POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Obstacle Avoidance for an Holonomic Robotic Manipulator with Constraint-Based Model Predictive Control

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Gonzalo Jesús Meza Pérez**

Student ID: 992711
Advisor: Prof. Lorenzo Fagiano
Co-advisors: Kristoffer Løwenstein
Academic Year: 2023-2024

Acknowledgements

Quisiera expresar mi más profundo agradecimiento a mi familia, que ha sido mi apoyo incondicional a lo largo de este camino. A mi difunto padre, cuya memoria continúa enseñandome, inspirándome y guiándome, le debo tanta gratitud que las palabras apenas podrían expresar. Su espíritu de justicia, sabiduría, alegría y amor siguen siendo una luz guía en mi vida. A mi madre, mi más sincero agradecimiento por su amor sin límites, apoyo y sacrificio. Su fuerza e inteligencia, su capacidad para enfrentar desafíos, los miles de últimos esfuerzos, y su dedicación a sus empeños académicos y profesionales, han sido una inagotable fuente de admiración e inspiración. A mi hermana, por su compañia, su incondicional apoyo e incesante alegría, que sin ello esto no hubiera sido posible.

My profound gratitude to my supervisor Professor Lorenzo Fagiano, along with my co-advisor Kristoffer Løwenstein, for their continuous and timely support during the development of this work, for the advice, the discussions, and the encouragement.

Through the past years, different colleagues have been by my side making this journey much more enjoyable. I would like to personally thank them Alp, Toñito, Jesucristo, Lloyd, Piero, Ruxandra, Lenin, Santi and my cousin Jordan for their company studying or outside the academic context. All of you will always have a place in my heart.

I would also like to thank the Italian government for its financial support during the course of the master's degree and to all my professors at Politecnico di Milano that I had the gratitude to know. For their invaluable contribution over the past years, not only in terms of imparting essential knowledge and technical skills but also in instilling a deep passion for the academic disciplines and scientific research.

Abstract

In the field of robotics, ensuring the safe and smooth motion of robotic arms within dynamic environments is a critical challenge. This thesis addresses this challenge by presenting a novel solution for obstacle avoidance and inverse kinematics control for a 6 degrees of freedom robotic arm in an unknown environment.

The objective is to safely drive the robot arm to a accomplish a desired task in an unknown workspace by relying only on a partial description of its environment provided by an exteroceptive sensor. In the design of the overall architecture, a hierarchical control system was adopted: a high-level strategy for obstacle free path generation, a medium level Model Predictive Controller, and low level controllers for set-point tracking.

In the context of Local Path Planning, an innovative approach based on the construction of obstacle free convex polytopes is proposed. This method generates intermediate obstacle free trajectories inside the convex sets of safe regions by exploiting only local sensor measurements without the need of an a-priori description of the environment.

To achieve precise end-effector positioning in the workspace, a smooth Inverse Kinematics-Model Predictive Control (IK-MPC) was developed. Since the IK-MPC is recomputed at every control iteration, it is possible to deal with dynamic and unknown scenarios. The formulation of latter optimization problem was based on a Quadratic Programming(QP) approach which allowed the inclusion of motion constraints: position, velocity and acceleration, together with obstacle collision and self-collision avoidance constraints. The linear-quadratic problem formulation has been retained while extending these constraints across a prediction horizon.

The formulation of the overall system architecture ensures a safety task achievement of the robotic arm through complex and dynamic environments respecting physical constraints of the chosen robotic model. The tests were performed on a developed Digital Twin of a MyCobot280 Robotic Arm model that shows the effectiveness of the proposed approach.

Keywords: Model Predictive Control, Inverse Kinematics, Optimization, Quadratic Programming, Local Path Planner

Abstract in lingua italiana

Nel campo della robotica, garantire il movimento sicuro e fluido di bracci robotici in ambienti dinamici rappresenta una sfida critica. Questa tesi affronta tale sfida presentando una soluzione innovativa per l'evitamento degli ostacoli e il controllo cinematico inverso per un braccio robotico a 6 gradi di libertà in un ambiente sconosciuto.

