, the main points of the dynamic computation are given: using CasADi, T and V are symbolically computed, then L is obtained and consequently Lagrange's formula is applied. After that, CasADi automatically constructs and outputs three symbolic functions $M(q)$, $C(q, \dot{q})$ and $G(q)$ representing inertia matrix, centrifugal and coriolis terms, and gravity, respectively. These symbolic CasADi functions are then passed to the NMPC to simulate the robot's dynamics over time, thereby optimising the torques to be applied to follow the desired trajectory.

Code 3: Dynamic Model - CasADi

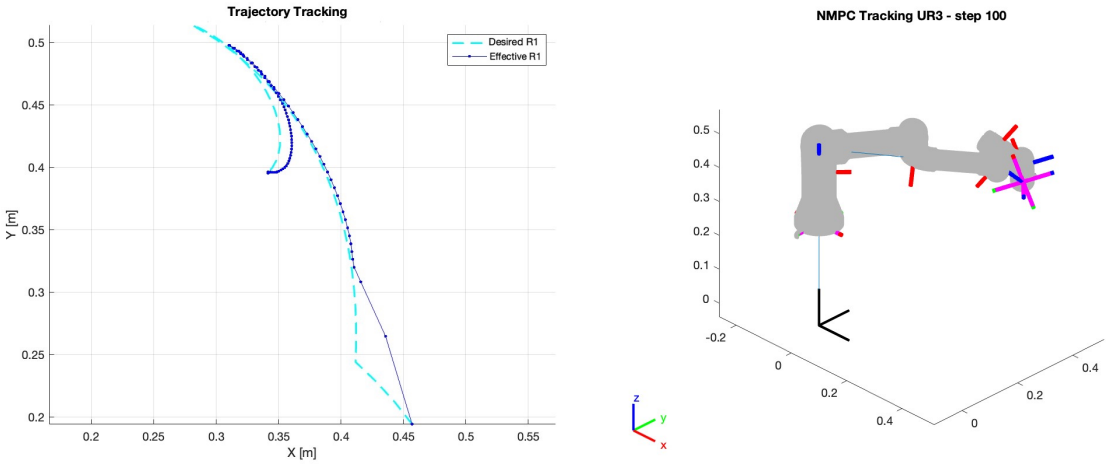
```
%% Derivation of dynamics
    L = T_kin - V;
    M = jacobian(jacobian(L, qd).', qd);
    C = jacobian(jacobian(L, qd).', q) * qd - jacobian(L, q).';
    G = jacobian(V, q).';

%% CasADi Functions
    M_fcn = Function('M_fcn', {q}, {M});
    C_fcn = Function('C_fcn', {q, qd}, {C});
    G_fcn = Function('G_fcn', {q}, {G});
```

The next step concerns the implementation of the NMPC controller. A function named *nmpcUR3Step_torque*$(q, qdot, qdFunc, s\_now, dt, M\_fcn, C\_fcn, G\_fcn)$ is developed, whose inputs are: the vector of the robot's joint positions, the vector of joint velocities, the function that provides the desired trajectory, the current time step, the integration step size, and the functions obtained via CasADi representing respectively the inertia matrix, the centrifugal and Coriolis terms, and the gravity vector. The goal is to solve the following optimization problem over a prediction horizon set to N=10:

$$\min \sum_{i=0}^{N-1} (q_i - q_{des})' \, Q \, (q_i - q_{des}) + \dot{q}_i' \, Q_{dot} \, \dot{q}_i + \tau_i' \, R \, \tau_i; \tag{16}$$

where the weights matrices Q, Q_{dot}, and R respectively penalize: tracking error, joint velocity, and control effort. The optimization problem is solved at each time step, and in case the solver *ipopt* fails (e.g. due to convergence issues or NaN values in the dynamics), the code handles the exception using a *try & catch* block (Code 4). In such cases, the current state is returned as a fallback to prevent the simulation from crash-

ing. Naturally, the goal in the following study is to obtain results without relying on this fallback mechanism. The results obtained are shown in Figure 20, where, in addition to the trajectory tracking plot, the configuration of the robot at the end of the simulation is also reported, confirming that the trajectory was correctly followed and the penultimate joint has correctly performed the rotation (a detailed illustration of the rotational part of the trajectory can be found in Figure 16a). In terms of accuracy, the largest error is observed at the beginning of the trajectory, indicating the controller's ability to effectively adapt to different segments of the motion. As an additional note, Figure 20b represents the robot in a more stylised manner in order to make the simulation faster.



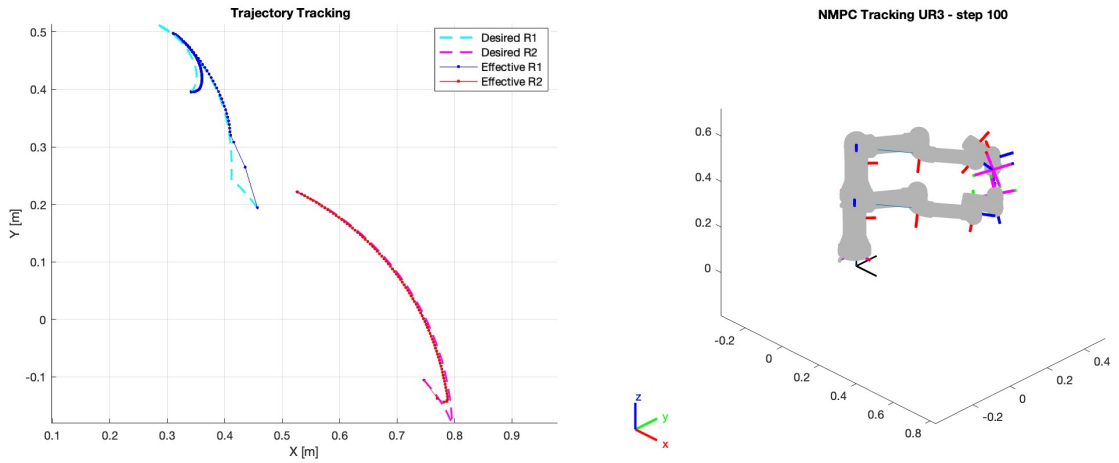
(a) Trajectory Tracking. (b) Final Configuration.

Figure 20: Single Arm NMPC.

Code 4: Try & Catch

```
1       try
2           sol = opti.solve();
3           qNext = full(sol.value(q_var(:,2)));
4           qdotNext = full(sol.value(qd_var(:,2)));
5           tauOpt = full(sol.value(tau_var(:,1)));
6       catch
7           fprintf("NMPC error at step s=%.4f\n", s_now);
8           qNext = q;
9           qdotNext = qdot;
10          tauOpt = zeros(nq,1);
11      end
12  end
```

At this point, it is very simple to extend this strategy to the case where the system consists of two independent UR3 robots. The function responsible for developing the NMPC is not changed, and neither is the function that defines the dynamics. All the steps remain the same. It is simply a matter of applying the dynamics and NMPC functions for each arm. The results obtained are shown in Figure 21: trajectory tracking is sufficiently accurate and the final robot configuration result to be correct.



(a) Trajectory Tracking. (b) Final Configuration.

Figure 21: Dual independent Arm NMPC.

The final implementation of the control architecture, that is the one enabling the development of a 12 degree-of-freedom NMPC, is now discussed. As usual, the models of the two manipulator arms and their respective reference trajectories are first loaded. The trajectories are then interpolated using the *spline*() function, generating two separate qd_Func, which are subsequently combined into a single function $qdFuncTotal$, defining the desired function to be given as input to the NMPC controller. The symbolic dynamics for each UR3 are then generated, as in the case of a single manipulator, obtaining the symbolic expressions of the inertia matrices $M(q)$, the centrifugal and Coriolis terms $C(q, \dot{q})$ and the gravity vector $G(q)$ (see Code 3).

However, these functions must be combined in order to have a total and complete dynamics of the overall system. For this reason, an additional function, which is called *getUR3DoubleDynamics*(*q, qdot, M_fcn, C_fcn, G_fcn*) is implemented, in which the symbolic dynamics of the 12 degree-of-freedom system is defined. This function takes as input the full state vectors $q \in \mathbb{R}^{12}$ and $\dot{q} \in \mathbb{R}^{12}$, as well as the symbolic expressions for the inertia matrix M, centrifugal and Coriolis terms C, and the gravity vector G. The

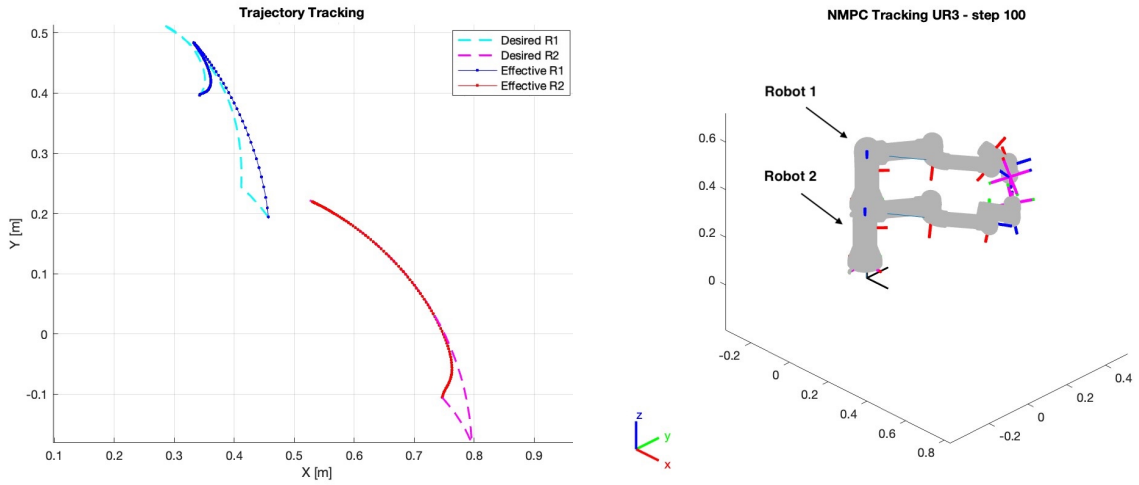
state is then split into two separate 6 degree-of-freedom subsystems, representing the two arms of the robot. For each subsystem, the corresponding dynamic quantities are computed independently using the provided symbolic functions. The final output then consists of the combined dynamics of the full 12 degree-of-freedom system: the inertia matrix M_{total}, the centrifugal and Coriolis terms C_{total} and the gravity vector G_{total}.

Some modifications are also necessary with regard to the function implementing the NMPC: the symbolic optimisation problem in CasADi now has 12 state variables q and 12 velocity variables $\dot{q}$ over a prediction horizon $N = 10$, and the objective function, which remain with the same structure as before, simultaneously minimises the two trajectories now. This extension makes it possible to simulate a cooperative control in which two robots manipulate workpieces. In addition, realistic constraints are introduced on the state, velocity and torque variables that are simultaneously valid for both robots and visible in Code 5. Moreover, as before there is a *try&catch* block (Code 4) to handle possible failures of the solver, which is not the case for the simulations tested.

Code 5: Limit Constraint in the NMPC function

```
% Limit Constraint
    q_min = -pi * ones(nq,1);
    q_max =  pi * ones(nq,1);
    qd_max = 2*pi; %pi
    tau_max = 80;

    for i = 1:N+1
        opti.subject_to(q_min <= q_var(:,i) <= q_max);
        opti.subject_to(-qd_max <= qd_var(:,i) <= qd_max);
    end
    for i = 1:N
        opti.subject_to(-tau_max <= tau_var(:,i) <= tau_max);
    end
```

The results obtained are those of Figure 22: in addition to the final configuration of the two UR3s representing the two end-effectors facing each other, meaning that the end-effector of robot 1 has performed its rotation, there is the trajectory tracking of each robot. This allows the desired trajectory to be compared with the one actually executed, highlighting the effectiveness of the control in maintaining tracking on both arms. In fact, as in the previous cases, the biggest error is found at the start of the trajectory but then the NMPC is able to correct it.



(a) Trajectory Tracking. (b) Final Configuration.

Figure 22: Double arm with a 12DOFs NMPC.

In conclusion, the development and validation of the NMPC framework in both reduced and full form demonstrated the feasibility and effectiveness of the proposed approach. The reduced 6 degree-of-freddom version provided a valid starting point for testing the basic control logic. However, the full 12 degree-of-freedom formulation allows the entire joint dynamics of the robot to be modelled, providing a more accurate representation of the system. For this reason, the 12 degree-of-freedom NMPC controller is chosen as the final control strategy used in the following work.

In the next chapter, the robustness analysis of the system will be further investigated. External noise and disturbances will be introduced to evaluate its performance under more realistic operating conditions. Furthermore, another constraint will be added, in addition to the limits already introduced for joint positions, velocity and torque, in order to ensure the feasibility and safety of the planned trajectories, thus bringing the system closer to a real application.

4 Design Enhancement: Noise and Constraints

After defining and implementing a control strategy based on a 12 degree-of-freedom NMPC, the next step involves refining the system design. The aim of this section is to introduce elements that bring the model closer to real-world operating conditions, moving beyond the ideal assumptions considered so far. In particular, external disturbances and noise are introduced to assess the robustness and reliability of the controller under uncertainty and environmental perturbations. Additionally, an extra constraint is added to the system: a maximum allowable distance between the two end-effectors of the two manipulators. This steps represent a fundamental extension of the initial design and open the way to a more comprehensive evaluation of the proposed system in more realistic and complex scenarios.

The system considered from now on is the one consisting of two UR3 manipulator arms, which are required to follow a reference trajectory generated through the Orientation-Constrained RRT path planning algorithm. These trajectories will then be the input to the controller, developed using a 12 degree-of-freedom Nonlinear Model Predictive Control.

4.1 Noise to the sensor

The purpose of this phase is to test the robustness of the system in the face of added noise on the sensors. This is a fundamental step that allows the model developed under realistic conditions to be evaluated while taking into account the unavoidable imperfections in signal measurement. What is therefore done is to introduce noise to the sensor readings which provide as input to the controller the joint positions q and the joint velocities $\dot{q}$. Gaussian noise is chosen for both inputs: in particular, the noise is modelled as a normal disturbance with zero mean and scaled variance to be applied to the current values of q and $\dot{q}$:

$$q_{noisy} = q_{current} + param_q \cdot \mathcal{N}(0, 1) \tag{17}$$

$$\dot{q}_{noisy} = \dot{q}_{current} + param_{\dot{q}} \cdot \mathcal{N}(0, 1) \tag{18}$$

where $param_{q,\dot{q}}$ are nothing other than the standard deviation of the Gaussian noise defining the amplitude of the noise on the measurement. The reason for choosing a

noise of this type, i.e. Gaussian noise, is that it represents many types of measurement errors due to electronic sensors (such as encoders and gyroscopes) in a statistically realistic manner. In MATLAB, the noises are generated via the *randn*() function and it is assumed that the position measurements have $param_q$ of equation (17) set to 0.01 rad and those for velocities have $param_{\dot{q}}$ of equation (18) set to 0.005 rad/s. These values are then given as input to the *nmpcUR3DoubleStep_torque*() function presented in the previous chapter, specifically in Subsubsection 3.3.2.

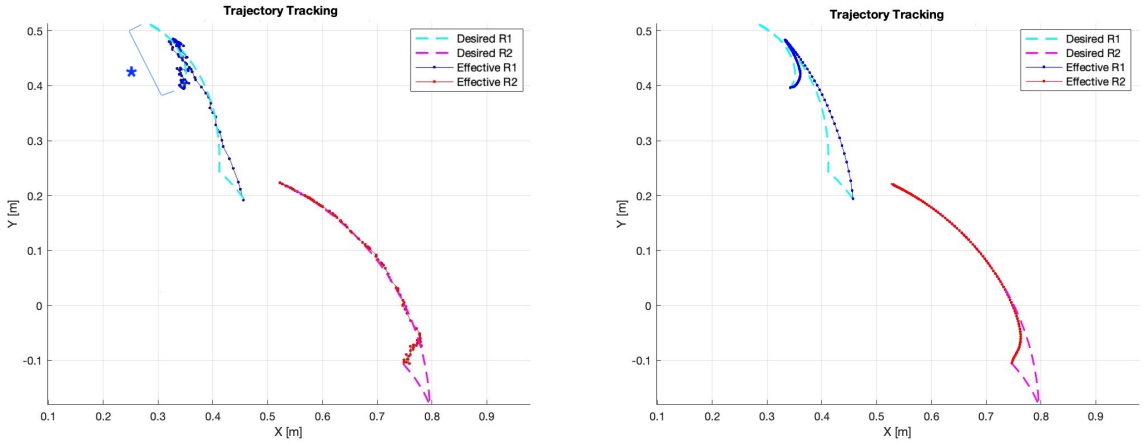
For a moment, let 's focus on the choice of values assigned to the noise standard deviations, i.e. $param_q$ and $param_{\dot{q}}$. For a complete reasoning, it is necessary to refer to the official datasheet of UR3 ([33]). Here it can be seen that the value of Repeatability is ± 0.1 mm, which identifies the accuracy in returning to the same point, but referring only to the arm end point and not directly to the joints. The value $param_q = 0.01$ rad corresponds to an angular movement of approximately 0.5 degrees, which if applied to a joint is equivalent to a displacement of a few millimetres at the arm end. Therefore, it can be said that the choice of a standard deviation of 0.01 of the noise applied to the position measurement is a realistic value for simulating a slight noise in the measurement of the position of a joint without being either too low (and therefore useless) or too high (and therefore unrealistic). Turning now to the analysis of $param_{\dot{q}}$ in the same datasheet [33] the following velocity data are found: *"All wrist joints:* 360 *degrees/s. Other joints:* 180 *degrees/s"* which therefore translates into approximately 6 and 3 rad/s respectively. If one considers the value 0.005 rad/s, this is only about 0.08% and 0.16% respectively of the maximum. So this is a small noise, but it is able to reproduce signal fluctuations simulating normal operational inaccuracy.

The Table 1 shows a summary of a range of values for a realistic test scenario which do not lead to degeneration in the simulation: they do not falsify the dynamics, they induce slight inaccuracy and they allow the robustness of the controller to be assessed in the presence of non-ideal measurements. Note how the values selected are perfectly within the plausible range.

Table 1: Plausible ranges for Gaussian noise in UR3 joint sensors

Input Type	**Standard Deviation**	**Suggested Range**	**Justification**
Position (q)	$param_q$	0.001 – 0.02 rad	It simulates realistic encoder errors. UR3 have high-resolution encoders, but noise due to vibration, backlash, or quantisation is still present. Values above 0.02 become unrealistic.
Velocity ($\dot{q}$)	$param_{\dot{q}}$	0.001 – 0.01 rad/s	Models errors in velocity estimation, which is often derived and subject to noise or latency. Higher errors may compromise the responsiveness of the control, but values within this range are considered safe.

It is now time to present the results of the system's noise response. Not only the model is tested with the noise added to the position measurement or to the velocity measurement (Figure 23a and Figure 23b respectively), but also the system's response when both noises are present at the same time (Figure 24) is analysed. It can be seen that the noise introduced at $\dot{q}$ does not perturb the system very much, while the noise introduced at position has a more pronounced effect, especially during the rotation phase of the end effector of robot 1 (underlined by the $*$ symbol in Figure 23a). In the case where the robots are affected by both noises simultaneously, similar results to those already described for Figure 23 can be seen. However, the manipulators are able to arrive at the desired final configuration without any problems, thus achieving a correct configuration for the final welding operation. It can therefore be concluded that the system performs robustly with respect to this type of noise.



(a) Noise in q. (b) Noise in $\dot{q}$.

Figure 23: Noise in just one input.

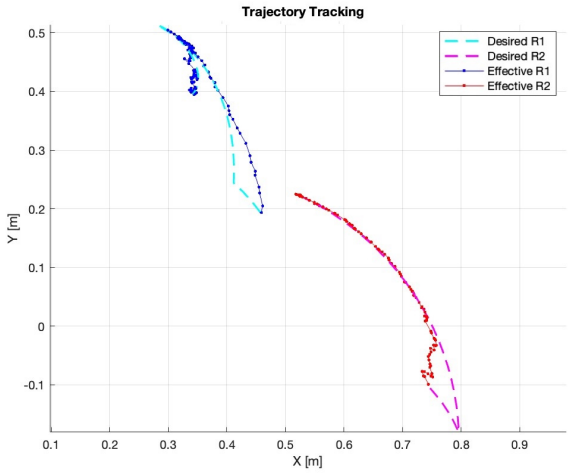


Figure 24: Noise in both inputs

4.2 Dynamic disturbances

A different type of disturbance is now introduced, namely a dynamic disturbance into the simulation, in order to act directly on the computation of the system dynamics. This further tests the robustness of the NMPC control. In particular, this disturbance simulates unmodelled torques (or forces) that may arise during real execution due to phenomena such as collisions, variable friction, interaction with the environment or inaccuracies in the model. Disturbances are therefore modelled as additional moments (or forces) introduced into the dynamic model during the calculation of joint accelerations, thus affecting the temporal evolution of the robotic system. The model can therefore be rewritten as:

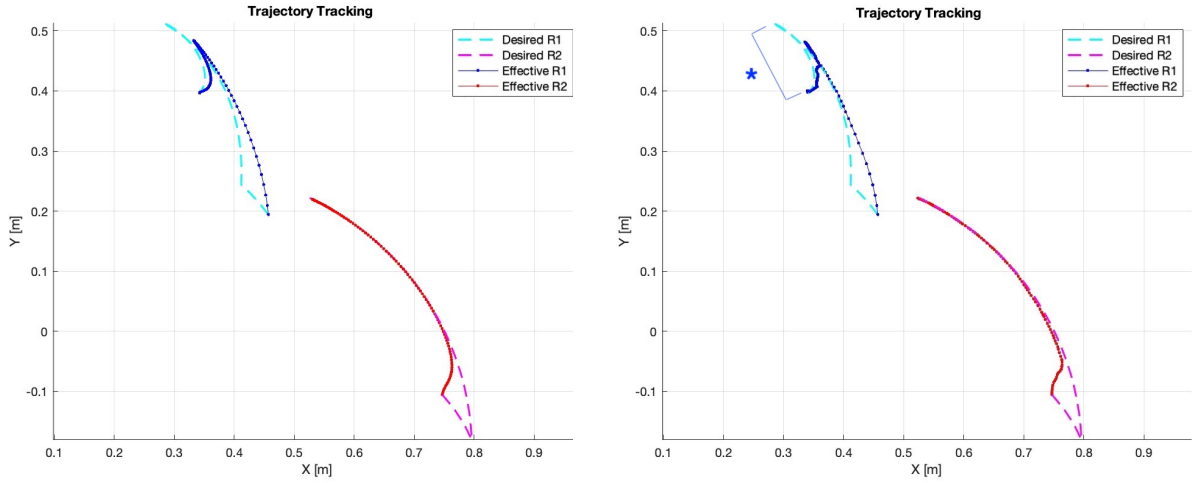
$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau + \tau_{dist} \tag{19}$$

where τ_{dist} represents precisely the dynamic disturbance introduced, which is defined as a Gaussian disturbance ($\tau_{dist} \sim \mathcal{N}(0,1)$). Again, the choice to use a Gaussian noise is because it represents a good realistic statistical model for the disturbance.

However, some implementation changes are necessary with regard to the NMPC: within the predictive optimisation cycle, the disturbance is added to the system dynamics. At each prediction step, a Gaussian disturbance with zero mean and standard deviation of 0.5 is considered. The choice of this value is justified by the fact that it represents a moderate but significant force with respect to the system dynamics, sufficient to cause a visible deviation in the trajectory without, however, compromising the success of the simulation. Furthermore, it allows a concrete test of the controller's ability to react to non-ideal conditions. More specifically, it can be stated that: the UR3 robot presents a maximum instantaneous torque set at 80 Nm (see Code 5) Therefore, it can be stated that the choice of a standard deviation of $\delta = 0.5$ Nm: represents a disturbance of less than 1% of the maximum torque for the joints, thus remaining within the range of real disturbances that can be caused by load variations, variable friction or noise in the identified models. The disturbance is then included in the dynamic equation. The cost function remains unchanged from the previous cases and continues to penalise deviation from the desired trajectory, velocities and control efforts. The initial conditions and constraints remain consistent with previous versions and the solver used is still *'ipopt'*.

The addition of such a disturbance allows for a more realistic evaluation of the system's behaviour when the robot's dynamics are not perfectly known, and represents a further

step forward in validating the robustness of the dual-robot NMPC approach. The result is therefore presented below. While in Figure 25a the simulation is performed without any dynamic disturbance, on the other hand Figure 25b depicts the trajectory tracking obtained by adding the Gaussian disturbance to the dynamic model.



(a) No Dynamic Disturbances. (b) Final Configuration.

Figure 25: Presence of Dynamic Disturbances.

Therefore it can be concluded by analysing the results that the system also responds very well to this type of disturbance and is able to perform good trajectory tracking, even if, as in the previous case, the part of the simulation most subject to error is that in which robot 1 rotates the penultimate joint to align itself with the end-effector of robot 2 (again identified by $*$). Again both robots reach the correct final point in the trajectory tracking: this means that the two end-effectors result to be aligned with each other, which is crucial for the welding task. In conclusion, therefore, the implemented system is robust to dynamic disturbances.