L'obiettivo è guidare in modo sicuro il braccio robotico per completare un compito desiderato in uno spazio di lavoro sconosciuto, basandosi solo su una descrizione parziale dell'ambiente fornita da un sensore esterocettivo. Nella progettazione dell'architettura complessiva, è stato adottato un sistema di controllo gerarchico: una strategia di alto livello per la generazione di percorsi liberi da ostacoli, un controllore predittivo di livello medio e controllori di basso livello per il tracciamento dei *set-point*.

Nel contesto della pianificazione di percorsi locali, viene proposto un approccio innovativo basato sulla costruzione di poliedri convessi liberi da ostacoli. Questo metodo genera traiettorie libere da ostacoli all'interno dei poliedri convessi delle regioni sicure, sfruttando solo misurazioni sensoriali locali senza la necessità di una descrizione a priori dell'ambiente.

Per ottenere un posizionamento preciso dell'effettore finale nello spazio di lavoro, è stato sviluppato un *Inverse Kinematics-Model Predictive Control*(IK-MPC). Poiché l'IK-MPC viene ricalcolato ad ogni iterazione di controllo, è possibile affrontare scenari dinamici e sconosciuti. La formulazione del problema di ottimizzazione è basata su un approccio di Programmazione Quadratica(QP), che consente l'inclusione di vincoli di movimento: posizione, velocità e accelerazione, insieme a vincoli di evitamento delle collisioni con ostacoli e auto-collisioni.

L'approccio ottiene l'evitamento di collisioni in ambienti complessi e dinamici in cui il manipolatore opera. Le prove in simulazione su un modello del braccio robotico MyCobot280 mostrano l'efficacia della tecnica proposta.

Parole chiave: Controllo predittivo(MPC), Cinematica inversa, Ottimizzazione, Programmazione quadratica, Pianificatore di percorsi.

Contents

List of Figures

List of Tables

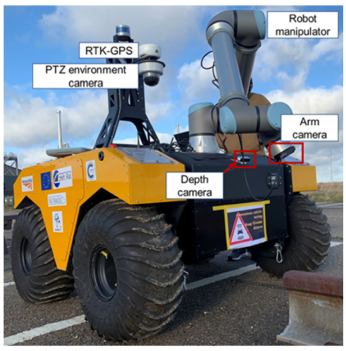
1 | Introduction

1.1. Problem statement

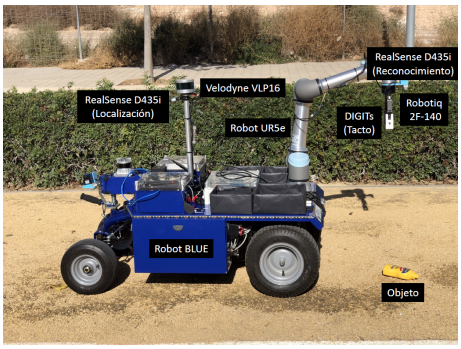
In recent years, the integration of robotic arms into real-world applications has opened a new era of automation and efficiency in different industries. Nevertheless, new challenges have arisen with the ever evolving technologies and applications. Robotic manipulators have been widely included into manufacturing industries, automation and lastly into more complex structures such mobile robotic manipulators. Mobile manipulation systems (MMSs) are robotic systems consisting of one or more robot arms mounted on a mobile base as shown in Figure 1.1. The coupling of the overall system enable the manipulator to navigate and expand its operational workspace, all while preserving its manipulative capabilities. These systems are able to perform extensive series of tasks in various environments which can vary from agriculture [32], inspection [30], logistics [17], pick and place [8] to even construction [27]. In this regard, the secure and safety maneuvering of the whole mobile robotic manipulator within dynamic and unknown environments has been an interest of recent works [43][5][1][7]. Nevertheless, with the increasing number of applications it seems inevitable that more dexterity(than 3-DOF) of the robotic manipulators will be required independently from the mobile part of the robot (e.g. for confined spaces or coexistence with humans). This is the reason why this thesis is focused on a novel solution for obstacle avoidance and inverse kinematics control tailored for a 6-degrees-of-freedom robotic arm operating in unknown environments, without loss of generality of the proposed approach that can be extended to the mobile part.