4.3 Additional Constraint in the NMPC

Having reached this point, the goal is to integrate a further constraint into the NMPC model: it is desired to limit the maximum distance between the end-effectors of the two UR3 manipulator arms during the execution of the trajectory. This requirement arises from the fact that, especially from the second part of the trajectory onwards (i.e. from when the end-effector of robot 1 rotates the penultimate joint to align its end-effector with the one of robot 2), it is essential to preserve a solid spatial configuration between the two arms so that the welding operation can take place in a coordinated, efficient

and safe manner.
The constraint is to be expressed in terms of the squared Euclidean distance between the Cartesian positions of the end-effectors, obtained by direct kinematics from the joint configurations, and the maximum acceptable distance is set at 10 cm. The constraint therefore results to be:

$$||p_1(q_{tot,1}) - p_2(q_{tot,2})||_2^2 \leq d_{max} \tag{20}$$

where d_{max} is precisely the maximum allowable distance between the two end-effectors, $q_{tot,1}$ and $q_{tot,2}$ are the complete joint configurations of robot 1 and 2 respectively, and finally $p_1(q_{tot,1})$ and $p_2(q_{tot,2})$ are the results of direct kinematics, namely they represent the position in space of end-effectors 1 and 2 respectively.

The implementation of this constraint is presented in two distinct ways, first by means of a soft constraint, thus incorporated in the optimisation cost, and then by a hard constraint, directly imposed as a rigid constraint on the problem.

4.3.1 Soft Constraint

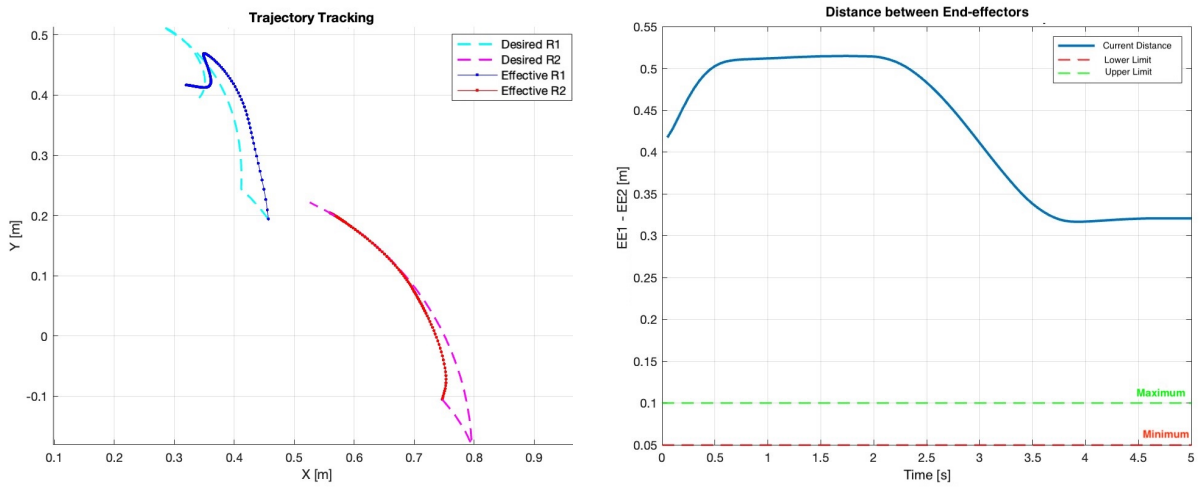
The first strategy presented is the one that applies the constraint via a soft constraint: in this case it is added to the cost function and penalised when violated. Specifically, at each prediction step *i* the quadratic distance between the end-effectors is calculated, and in the case of exceeding the predetermined limit d_{max}, a quadratic penalty proportional to the excess is applied. Mathematically speaking:

$$\begin{aligned} excess_i &= max(||p_{1,i} - p_{2,i}||_2^2 - d_{max}, 0) \\ J &= J_{tracking} + \lambda_{dist} \sum_{i=1}^{N} excess_i^2 \end{aligned} \tag{21}$$

where $p_{1,i}$ and $p_{2,i}$ are the end-effectors cartesian position at step *i*, and λ_{dist} is the parameter regulating the influence of the constraint on the total cost. The following formulation is integrated into the NMPC code where λ_{dist} is set to 1000, which means that an almost rigid constraint is needed but it should not make the solver fails. Through this approach, the system is allowed to have small temporary violations of the constraint, if they are functional to achieving the desired tracking.

However, if one looks at the results in Figure 26, they are not satisfactory: not only the

tracking gets worse, especially for robot 1 in the rotation phase, but the constraint is never satisfied or even close to being satisfied. For this reason, this implementation cannot be considered for the end-effectors distance constraint and a second strategy, namely the hard constraint, must be tested.



(a) Trajectory Tracking. (b) Distance between EE1 and EE2.

Figure 26: Soft Constraint.

4.3.2 Hard Constraint

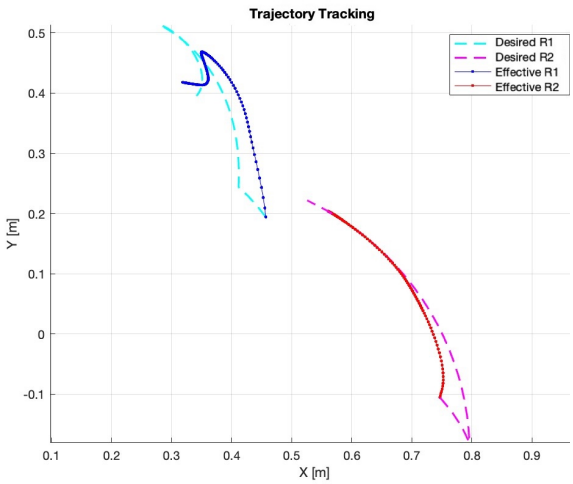
A second solution is therefore presented so that the distance constraint can be introduced into the NMPC algorithm. In the case of hard constraint, the constraint is formulated as a true optimisation restriction, namely for each step i:

$$||p_{1,i} - p_{2,i}||_2^2 \leq d_{max}^2 \quad \forall i \tag{22}$$

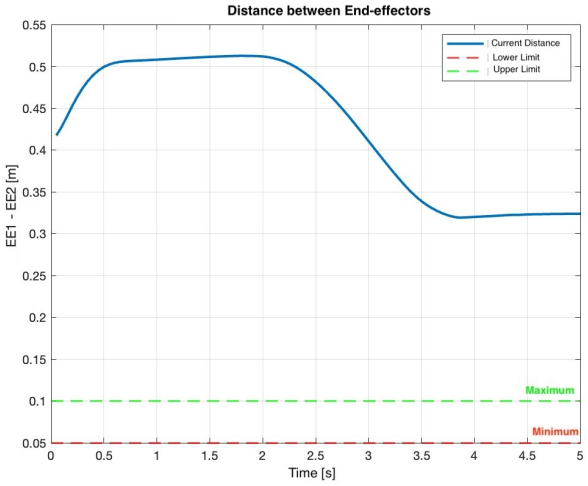
This formulation need now to be integrated into the NMPC function, and the constraint need to be set explicitly. In this way, exceeding the maximum distance between the end-effectors (d_{max}) should be strictly prevented, thus ensuring effective spatial cooperation between the two manipulators.

However, when looking at the results of Figure 27, the constraint is again found not to be respected at any point of the trajectory and the trajectory tracking is not satisfactory as well: robot 1 does not follow the reference trajectory at all (light blue dotted linee in Figure 27a) during the rotation of its penultimate joint, and robot 2 is not able to complete the trajectory and to reach the last point of the reference trajectory (red dotted

linee in Figure 27a), by placing in this way its end effector at an even greater distance from the other.



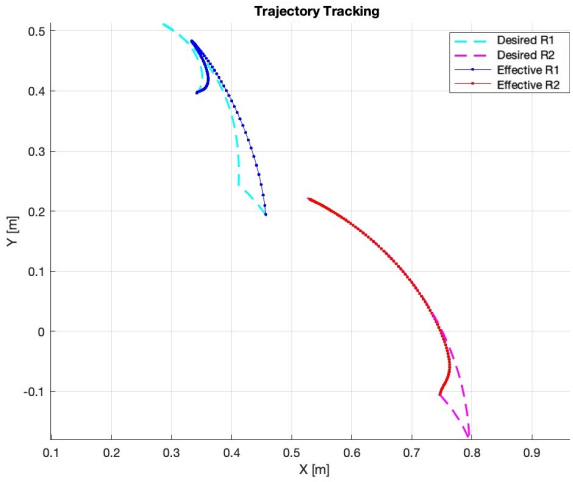
(a) Trajectory Tracking.



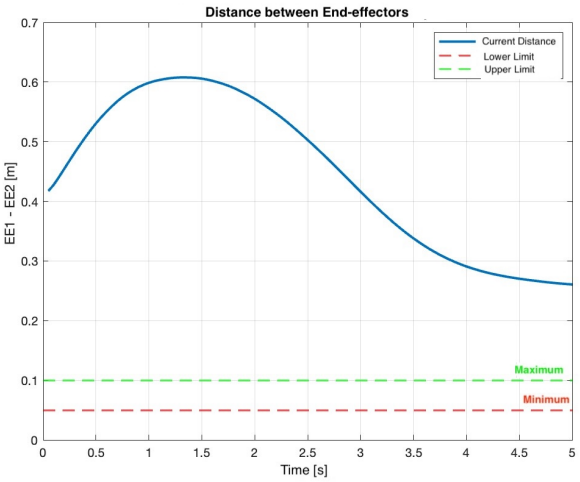
(b) Distance between EE1 and EE2.

Figure 27: Hard Constraint: $d_{max} = 0.1$ m.

The algorithm is also tested with a less stringent constraint, i.e. with a value of $d_{max} = 0.2$, but even if Figure 28 shows improvements both in trajectory tracking and in the graph of the distance between the end-effectors, it is not satisfactory enough, making further study of this constraint necessary.



(a) Trajectory Tracking.



(b) Distance between EE1 and EE2.

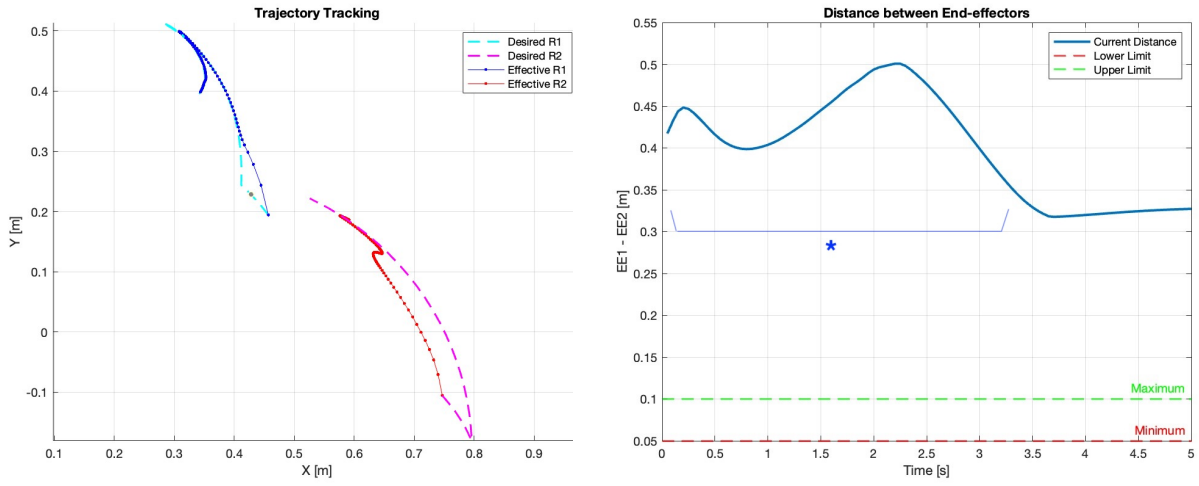
Figure 28: Hard Constraint: $d_{max} = 0.2$ m.

For this reason, the weight matrix Q is modified in order to test the algorithm with different weights for different joints to try to solve the problem. For example, let 's take the weight matrix:

$$Q = \text{diag}([100\ 10\ 1\ 1\ 1\ 1\ 1\ 10\ 10\ 1\ 100\ 1])$$

where different weights are taken into account not only for each joint but also for each robot. In this case, looking at the results of Figure 29 it can be seen that the trajectory tracking of robot 1 improves, but not only that: the system is trying to meet the imposed distance constraint. Unfortunately this is not enough, in fact looking at Figure 29b it can be seen that again it is never satisfied and is not even close to being satisfied, although there is some improvement in the distance in the first part of the trajectories (this part of the trajectory is depicted by: $*$). Several tests with other Q matrices are done but nothing has changed in terms of results.