The core mission is to guide the robotic arm safely toward the successful execution of desired tasks in an environment that is characterized to be unknown, with fixed or dynamic obstacles, relying only on partial environmental information provided by an exteroceptive sensor. The complete architecture of the system embraces a hierarchical control system, comprising a high-level strategy for generating obstacle-free trajectories, a medium-level Model Predictive Controller(MPC) as Inverse Kinematics(IK) and reactive obstacle avoidance, and assumed low-level controllers for joint set-point tracking. More in detail, the overall framework should address the following tasks:

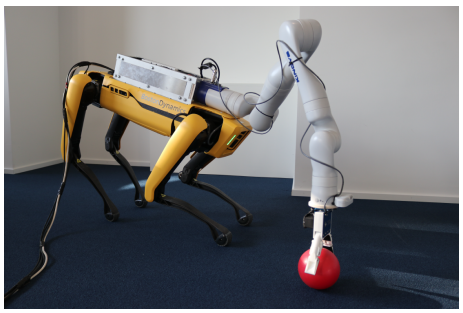
- **Local Path Planning**: The local trajectory planner should guarantee the generation of safe obstacle free paths as partial paths towards the desired goal. This will be tackled by an innovative approach based on the construction of obstacle free convex polytopes which in recent works has been proposed for other applications in autonomous vehicles and unmanned aerial vehicles [10][31].
- **End-Effector positioning in the workspace**: This task should involve an Inverse Kinematics algorithm to map the trajectory generated in task space into joint space while reactively avoiding obstacles of the environment and respecting other enforced constraints. To this end, an Inverse Kinematic-Model Predictive Controller(IK-MPC) is proposed which accounts for the aforementioned tasks. The formulation of the controller as an optimization problem is performed in a Quadratic Programming(QP) approach, allowing the integration of motion constraints covering position, velocity, and acceleration, alongside obstacle collision and self-collision avoidance constraints.



(a) Reference [30]



(b) Reference [8]



(c) Reference [43]

Figure 1.1: Example of Mobile Robotic Manipulators

1.1.1. Robotic manipulator

The fundamental characteristic defining a robot is its mechanical structure. Commonly, robots are categorized based on their structural configuration, namely those with a stationary foundation, known as *robot manipulators*, and those equipped with a mobile base, referred to as *mobile robots*. The mechanical composition of a robot manipulator comprises of a set of rigid bodies (links) interconnected through articulations (joints).

Typically, a manipulator is characterized by an arm that ensures mobility, a wrist that confers dexterity, and an end-effector that performs the task required of the robot. When referring to a robot manipulator there exist some key components, this can be characterized by the numbers of *degrees-of-freedom*(DOF), its *workspace* and the type of joints that it has. The articulation between two consecutive links can be done by means of *prismatic*(translational motion) or *revolute*(rotational motion) joints. Thus, each joint provides a single DOF to the whole kinematic chain, these characterize the dexterity of manipulator to operate a given task. In a three-dimensional (3D) space for positioning and orientating an object, six DOF are required. While the workspace represent the feasible 3D space were the End Effector of the robot can access.

For the purpose of the kinematic modelling and simulation of this work, a MyCobot280 6-DOF Robotic Arm was chosen. This model is part of a family of laboratory robots characterized by six revolute joints which make it able to fully operate(positioning and orientation) in a 3D space. The kinematic parametrization will be performed by means of the Denavit-Hartenberg method explained below, on top of it, an analysis of the *workspace* will be presented in further sections.

The boundaries that are related to the joint position and velocity are specified in the *datasheet* [14] of the robot model. However, it is not the case of the acceleration. Thus, several experiments were performed in the physical robot to obtain the boundary acceleration values by achieving the maximum velocities of the robot with initial zero speed. Table 1.1 shows the results obtained.

Joint Number	$q_{min}[rad]$	$q_{max}[rad]$	$\dot{q}_{min}[\frac{rad}{s}]$	$\dot{q}_{max}[\frac{rad}{s}]$	$\ddot{q}_{min}[\frac{rad}{s^2}]$	$\ddot{q}_{max}[\frac{rad}{s^2}]$
Joint 1	-2.7053	2.7053	-3.1416	3.1416	-15	15
Joint 2	-2.7925	2.7925	-3.1416	3.1416	-15	15
Joint 3	-2.7925	2.7925	-3.1416	3.1416	-15	15
Joint 4	-2.7925	2.7925	-3.1416	3.1416	-15	15
Joint 5	-2.7925	2.7925	-3.1416	3.1416	-15	15
Joint 6	-2.9671	2.9671	-3.1416	3.1416	-15	15

Table 1.1: Constraints limits. MyCobo280 Robotic Arm.

Denavit-Hartenberg Parametrization

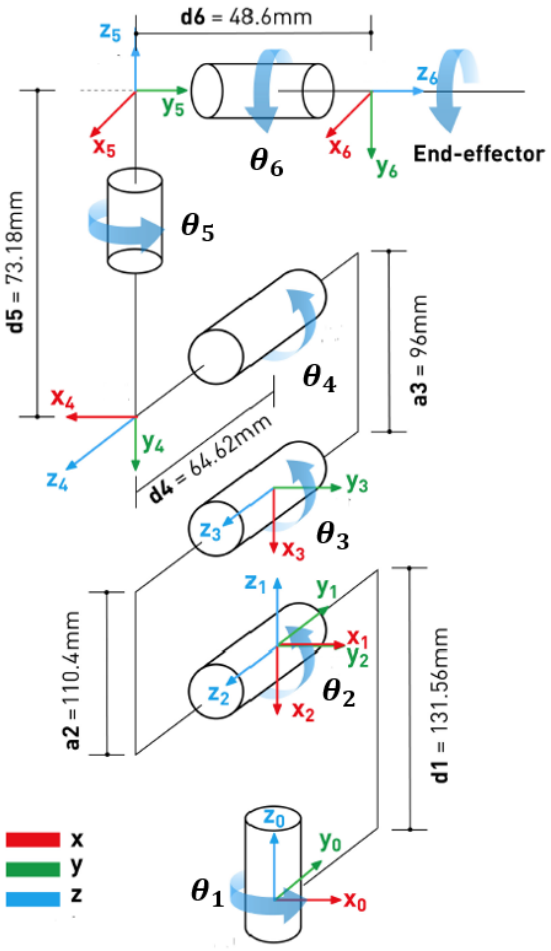
The Denavit-Hartenberg (DH) method is a systematic approach used in robotics to describe the kinematic structure and geometry of a robot manipulator. It provides a standardized framework to define the coordinate systems and parameters that characterize the motion of each joint in a robotic arm. These parameters play a crucial role in the forward and inverse kinematics calculations necessary for robot control and trajectory planning [34]. For the purpose of simulation, a MyCobot280 model Robotic Arm was chosen, the overall structure of the manipulator is shown in Figure 1.2a while the schematic coordinate frames attached to each link according to the Denavit-Hartenberg convention are shown in Figure 1.2b. Without any assumption of a specific tool attached to the end effector,for the sake of simplicity, the end effector position coincides with the reference frame of the last link. Additionally, each frame position is referred with a sub-index starting from 1 to 6, while the index 0 refers to the base frame which is fixed.

Using the standard DH convention, the computed parameters of the referenced model are reported in Table 1.2. The orthogonal distance between the origins of two consecutive frames, called link length, is represented by a, while d identifies the offset between the links. Both are expressed in meters and were obtained from the corresponding data sheet of the robot model [14]. The α parameter represents the link twist and θ is the joint angle of the each corresponding joint. For a detailed explanation of the DH-standard convention used for the parametrization please refer to [34].

Joint Frame	α (rad)	a (m)	d (m)	θ (rad)
1	$\frac{\pi}{2}$	0	0.13156	0
2	0	-0.1104	0	$-\frac{\pi}{2}$
3	0	-0.096	0	0
4	$\frac{\pi}{2}$	0	0.06639	$-\frac{\pi}{2}$
5	$-\frac{\pi}{2}$	0	0.07318	$\frac{\pi}{2}$
6	0	0	0.0436	$-\frac{\pi}{2}$

Table 1.2: Denavit-Hartenberg kinematic parametrization.

(a) MyCobot 280.



(b) DH Frames [14].

Figure 1.2: Robotic Manipulator Model and DH Frames.

1.1.2. LiDAR sensor

Light Detection and Ranging(LiDAR) is a laser based advanced sensing technology that has gained significant prominence in the fields of robotics, autonomous vehicles and industrial automation. This remote sensing method uses laser light to measure distances and generate precise three-dimensional maps of its surroundings. In Figure 1.3, a classic structure of LiDAR sensor is presented.