It can therefore be concluded that, although the modifications made to the weight matrix Q lead to a partial improvement in the system's behavior, they are not sufficient to ensure compliance with the imposed distance constraint. In all tested configurations, the constraint was never fully satisfied, suggesting that the issue lies not so much in the choice of the penalty weights, but rather in the nature of the trajectory imposed on the two robots. Specifically, it is observed that the planned trajectory requires movements involving distances between the end-effectors that exceed the allowable threshold, making the constraint incompatible with the trajectory itself, even under optimization. This behavior indicates that the constraint is structurally infeasible for the given trajectory. Therefore, it is deemed necessary to reconsider the trajectory planning in order to ensure compatibility with the physical constraints of the system. A more flexible trajectory, which takes into account the maximum allowable distance between the two manipulators from the outset, thus appears to be the only way to guarantee the feasibility of the problem.



(a) Trajectory Tracking. (b) Distance between EE1 and EE2.

Figure 29: Hard Constraint: different Q matrix.

4.3.2.1 New OC-RRT Trajectories

In light of the previous observations, it became necessary to redesign the desired trajectory for the two manipulators in order to make it compatible with the constraints imposed by the predictive control strategy, particularly the distance constraint between the end-effectors. It is also worth noting that, in the process of replanning the trajectory, the opportunity is taken to construct a path that not only complies with the imposed constraints, but is also more in line with the actual operational requirements of the welding task, thus being more realistic and representative of a more concrete application scenario. In order to achieve this, the OC-RRT planner is used. The procedure is the same as that presented in Subsubsection 3.2.2, which is suitably adapted to include the geometric constraint directly. To this end, the function defining the constraints is modified and called *New_robotDynamicsConstraint_OCRRT2* instead of *robotDynamicsConstraint_OCeasy* and it is given to *fmincon*. This last new function is therefore the one responsible for integrating the additional distance constraint. It is reported below in Code 6 the pseudocode of it where also the velocity constraint for joints is included. Note then that both constraints belong to inequality constraints and, while the velocities of the joints are required to be less than or equal to π, for the distance between the robot end-effectors a range of 0.03 to 0.10 meters is considered.

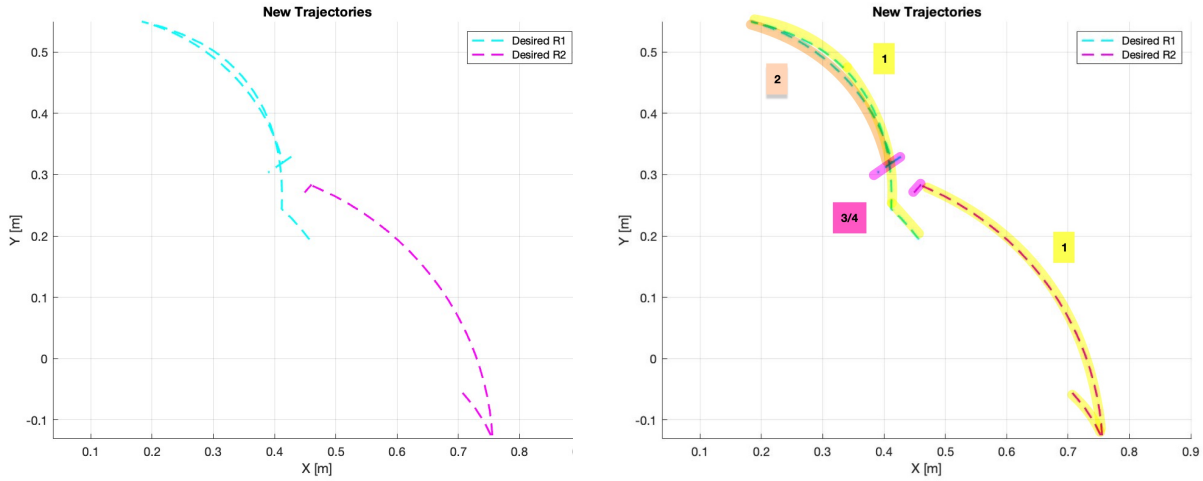
Code 6: New Trajectory - Constraints

```
function [c,ceq]=New_robotDynamicsConstraint_OCRRT2(config,
                                    robot,otherConfig,otherRobot)

  % Get the End-effector Positions (using getTransform)

  %Defining the distance constraint:
  %1) Compute the norm between the end-effectors position
  %2)define minDistance = 0.03 and maxDistance = 0.10; %10cm

  % Distance Constraint
   distanceConstraint = [minDistance - distance;
                         distance - maxDistance];

  % Velocity Constraint
   jointVelLimits = pi*ones(1, length(config));
   jointLimitConstraint = abs(config-otherConfig)-jointVelLimits;

```

```
% Inequality Constraint
c = [distanceConstraint; jointLimitConstraint(:)];
ceq = []; %Equality Constraint
```

Instead, talking about the new movements that constitute the trajectory, there are now four main phases: an upward movement, the rotation of the penultimate joint of robot 1, a welding movement performed by joint 5 of both robots and finally a downward movement to bring down the welded object. It is important to specify that in the first part of the movement, the upward movement, the original OC-RRT algorithms are used, i.e. those without constraints. This is because, even if one tried to impose the constraint, it would be unfeasible due to structural problems, since the end-effectors of the robots in the home configuration are placed at a distance greater than 10 cm. By contrast, the new OC-RRT algorithms including the constraint is used from the following phase onwards. This is not a problem since coordination is required from the moment in which the two end-effectors are aligned in front of each other, just before the welding phase begins. The trajectories are presented in Figure 30 and the various phases of movement are identified by:

- Number 1 identifies the upward movement for both robots.
- Number 2 identifies the rotation of the penultimate joint of robot 1.
- Number 3/4 identify the welding movement and the downward movement (they are overlapped)



(a) New Trajectory Plot. (b) Trajectory's phases.

Figure 30: New Trajectory.

Additionally, a graph of the distances between the two end-effectors along the trajectory planning phase is shown in Figure 31: again, it is very clear when OC-RRT with constraint is applied satisfying the requirement (green line in the graph).

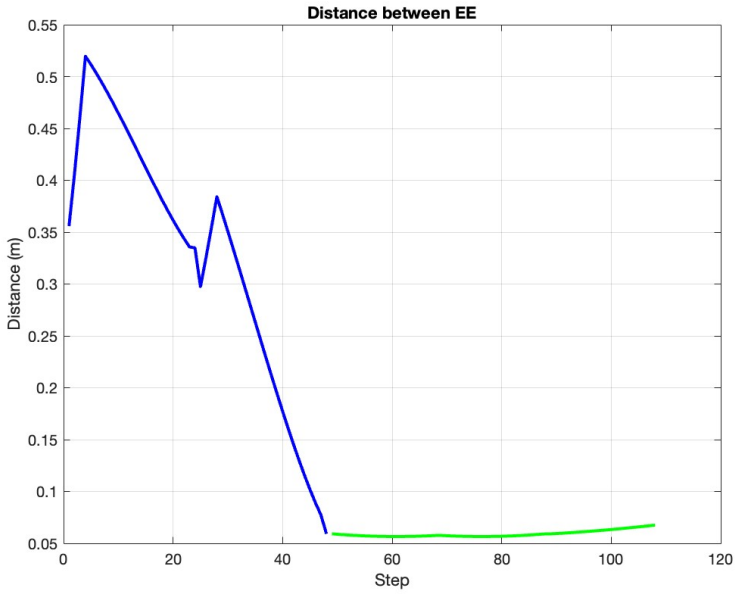
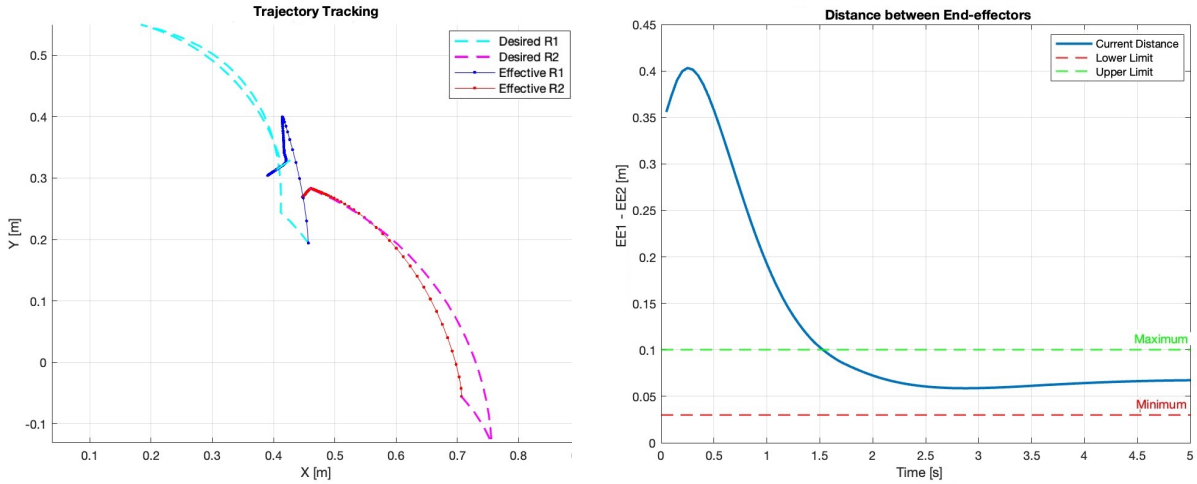


Figure 31: Distance between EE.

Once the new reference trajectory is obtained, designed so that the distance constraint between the end-effectors can be respected, it is provided as input to the NMPC controller. The objective is to verify that the control strategy continues to operate effectively, even with the new trajectory with the additional distance constraint, allowing the system to follow it correctly and maintaining performance consistent with that observed in previous implementations.

By analysing the results reported in Figure 32b, it is observed that the end-effectors distance constraint is respected during most of the planned trajectory. The only violations occur in the initial phase of the path (see phase 1 in Figure 30b), due to intrinsic structural limitations in the configuration of the robots. As far as trajectory tracking is concerned, namely in Figure 32a, there is a slight reduction in performance compared to the cases without constraints, due to the necessary trade-off asked to the controller to satisfy the constraint. However, the overall behaviour of the system is still satisfactory, confirming the effectiveness of the proposed approach.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 32: New Trajectory Performance

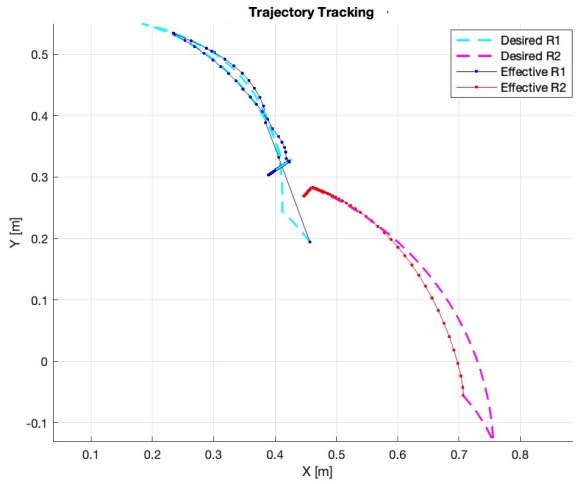
By trial-and-error better results can be obtained using different weight matrices Q. Thus as an example, the result with:

$$Q = \text{diag}([10000\ 10\ 10\ 10\ 20000\ 100\ 1\ 1\ 1\ 1\ 1\ 1])$$

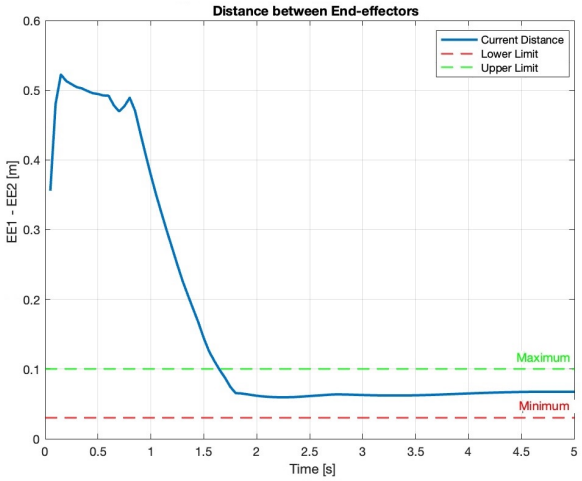
is given here. In fact, in Figure 33 it can be seen that the constraint continues to be satisfied for the major part of the trajectory and a considerable improvement in tracking is visible.

In conclusion, the combination of the new trajectory with the additional conatraint and the NMPC controller, which in turn is integrated with the constraint, has demonstrated an overall effective behaviour, with satisfactory results both in terms of respect of the constraint and tracking accuracy, thus constituting a complete architecture of the system under examination.