The principle of operation is based on emitting laser pulses and measuring the time it takes for these pulses to return after bouncing off objects in the environment. By calculating the time-of-flight, LiDAR sensors can determine the distance to each point in their field of view. A rotating scan mirror allows multiple laser pulses to be emitted in quick succession, then a photo-detector measures the reflected light, and the data processing unit calculates the distances and assembles the point cloud data. This real-time data provides a 360-degree visibility that let robots to capture detailed spatial information to create a local map of the environment [4].

In the field of robotics, LiDAR technology plays an important role as a feedback solution which enables the perception and navigation capabilities of autonomous systems. Mounted on robotic arms, mobile robots, or drones, LiDAR sensors allow robots to navigate through complex, unknown or dynamic environments with a high degree of accuracy and safety. The integration of LiDAR technology with optimization strategies offers immense potential for improving various robotic tasks. Thus, robots can exploit the real-time environmental data provided by the LiDAR to optimize their movement, making dynamic decisions based on optimization algorithms that can be applied into several fields such path planning, object recognition, and obstacle avoidance, ensuring that robots perform tasks efficiently and reliably.

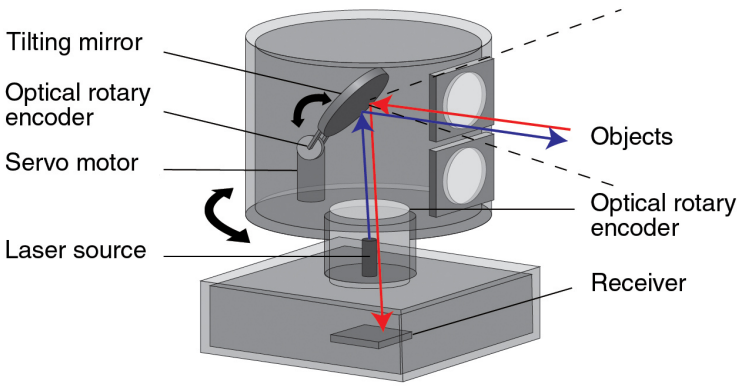


Figure 1.3: Construction diagram of a LiDAR sensor.

The main features for the specification of a 3D LiDAR sensor are:

- Angular horizontal resolution θ_r $[rad]$
- Angular vertical resolution φ_r $[rad]$
- Angular horizontal field of view θ_h $[rad]$
- Angular vertical field of view φ_v $[rad]$
- Maximum distance range r_{Li} $[m]$
- Distance resolution d_{res} $[m]$

1.2. Original Contributions

The main contribution of this thesis is focused on providing a novel solution to the problem of the autonomous operation of a robotic manipulator in unknown environments. To this end, multiple innovative contributions have been proposed, the principal ones are:

- Novel Local Path Planner based on the generation of obstacle free convex polytopes. Additionally, a original Target Shifting strategy is presented as an heuristic approach to the obstacle avoidance task.
- Development of a Model Predictive Controller (MPC) for the Inverse Kinematics(IK) and reactive obstacle avoidance. A Quadratic Programming formulation is proposed including a predictive strategy(for states and the jacobian), direct acceleration constraints (not only joint and velocity constraints), jerk penalization in the cost function, self-collision and obstacle as inequality constraints.
- Together, the proposed framework provides double layer of safety (obstacle free trajectory generation and reactive obstacle avoidance) for obstacle avoidance which is the main contribution of this work.

1.3. Thesis structure

The overall structure of presented Thesis is organized as follows.

In *Chapter 2* the Inverse Kinematics Problem is generally discussed providing ground insights about the solution presented. Additionally, the Kinematic Modelling of the chosen MyCobot280 robot model is presented, delving into the Jacobian computation and a feasible workspace analysis.

Chapter 3 introduces preliminaries representations for the LiDAR simulation and obstacles. Then, presents the proposed method by decoupling the entire structure into Local Path Planning, Inverse Kinematics as an Optimization Problem and the complete iterative Algorithm. For every section a extensive discussion of the related works is done as well as the mathematical formulations.

In *Chapter 4* simulation results are presented. The partial results are divided into the two main algorithms developed for the Local Path Planning and Inverse Kinematics. While the entire system results are presented in depth with an analysis of each complex scenario explaining every iteration made by the overall architecture.

Chapter 5 presents the conclusions derived from this study. It also outlines potential future directions for research and development, highlighting onto possible areas for further exploration and enhancement.