In the following chapter, the validation of the robustness of the entire system will be carried out, analysing its ability not only to maintain good performance, but also to guarantee compliance with the constraint imposed in the presence of disturbances and noise on the measurements.



(a) Trajectory Tracking.



(b) Distance between EE.

Figure 33: Performance with a different Q.

5 Validation

This chapter is dedicated to validating the robustness of the entire control system in the presence of external disturbances. After verifying that the proposed NMPC strategy, together with the redefined reference trajectory, is able to guarantee satisfactory performance in tracking and to respect the distance constraints imposed between the manipulators, as well as to better represent the case in reality, it becomes fundamental to analyse the behaviour of the system under more realistic operating conditions. In particular, the aim is to assess whether the system can maintain performance and respect the constraints even in the presence of noise, which may derive from sensory inaccuracies, modelling approximations or external perturbations. This analysis aims not only to verify the general effectiveness of the control strategy, but also to demonstrate its reliability in real scenarios, in which the absence of disturbances cannot be assumed.
To this end, simulations are conducted by introducing additive noise into the system, and the results are analysed to observe the impact on both trajectory tracking and constraint compliance. The observations reported in this chapter are therefore a first step towards validating the applicability of the system in practical contexts.

5.1 Noise to the Controller Input

The first thing that is tested is the robustness of the system under added noise on the sensors, thus taking into account unavoidable imperfections in signal measurement. The procedure is the same as in Subsection 4.1, the codes are modified in the same way and Gaussian noise is considered. The noise is therefore added to the joint positions q and to the joint velocities $\dot{q}$ and the expressions are recalled:

$$q_{noisy} = q_{current} + param_q \cdot \mathcal{N}(0,1)$$
$$\dot{q}_{noisy} = \dot{q}_{current} + param_{\dot{q}} \cdot \mathcal{N}(0,1)$$

where, however, in this case different values of $param_{q,\dot{q}}$ are tested, in particular:

$$\begin{aligned} param_q &= 0.01 \quad \text{or} \quad param_q = 0.001 \\ param_{\dot{q}} &= 0.005 \quad \text{or} \quad param_{\dot{q}} = 0.001 \end{aligned} \tag{23}$$

It should be noted that the values fall perfectly within the ranges of Table 1, which is the

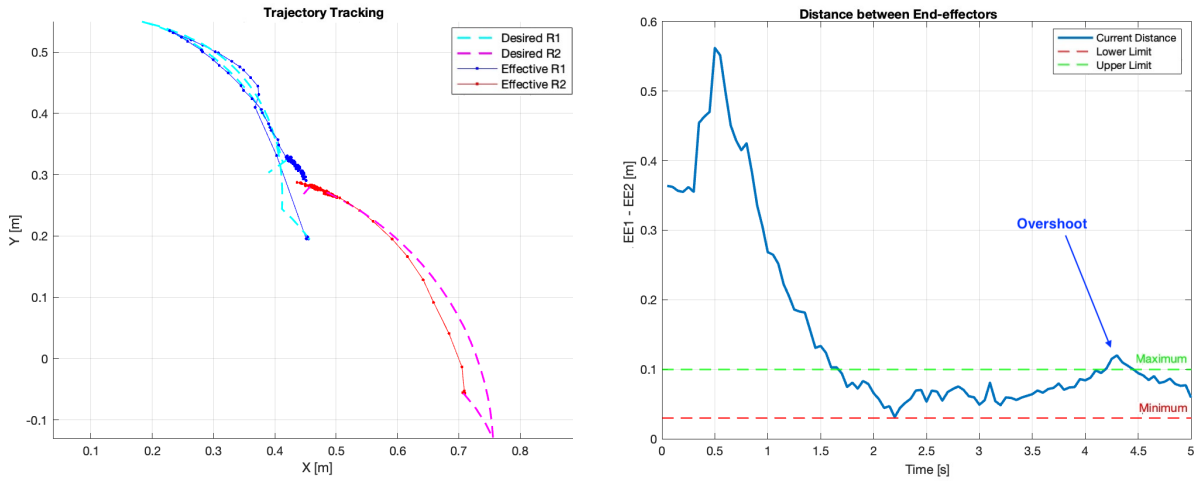
reason why the considerations previously expressed regarding the adoption of orders of magnitude of this type remain valid.

The first simulation is performed using the Q matrix that previously provided the best tracking results, namely:

$$Q = \text{diag}([10000\ 10\ 10\ 10\ 20000\ 100\ 1\ 1\ 1\ 1\ 1\ 1])$$

Robustness to noise is now tested using both noises simultaneously in the presence of maximum error, which means $param_q = 0.01$ and $param_{\dot{q}} = 0.005$ (Figure 34), and in the presence of very small errors, which means $param_q = 0.001$ and $param_{\dot{q}} = 0.001$ (Figure 36).

Analysing the results of Figure 34 it can be affirmed that they are not at all satisfactory: the tracking of the trajectories is not correct, in particular, neither of the two robots can complete the trajectory due to the noises introduced, which is even more clear by looking at Figure 35. Furthermore, looking at the plot of the distances between the end effectors, it can be seen that even here the system fails to satisfy the constraint completely as, apart from the first part of the trajectory where it is allows to fail, towards the end of the path there is an overshoot from the maximum limit. The system cannot therefore be said to be robust in the presence of such an error.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 34: Biggest Error.

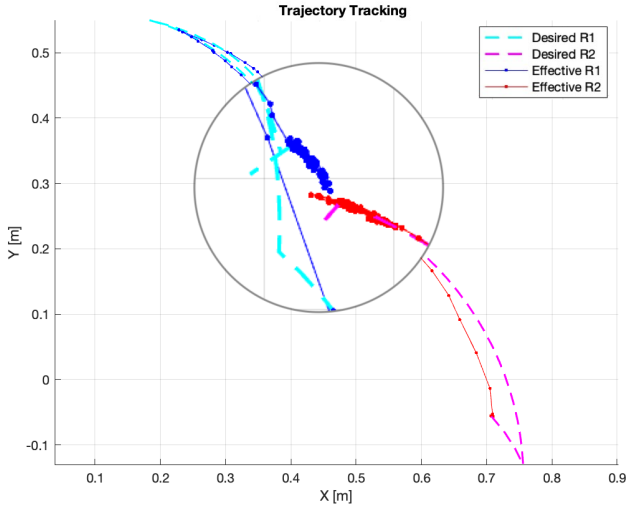
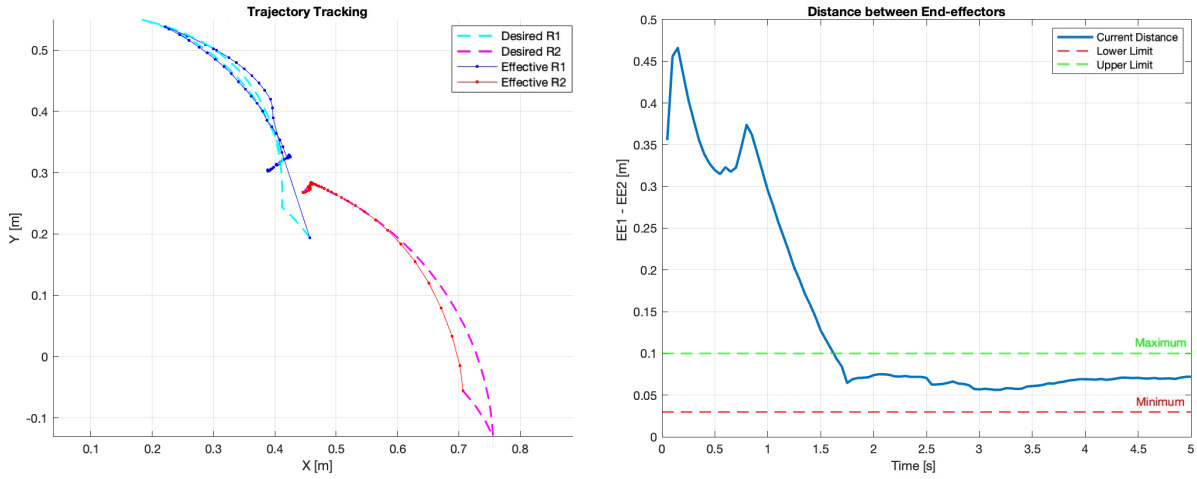


Figure 35: Zoom on the final points of the trajectories.

Turning instead to the validation where the smallest parameters are used, we obtain the results of Figure 36. In this case, things get better: the tracking of the trajectory is very good, although the effect of noise can still be seen, and the constraint is always satisfied in the segments of interest. This therefore makes the system appear robust to noise of this magnitude.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 36: Smallest Error.

However, there is one thing that both of these results have in common: the need for the *try&catch* block in the NMPC code, previously presented in Code 4. This piece of

code comes into action whenever the solver is unable to solve the optimisation problem. In such a case, an error message is printed out and the current robot configuration is maintained: position and speed are preserved, while torque is set to 0 (i.e. no command to the motors). The use of the *try&catch* structure is a safety measure, but should not become a recurring operating condition: although it is not an error at syntax or compilation level, the frequent occurrence of a failure in solving the problem may indicate weaknesses in the formulation of the predictive model or constraints.

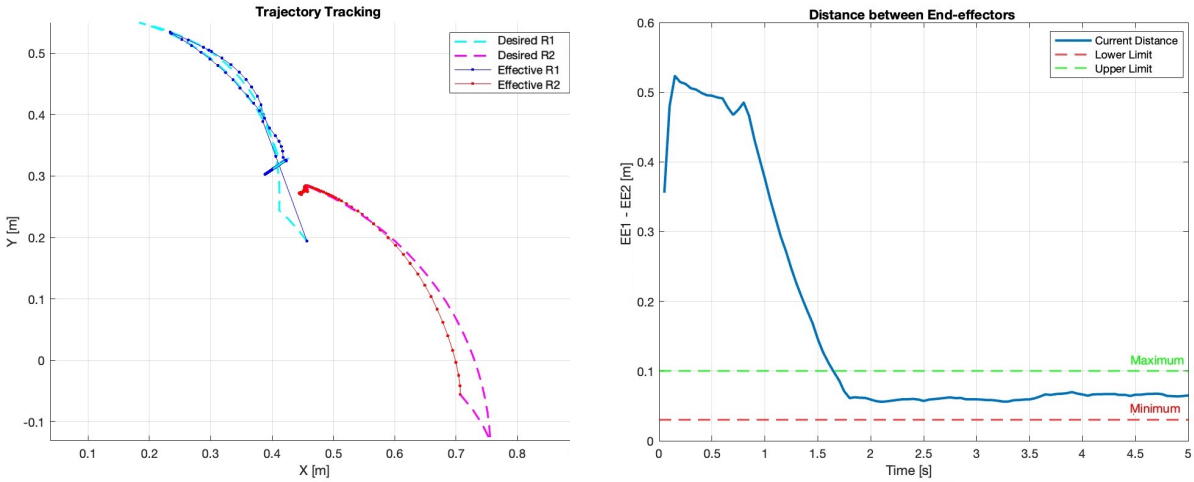
Many trial-and-error attempts were made, both by trying to change the value of the weights in the Q matrix and by trying different settings for the *ipopt* solver, such as tolerances, maximum number of iterations and various initialisation settings. However, no case in which the current system configuration completely avoids the use of the *try&catch* structure has been identified, suggesting that the problem is structurally difficult to solve in a reliable manner with the settings adopted.

5.2 Dynamic Disturbances

The considerations are different, however, when it is decided to perturb the system with another type of disturbance, namely a dynamic disturbance in simulation that acts directly in the computation of the system's dynamics. Remember how this disturbance simulates unmodelled torques, variable friction or inaccuracies in the model and it is modelled as an additional moment to the dynamic model. Again, the disturbance is of Gaussian type and the addition of it to the system follows the same implementation presented in Subsection 4.2.
The results are obtained using the weight matrix already used previously:

$$Q = \text{diag}([10000\ 10\ 10\ 10\ 20000\ 100\ 1\ 1\ 1\ 1\ 1\ 1])$$

and are shown in Figure 37. From the analysis of the simulations, it can be seen that the distance constraint is respected along almost the entire trajectory, particularly in the section of interest (after phase 1 of the trajectories, Figure 30b). Furthermore, the quality of tracking remains high overall, the only exception occurring in the final phase of the path (i.e. phases 3/4), where the effect of the disturbance appears more noticeable, as evidenced in Figure 38. Despite this, it can be concluded that the system shows good adaptability even in the presence of dynamic disturbances.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 37: Dynamic Disturbance.