2 | Inverse Kinematics Problem

2.1. Introduction

Typically, the desired trajectory of a robot is planned in Cartesian space, whereas the robot's controller requires a joint space trajectory. Therefore, the classic Inverse Kinematics (IK) problem is defined as the process of determining a set of joint parameters or angles that yield a desired position and orientation for the end-effector of a robotic arm in the cartesian space. The significance of IK lies in its application in trajectory planning and control, where it enables robots to perform precise and controlled movements of a desired motion. Between the traditional methods for solving the inverse kinematics there exist two main types: completely analytical (closed form solution) and numerical optimization techniques [34].

In general, consider a n-degree-of-freedom manipulator and denoting q, $\dot{q}$ and $\ddot{q} \in \mathbb{R}^n$ its joint position, velocity and acceleration vectors respectively. The robot is required to perform a main task $x_d(t) \in \mathbb{R}^m$ in the Cartesian space, such that $f_0(q(t)) = x_d(t) \forall t$ where f_0 is the forward kinematic function related to the manipulator. This task can be considered at differential level exploiting the relationship between the joint velocity space and the operational velocity space. This relation is expressed as follows:

$$\underbrace{\begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix}}_{\dot{x}_d} = \underbrace{\begin{bmatrix} J_P \\ J_O \end{bmatrix}}_{J} \dot{q} \tag{2.1}$$

Where J_P is the $(3 \times n)$ matrix relating the contribution of the joint velocities $\dot{q}$ to the end-effector linear velocity $\dot{p}_e$, while J_P is the $(3 \times n)$ matrix relating the contribution of the joint velocities $\dot{q}$ to the end-effector angular velocity ω_e. Thus, the $(6 \times n)$ matrix J is the manipulator geometric Jacobian. By considering (2.1), the joint velocities can be obtained via simple inversion of the Jacobian matrix.

$$\dot{q} = J^{-1}(q)\dot{x}_d \tag{2.2}$$

If the initial joint configuration of the manipulator q(0) is known, then the joint positions can be computed by integrating velocities over time. Although this integration can be performed in discrete time by numerical techniques, it can also lead to divergence of the solution known as *drift phenomena* and as a consequence the end-effector pose corresponding to the joint positions differs from the desired one. This problem is usually adressed by including the *operational space error* which accounts for the difference between the end-effector actual position and the desired one [34]:

$$e = x_d - x_e \tag{2.3}$$

Thus, including the operational space error as state feedback, the inverse kinematic problem can be rewritten as:

$$\dot{q} = J_A^{-1}(q)(\dot{x}_d + Ke) \tag{2.4}$$

Where K is a positive definite diagonal matrix weighting the error and J_A is the Analytical Jacobian. The latter uses a specific minimal representation of the end effector orientation (eg. ZYZ Euler Angles) for the rotational velocity $\dot{\varphi}_e$ instead of angular velocities ω_e. The relation between both can be expressed as:

$$\omega_e = T(\varphi_e)\dot{\varphi}_e \tag{2.5}$$

Matrix $T(\varphi_e)$ represents the contributions of each rotational velocity to the components of angular velocities about the axes of the reference frame. Once this matrix is obtained the analytical Jacobian and geometric Jacobian can be related as:

$$J = \underbrace{\begin{bmatrix} I & 0 \\ 0 & T(\varphi_e) \end{bmatrix}}_{T_A(\varphi_e)} J_A \tag{2.6}$$

It can be proven that, if J_A is square and non-singular, the choice of (2.4) leads to a convergence of the error towards zero. In particular, this solution is known as the Closed-Loop Inverse Kinematic(CLIK) method and the corresponding block scheme is shown in Figure 2.1. The general overview of this closed loop solution will help to understand and develop the next steps related to the Inverse Kinematics algorithm proposed in this work.