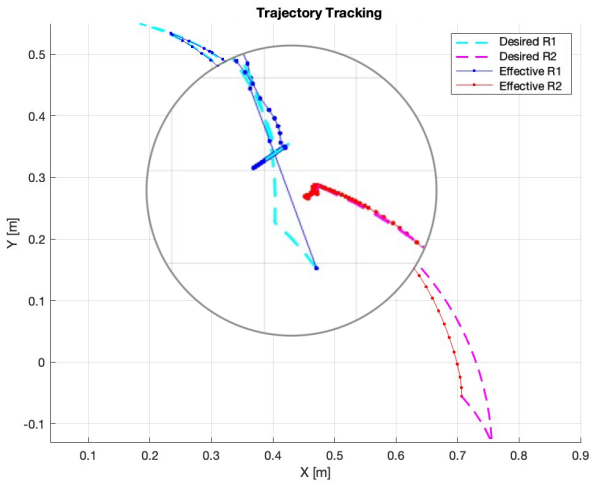


Figure 38: Dynamic Disturbance Effect.

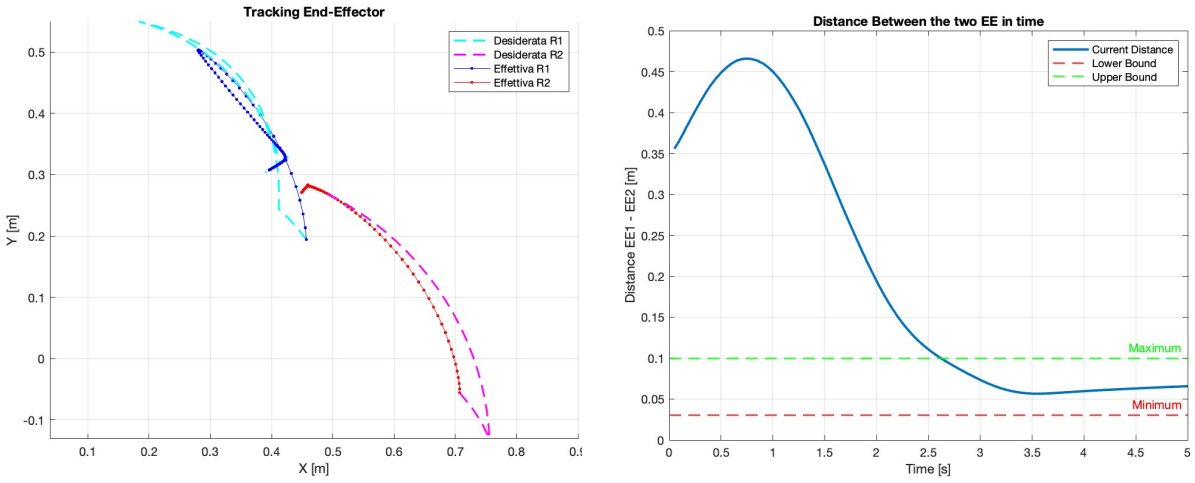
5.3 Validation of the whole architecture

From the above analyses, it has become clear that the inclusion of the distance constraint within the NMPC controller can introduce significant critical issues through the call to the *try&catch* block in the presence of noise on the input, thus compromising the successful execution of the simulation in some cases.

Although the definitive solution to this problem would require a specific in-depth study, which it is considered appropriate to leave as a possible future development, it is decided to adopt an alternative solution that is more functional and reliable in the current

experimental context. In particular, it is decided to remove the geometric constraint from the NMPC controller, keeping it instead in the trajectory planning phase via OC-RRT. This solution aims to guarantee the reliability of the system even under noise conditions, avoiding fallbacks while preserving the effectiveness of tracking and respecting critical constraints.

The results obtained from this new final configuration are presented and discussed in Figure 39. The reference implementations are therefore the same as those introduced in Subsubsection 3.2.2, and Subsubsection 3.3.2 with the only modification of the predictive horizon that is reduced from $N = 10$ to $N = 5$ in order to avoid fallback mechanisms. Furthermore, the matrix Q is defined as $Q = 1.8 \cdot eye()$, it is kept simple as the results obtained are already satisfactory in terms of performance. The trajectory tracking plot (Figure 39a) confirms the effectiveness of the tracking strategy, despite minor inaccuracies at the beginning of the trajectory, especially with respect to robot 2, which is the the one with reference trajectory represented with a pink dotted line and effective trajectory with a red line in the reference figure. Morevoer, regarding the constraint, it is respected along the entire trajectory (Figure 39b) except as usual in the first part of the trajectory (see Figure 30b). Hence, this represent a good trade-off between precision and complexity in the computation.



(a) Trajectory Tracking. (b) Distance Between EE.

Figure 39: Final System Behaviour in Nominal Condition.

Next, the behaviour of the same configuration in the presence of noise on the inputs is analysed. In this scenario, good and consistent results are finally obtained. Recall that joint positions and joint velocities are then modelled as:

$$q_{noisy} = q_{current} + param_q \cdot \mathcal{N}(0, 1)$$

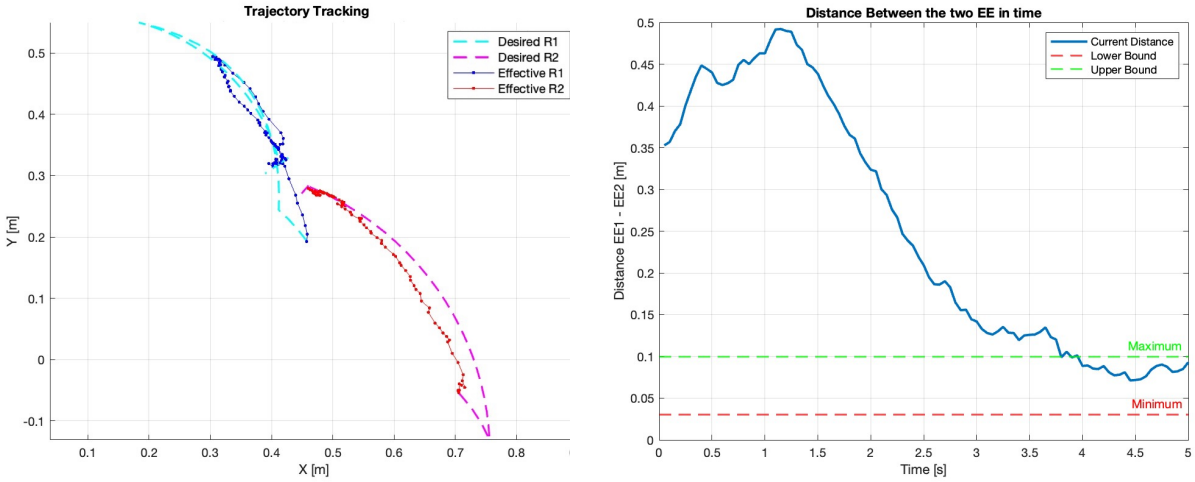
$$\dot{q}_{noisy} = \dot{q}_{current} + param_{\dot{q}} \cdot \mathcal{N}(0, 1)$$

In this case, several values for $param_q$ and $param_{\dot{q}}$ are considered, all of which are compatible with the ranges of Table 1. The use of these parameters in combination gives rise to three test cases, namely:

1. $param_q = 0.01$ and $param_{\dot{q}} = 0.005$
2. $param_q = 0.001$ and $param_{\dot{q}} = 0.005$
3. $param_q = 0.001$ and $param_{\dot{q}} = 0.001$

1. $\mathbf{param_q = 0.01}$ and $\mathbf{param_{\dot{q}} = 0.005}$ (High Noise)

In the first case, the one in which the largest error is taken into account, the predictive horizion set at $N = 5$ is maintained, but the weight matrix Q is reset to $Q = eye(12)$, i.e. a simple identity matrix, in order to not compromise the entire effectiveness of the system behaviour. As shown in Figure 40a, trajectory tracking is affected by the introduction of the disturbance, but the system still manages to complete the trajectory. It is also satisfying to see from Figure 40b that the constraint is respected in the last phases of trajectory tracking. Furthermore, an additional plot is present for these last simulations representing the total system, namely that of the tracking error plot, Figure 41. Although in the initial stages the trajectory tracked presents a significant deviation from the reference trajectory, a progressive improvement in the error is observed, which tends to become zero as the simulation progresses. It can therefore be concluded that the system, in its final configuration, is not only capable of operating under ideal conditions or under dynamic disturbances, but is also capable of effectively handling noise on the inputs.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 40: Big Error Case.

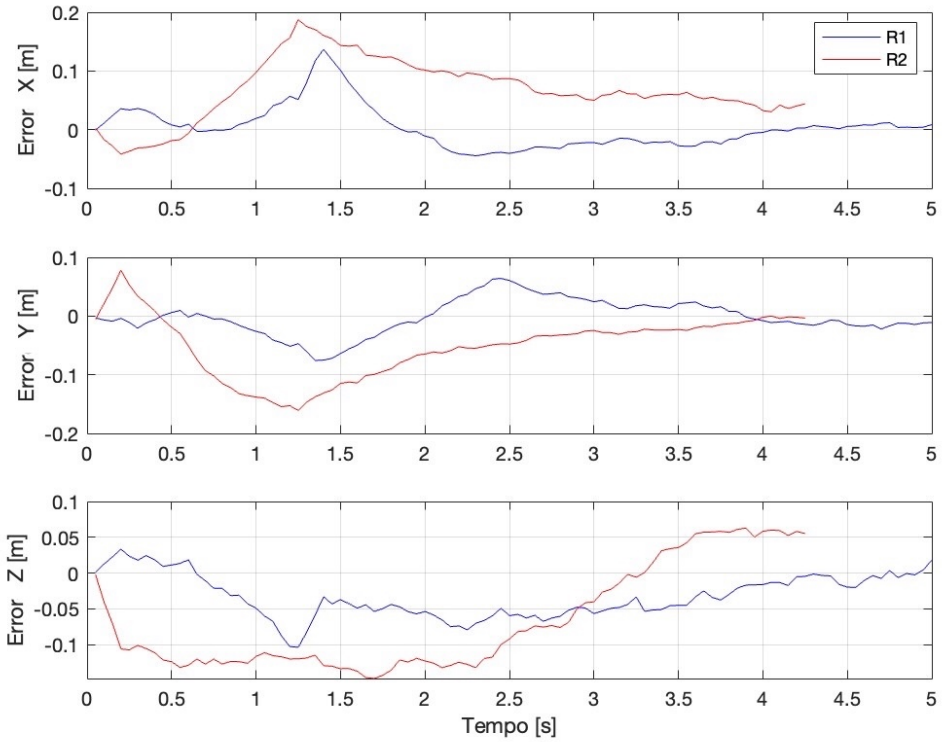
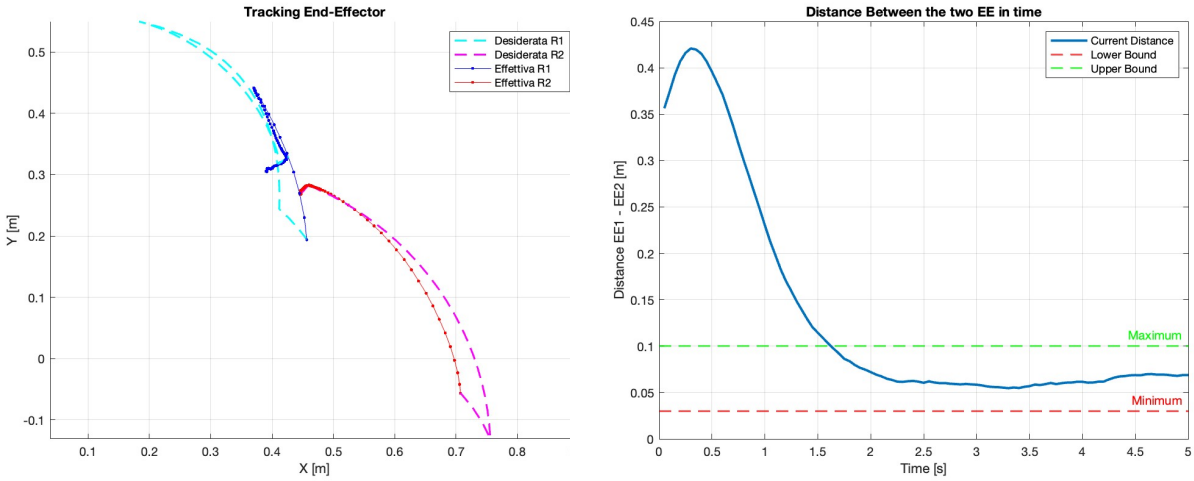


Figure 41: Tracking Error - Big Error Case

2. $\mathbf{param_q = 0.001}$ and $\mathbf{param_{\dot{q}} = 0.005}$ (Medium noise)

In the second case, in which a smaller error is introduced on the joint positions, the system not only continues to guarantee good performance while maintaining $N = 5$ and $Q = eye(12)$, but also allows further improvement of the dynamic behaviour without compromising the succesfull execution of the simulation, by adopting more ambitious parameters such as $N = 8$ and $Q = 1.5 \cdot eye(12)$. Compared to the previous case, the disturbance exhibits a less noticeable influence on the trajectory tracking, as shown in Figure 42a, and furthermore, the distance constraint begins to be satisfied earlier in the trajectory, as highlighted in Figure 42b. Moreover, the analysis of the tracking error in Figure 43 also confirms these improvements: the introduction of noise has a smaller impact, and the error tends to reduce more rapidly, converging to zero more quickly. These results show greater resilience of the system to noise, confirming the effectiveness of the configuration adopted.



(a) Trajectory Tracking.

(b) Distance between EE.

Figure 42: Mid Error Case.

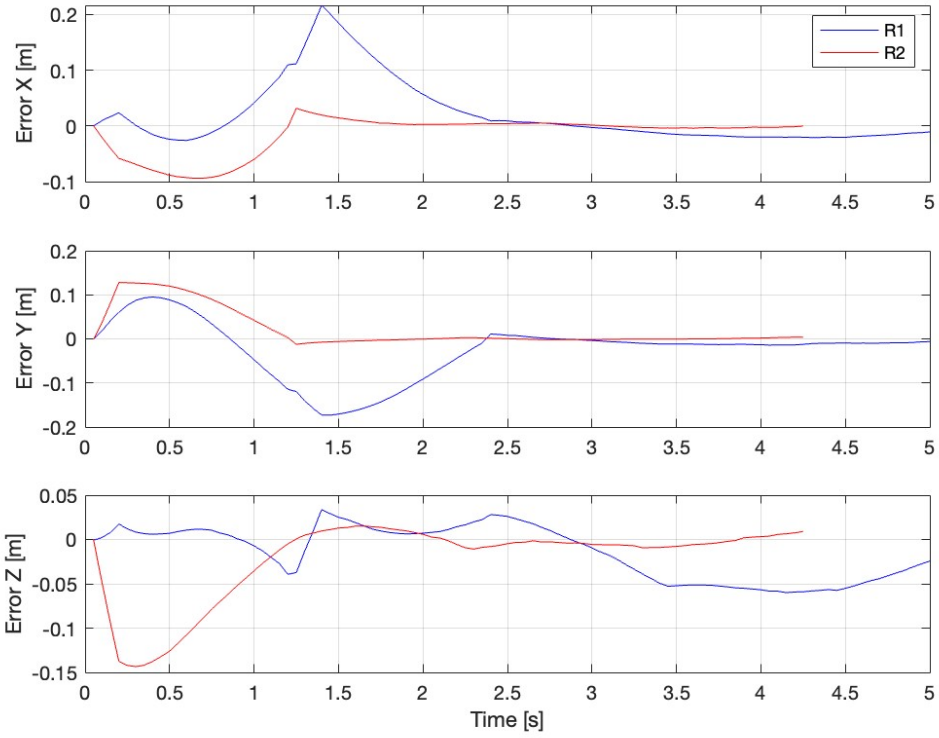
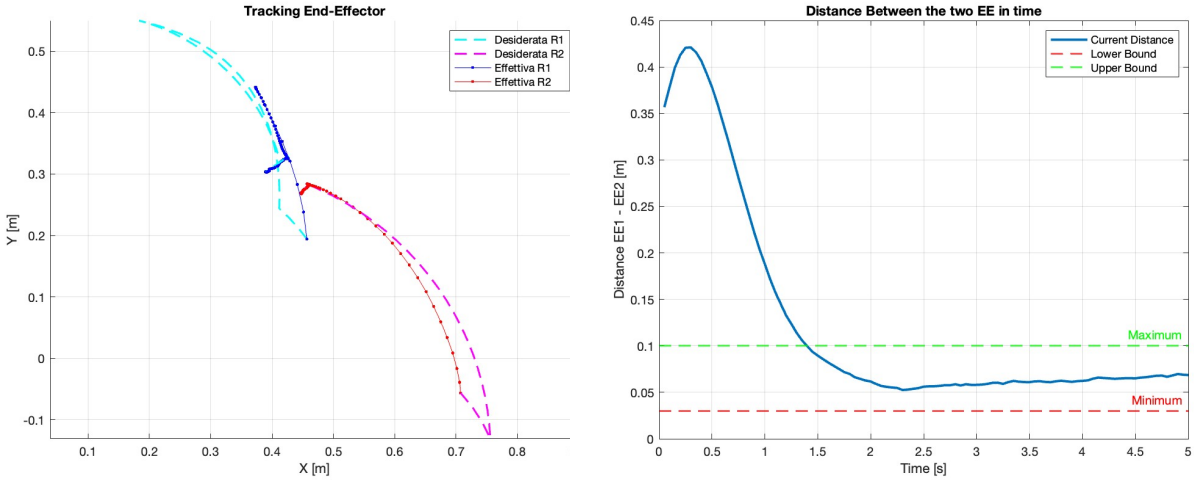


Figure 43: Tracking Error - Mid Error Case

3. $\mathbf{param_q = 0.001}$ and $\mathbf{param_{\dot{q}} = 0.001}$ (Low Noise)

The last case analysed concerns the introduction of an error of minimum magnitude, i.e. the smallest noise considered reasonable for the purposes of this analysis. In this configuration, the prediction horizon is maintained at $N = 8$, while the weight matrix is further increased, taking the form $Q = 2 \cdot eye(12)$. The behaviour of the system is almost indistinguishable from that observed in the absence of noise (see Figure 39): the trajectory tracking remains accurate (Figure 44a), the constraint on the distance between the end-effectors is respected in the phases of interest of the trajectory (Figure 44b), and the tracking error rapidly converges to zero (Figure 45). Therefore, even in this scenario, and even more than in the previous ones, the system demonstrates an extremely effective response to input perturbations.



(a) Trajectory Tracking. (b) Distance between EE.

Figure 44: Small Error Case.

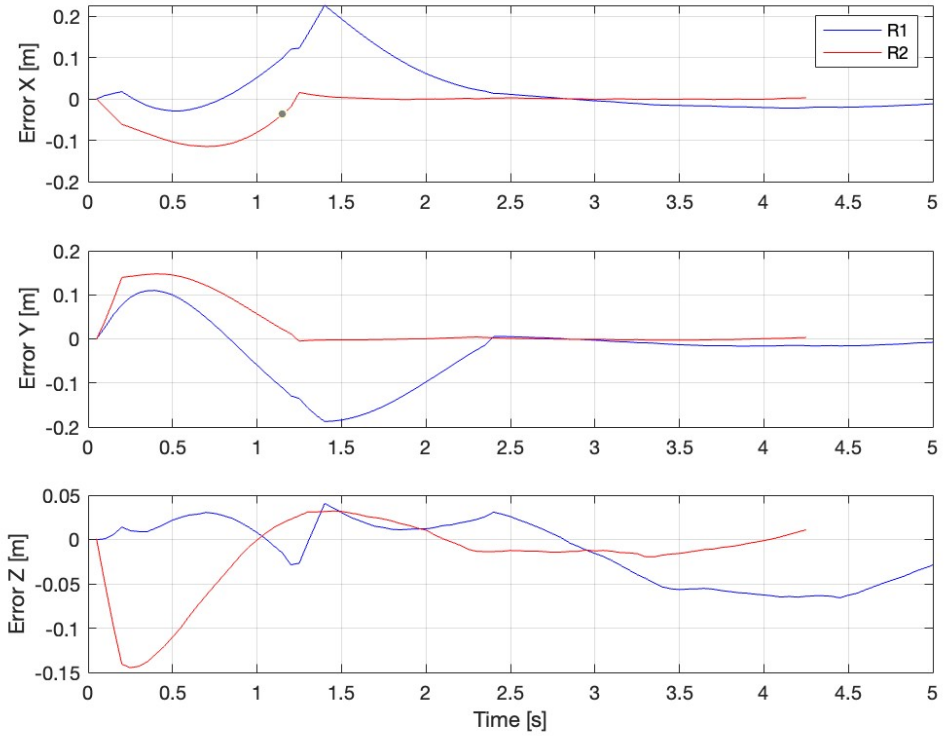


Figure 45: Tracking Error - Small Error Case

It is concluded, as expected, that the lower the magnitude of the noise, the better the performance achieved in terms of constraint compliance and trajectory tracking accu-

racy.

The results obtained in this section demonstrate how the final strategy adopted, involving trajectory planning using OC-RRT with integrated distance constraint and the use of the NMPC controller without constraint, represents an effective and robust solution. In particular, this configuration allows good tracking performance and compliance with the geometric constraint to be maintained even in the presence of significant disturbances. The three test cases analysed confirm the system's ability to adapt to different noise conditions without compromising its overall behaviour, thus this approach appears promising for application in real-world scenarios, while leaving the field open for future developments aimed at a more direct management of constraints within predictive control.

Finally, it should be noted as a last consideration that the test with dynamic disturbance was not repeated, as the results obtained previously have already shown that this type of disturbance does not compromise the functioning of the system or generate fallback phenomena. Its re-execution, therefore, would have produced behaviour almost identical to that already observed, being redundant for the purposes of the overall validation.

6 Project Planning & Budget

This section presents the temporal organisation and an estimate of the costs associated with the realization of this project. Firstly, a time planning analysis is introduced to illustrate how the development process is structured over the months of work. Here, a breakdown of the main tasks is provided, together with their scheduling, to show the project evolution. Secondly, a detailed cost estimation is presented. Since the entire project is carried out in a simulated environment, the main cost items concern: software licences, energy consumption for simulations, and costs related to human labour. It is worth noting that no physical components or laboratory related costs are involved.

6.1 Temporary Planning

In order to complete a project such as the one presented in this thesis, an incremental development strategy is used, based on a step-by-step approach, which made it possible to build an increasingly complex and efficient system. The work was carried out over a period of about five months and involved a series of consequential phases.

Each phase forms the basis for the development of the next, with the aim of realising a complete architecture capable of dealing with the complexity of a double arm robotic system in a realistic simulated context, the ultimate goal of which is the execution of an automated welding operation. The following table represents the Gantt chart and summarises the main activities over the five months in chronological order:

Table 2: Gantt chart of the project activities

Task	February	March	April	May	June
State of Art and Inizialization Setup	X	X	X		
Planning using RRT			X		
Planning using OC-RRT			X	X	X
Control Architecture (MPC)				X	
Control Architecture (NMPC)				X	X
System Constraints				X	X
Noise and Disturbance Modeling				X	X
Validation		X	X	X	X
Thesis Writing					X

As can be seen from the diagram, the project started with a state-of-the-art study phase, which continued in parallel for most of the development. Subsequently, it progressed to the trajectory planning phase, initially using the RRT algorithm, and then evolved into its OC-RRT extension. The same thing is done with regard to the control architecture, where after implementing a strategy based on MPC, it moved on to its more advanced and realistic version, namely the NMPC. In the last period of work then, an effort is made to bring the system as close as possible to real operating conditions through the introduction of constraints, noises and dynamic disturbances. On the other hand, it should be noted that the only thing that is presented in almost all the months of work is the validation step, a fundamental aspect for monitoring and verifying the correct behaviour of the entire system in all its evolution. Finally, the last part of the work is dedicated to the development of this document.

This time planning made it possible to progressively address the various technical challenges, maintaining a clear vision of the objectives and ensuring a controlled evolution of the entire project.

6.2 Licensing Cost

The project is implemented entirely in the MATLAB environment, with the support of open-source toolboxes and libraries: YALMIP, CasADi and Robotics System Toolbox. The licence costs are reported below in Table 3. However, it is useful to mention the cost that would be incurred if the project were conducted in a non-academic context (e.g. a company), where the purchase of the MATLAB licence and the necessary toolboxes may represent a significant item of expenditure.

Table 3: Software Licensing Costs

Software	**License**	**Cost (€)**
MATLAB + Toolboxes (e.g. Robotic System Toolbox)	Academic	0.00
MATLAB + Toolboxes (e.g. Robotic System Toolbox)	Commercial	4,355.00 ([34])
YALMIP	Open-source	0.00
CasADi	Open-source	0.00
Total (academic environment)		**0.00**
Total (commercial environment)		**4,355.00**

It is concluded that no direct software costs were incurred in this case due to the use of university licences and open-source tools.

6.3 Energy Cost

Even though the project does not require the use of physical machinery or laboratory equipment, the energy consumption associated with numerical simulations should be taken into account. All simulations, data analysis and code development were carried out on a personal laptop computer (a MacBook Air with an Apple M1 chip), for an estimated total of approximately 750 working hours.

According to reliable technical sources ([35] and [36]), the average consumption under moderate load for a MacBook Air M1 does not exceed 30 W, while the national average energy cost amounts to 0.16 €/kWh. Based on these values, the overall energy consumption can be estimated as follows:

$$750\ [h] \cdot 0.030\ [kW] = 22.5\ [kWh]$$

$$22.5\ [kWh] \cdot 0.16\ [€/kWh] = 3.60\ [€]$$

Therefore, the total energy cost for the entire project is about € 3.75, a low and sustainable value for the whole project, which confirms the efficiency of the device used.

6.4 Human Labor Cost

The successful completion of this project requires the contribution of a qualified engineer throughout all phases of the development. The overall effort is estimated at approximately 750 working hours and assuming a standard engineering rate of € 50 per hour the total labor cost result to be as reported in Table 4.

Table 4: Human labor cost

Role	**Hourly Rate (€)**	**Hours**	**Total Cost (€)**
Engineer	50	750	37,500.00

6.5 Total cost

By combining the three main cost categories, namely software, energy, and labor, it is possible to estimate the total cost of the project as follows:

Table 5: Total Cost

Item	**Cost (€)**
Software licenses	0.00
Energy consumption	3.60
Human labor	37,500.00
Total	**37,503.60**

The entire project development was possible with a low investment due to the use of free resources and academic licences, with the main cost being human labour rather than tools or materials. Due to the use of open-source tools, access to free university licences and a fully simulated environment, the project demonstrated high cost-efficiency.

In future scenarios, if the system were to be implemented on real physical robots, costs related to hardware, sensors and maintenance would also have to be incorporated into the budget.

7 Environmental impact

The environmental impact of this thesis work can be considered minimal, since it is a project carried out entirely in a simulation environment. Therefore, no physical materials, experimental laboratories, or real mechanical devices were involved, and the only resources used are software and computation work performed on a personal computer.

At this point, in order to estimate the environmental impact in terms of CO_2 emissions, the electrical energy consumed during the development, simulation and validation activities is calculated: the device used is a MacBook Air with an Apple M1 chip, which has an estimated energy consumption of 30 W, and the total work required is 750 hours. This results in a total energy consumption of 22.5 kWh.

Then, considering the electricity mix published by the *Comisión Nacional de los Mercados y la Competencia* (*CNMC*) on 24^{th} April 2025 ([37]), it can be observed that the average emission factor of the Spanish network for energy production is 283 gCO_2eq/kWh. Therefore, applying this coefficient to the energy consumption of the project gives a total emission of approximately:

$$22,5\ [kWh] \cdot 283\ [gCO_2eq/kWh] = 6367,5\ [gCO_2eq]\ = 6,37\ [kgCO_2eq]$$

where $[CO_2eq]$ means equivalent carbon dioxide. It can be stated that this quantity can be considered negligible compared to other types of experimental or industrial activities.

Finally, no significant transportation was required, and no materials were purchased from remote locations. The project did not involve transport, shipping or the use of equipment with a significant environmental impact.

In conclusion, the project was conducted in a highly sustainable manner. In particular, the choice of open-source software tools and the use of low-power devices helped to reduce the overall environmental impact of the work.

8 Conclusions

This thesis investigates the development and the validation of a planning and control architecture for a dual arm robot, consisting of two UR3 manipulators, which has the task of performing a weld between two workpieces, each of which is manipulated by one of the two robots. The two robotic arms must therefore be able to move in a coordinated and precise manner to complete the task while avoiding collisions both with each other and with the environment.

After studying the fundamental theoretical concepts for the current scenario and analysing the state of the art, the complete framework is designed in a simulated environment, using a step-by-step startegy aimed at making the architecture progressively more sophisticated and efficient. The ultimate solution is based on the integration of two advanced methodologies. As far as path planning is concerned, the algorithm called Orientation-Constrained Rapidly-exploring Random Tree (OC-RRT) is exploited, which generates optimised trajectories not only from a cost point of view but also taking into account and guaranteeing collision-free, geometric constraints, joint constraints and additional constraints as desired in the planning phase. Unlike its predecessor, the RRT, which only operates in configuration space and does not guarantee an optimal solution, the OC-RRT offers an optimal solution more suitable for real physical systems.

On the other hand, if control is considered, a Nonlinear Model Predictive Control (NMPC) is chosen instead, which is an extension of the classical MPC and overcomes the limitations of the latter. In fact, thanks to the NMPC, it is possible to obtain more accurate predictions and better and more realistic performance using the complete nonlinear dynamic model of the system. It is important to note that the controller is developed not to consider the two manipulator arms as two separate and independent units, but rather its construction reflects that of a 12 degree-of-fredom NMPC, with the aim of achieving better results in terms of coordination and control.

The system also adapts to realistic scenarios by setting an additional distance constraint between the end effectors of the two UR3 arms: first in trajectory planning, then also in the NMPC controller. However, beware of the case in which this constraint coexists in both: the system does not respond correctly when perturbed by noise in the inputs. For this reason, the final proposed architecture consists of OC-RRT with constraint for the trajectory planning part and NMPC with 12 degree-of-freedom for system control.

Using an architecture of this type, the system adapts well to realistic scenarios: in the

presence of sensor noise, dynamic disturbances, and environmental constraints, it remains reliable. This is evident in the validation part of this thesis carried out in MATLAB, which clearly highlights the architecture's ability to maintain good performance in terms of tracking accuracy and respect of the constraints, while managing conditions that are not only nominal.

In more extensive terms, the thesis demonstrates how the integration of techniques derived from advanced academic research can result in architectures potentially applicable in industrial and real-world environments. Although the work is developed entirely in a simulated environment, the results provide a solid basis for future implementation on real hardware.

Lastly, it is important to emphasise that, in addition to the strictly technical aspects, attention has been paid to the evaluation of the project budget and the environmental impact of the work carried out, thus integrating a more comprehensive and responsible view of the engineering work.

Although the project is completed in terms of its objectives, it leaves the way for several future developments, which are discussed in the following section.

8.1 Future work

Although the proposed architecture provides a complete solution to the problem, several opportunities for improvement, extension, and further investigation still emerge.

The first of these is to find a way to integrate the constraint also within the NMPC without relying on fallback calls to solve the problem when the system is perturbed by noise in the inputs. This is followed by the implementation on real UR3 robotic arms, a fundamental step to validate the robustness and reliability of the system under real-world operating conditions.

Another proposal to enhance the system is the integration of computer vision. Currently, the perception of the environment is modeled in a static or idealized way, but by including a vision system, it would be possible to detect and track the parts in real time, improving the system's adaptability to scenarios that are either partially unknown or subject to changes.

Finally, further extensions of the work could involve the introduction of force control strategies, in order to improve the physical interaction between the two robotic arms

and between the arms and the surrounding environment. This approach would be particularly useful since the task studied requires force regulation both during the welding phase and during the collaborative manipulation phase.

The above-mentioned proposals represent just a few of the many possible directions in which this work could be expanded. The modular structure of the developed architecture provides a solid foundation for adressing these challenges further, favouring the development of more advanced and efficient systems.

Bibliography

[1] Younsung Choi, Donghyung Kim, Soonwoong Hwang, Hyeonguk Kim, Namwun Kim, and Changsoo Han. Dual-arm robot motion planning for collision avoidance using b-spline curve. *International journal of precision engineering and manufacturing*, 18:835–843, 2017.

[2] Shuji Yang, Yonglei Zhang, Hao Wen, and Dongping Jin. Coordinated control of dual-arm robot on space structure for capturing space targets. *Advances in Space Research*, 71(5):2437–2448, 2023.

[3] Liding Zhang, Kuanqi Cai, Zewei Sun, Zhenshan Bing, Chaoqun Wang, Luis Figueredo, Sami Haddadin, and Alois Knoll. Motion planning for robotics: A review for sampling-based planners. *Biomimetic Intelligence and Robotics*, page 100207, 2025.

[4] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[5] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, pages 2997–3004. IEEE, 2014.

[6] Ming Jiang, Ming-Qu Fan, Ai-Min Li, Xue-Wen Rong, Hui Kong, and Rui Song. Coordination control of dual-arm robot based on modeled predictive control. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 495–499. IEEE, 2016.

[7] Chong Wu, Kai Guo, and Jie Sun. Dual pid adaptive variable impedance constant force control for grinding robot. *Applied Sciences*, 13(21):11635, 2023.

[8] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V Dimarogonas, and Danica Kragic. Dual arm manipulation—a survey. *Robotics and Autonomous systems*, 60(10):1340–1353, 2012.

[9] T Tarn, A Bejczy, and Xiaoping Yun. Coordinated control of two robot arms. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3,

pages 1193–1202. IEEE, 1986.

[10] Andrea Carron, Elena Arcari, Martin Wermelinger, Lukas Hewing, Marco Hutter, and Melanie N. Zeilinger. Data-driven model predictive control for trajectory tracking with a robotic arm. *IEEE Robotics and Automation Letters*, 4(4):3758–3765, 2019.

[11] Timm Faulwasser and Rolf Findeisen. Nonlinear model predictive control for constrained output path following. *IEEE Transactions on Automatic Control*, 61(4):1026–1039, 2016.

[12] Timm Faulwasser, Tobias Weber, Pablo Zometa, and Rolf Findeisen. Implementation of nonlinear model predictive path-following control for an industrial robot. *IEEE Transactions on Control Systems Technology*, 25(4):1505–1511, 2017.

[13] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022.

[14] Qihan Wu, Meng Li, Xiaozhi Qi, Ying Hu, Bing Li, and Jianwei Zhang. Coordinated control of a dual-arm robot for surgical instrument sorting tasks. *Robotics and Autonomous Systems*, 112:1–12, 2019.

[15] Guirong Han and Xubing Chen. Dual robot coordinated welding trajectory planning for single y-groove weld seam of plug-in cross pipe. In *2020 5th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 269–275. IEEE, 2020.

[16] Charles A Meehan, Mark Roberts, and Laura M Hiatt. Asynchronous motion planning and execution for a dual-arm robot. In *Workshop on Planning and Robotics: ICAPS*, 2022.

[17] Jiangping Wang, Shirong Liu, Botao Zhang, and Changbin Yu. Inverse kinematics-based motion planning for dual-arm robot with orientation constraints. *International Journal of Advanced Robotic Systems*, 16(2):1729881419836858, 2019.

[18] David Bacher, John R. Cooper, and Javier Puig Navarro. Trajectory generation with load constraints for robotic manipulators. Technical Report NASA/TP–20230013437, NASA Jet Propulsion Laboratory, 2023. Presented at

AIAA ASCEND 2023.

[19] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[20] Roudabe Seif and Mohammadreza Asghari Oskoei. Mobile robot path planning by rrt* in dynamic environments. *International journal of intelligent systems and applications*, 7(5):24–30, 2015.

[21] J. Doyle. Guaranteed margins for lqg regulators. *IEEE Transactions on Automatic Control*, 23(4):756–757, 1978.

[22] N. Lehtomaki, N. Sandell, and M. Athans. Robustness results in linear-quadratic gaussian based multivariable control designs. *IEEE Transactions on Automatic Control*, 26(1):75–93, 1981.

[23] A. Bemporad, M. Morari, and N. L. Ricker. *Model Predictive Control Toolbox User's Guide*. The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, U.S., 2014.

[24] The MathWorks Inc. *MATLAB version: 9.13.0 (R2022b)*. The MathWorks Inc., Natick, Massachusetts, United States, 2022. `https://www.mathworks.com`.

[25] The MathWorks, Inc. *Robotics System Toolbox*. The MathWorks, Inc., Natick, Massachusetts, United States, 2022. Version 4.1 (R2022b).

[26] Johan Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, pages 284–289. IEEE, 2004.

[27] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[28] Frank E. Curtis, Olaf Schenk, and Andreas Wächter. An interior-point algorithm for large-scale nonlinear optimization with inexact step computations. *SIAM Journal on Scientific Computing*, 32:3447–3475, 01 2010.

[29] Universal Robots. UR3: The world's first table-top cobot. `https://video.universal-robots.com/ur3-the-worlds-first-table-top-cobot`, 2024.

[30] Jordi Pelegrí. características y aplicaciones - universal robots ur5. https://www.universal-robots.com/es/blog/universal-robots-ur5/, July 2022.

[31] ABB. *IRB 120, data sheet*, 2019. Part of Product manual - IRB 120, Revision J.

[32] KUKA. LBR iiwa: Robot sensitivo para la automatización industrial. https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/lbr-iiwa, 2024.

[33] Universal Robots A/S. Ur3 technical specifications. Technical Datasheet Item no. 110103, Universal Robots, Odense, Denmark, March 2015. Published March 2015, rev. EN 03/2015.

[34] MathWorks. Matlab pricing and licensing. https://it.mathworks.com/pricing-licensing.html, 2025.

[35] Apple Inc. Macbook air (m1, 2020) - specifiche tecniche, 2025.

[36] Red Eléctrica de España. Pvpc - precios voluntarios para el pequeño consumidor, 2025.

[37] Generalitat de Catalunya. Factors d'emissió associats a l'energia. https://canviclimatic.gencat.cat/en/actua/factors_demissio_associats_a_lenergia/, 2025.