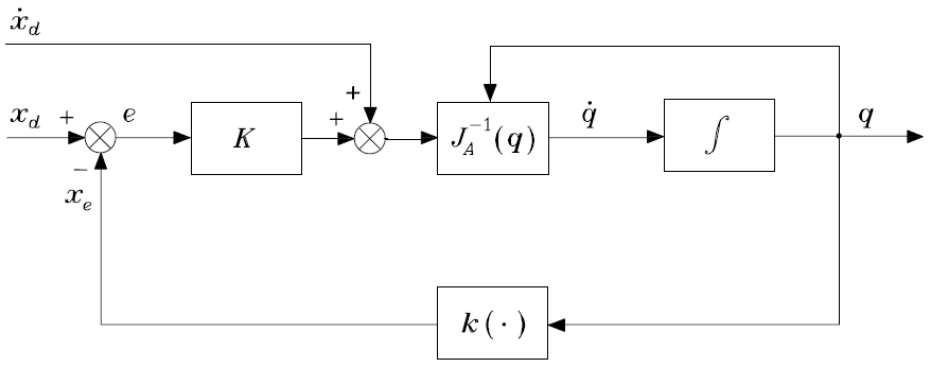


Figure 2.1: Inverse kinematics algorithm with Jacobian Inverse [34]

Although this solution works well for unconstrained Inverse Kinematics, if joint constraints need to be imposed, they cannot be easily ensured in the form of the referred CLIK. Even though there exist methods that account for the joint position limits, the constraints for joint velocities and especially accelerations are not straightforward to include [40]. This particular reason led the present research to adapt novel numerical constrained formulations discussed in the following sections.

2.2. Kinematic Modelling

In order to properly formulate the Inverse Kinematics problem it is first necessary to analyze the direct kinematics equations that governs the robotic system. As mentioned, this can be achieved through the well known Denavit-Hartenberg(DH) method which is also adopted in this work. This previous analysis will ease the computation of the Geometrical and Analytical Jacobian that are needed for the Inverse Kinematics formulation.

2.2.1. Preliminaries

Direct Kinematics

The direct kinematics aims to relate the pose of the end-effector as a function of the joint variables by means of homogeneous transformation matrices. These matrices expresses the coordinate transformation between two frames in a compact form and can be defined in terms of the DH parameters as [34]:

$$A_i^{i-1}(q_i) = A_{i'}^{i-1} A_i^{i'} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Where indexes relates the i-th and $(i-1)$-th frame, and the function only depends on the actual revolute joint θ_i and the DH parameters. Furthermore, the direct kinematic function T_e^b can be calculated by the product of the intermediate homogeneous transformation matrices, defined as follows:

$$T_e^b(q) = A_1^0 A_2^1 A_3^2 ... A_6^5 = \begin{bmatrix} n_e^b(q) & s_e^b(q) & a_e^b(q) & p_e^b(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

Where n_e^b , s_e^b and a_e^b are the unit vectors of the frame attached to the end effector, moreover, the rotation matrix from base to end effector is $R_e^b(q) = [n_e^b(q), s_e^b(q), a_e^b(q)]$. Thus, the pose(position and orientation) of the end effector can be attained by means of the direct kinematic function. This previous analysis will ease the computation of the Geometrical and Analytical Jacobian that are needed for the Inverse Kinematics formulation.

Euler Angles Representation

Additionally, as stated before, the use of a more comprehensive representation of the orientation angles will be employed. To this end, the *ZYZ Euler angles* were chosen to represent the rotational matrices of end effector. Given the rotational matrix $R_e^b(q)$ corresponding to the orientation of the end effector:

$$R_e^b(q) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.9}$$

Then the representation of the set φ_e of Euler ZYZ angles (ϕ_E,θ_E,ψ_E) is performed as follows:

$$\varphi_e = \begin{bmatrix} \phi_E \\ \theta_E \\ \psi_E \end{bmatrix} = \begin{bmatrix} atan2(r_{23}, r_{13}) \\ atan2\big(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}\big) \\ atan2(r_{32}, -r_{31}) \end{bmatrix} \tag{2.10}$$

Therefore, the orientation transformation matrix can be derived as:

$$T(\varphi_e) = \begin{bmatrix} 0 & -\sin\phi_e & \cos\phi_e \sin\theta_e \\ 0 & \cos\phi_e & \sin\phi_e \sin\theta_e \\ 1 & 0 & \cos\theta_e \end{bmatrix} \tag{2.11}$$

2.2.2. Jacobian Computation

As the Denavit–Hartenberg convention was used for the description of the kinematic chain, this resource is exploited in this section for the computation of the Jacobian in literal form. To this end, the transformation matrices of each joint were computed forward and used to get the intermediate positions ($p_1, p_2..p_6$) and intermediate orientations of the frames ($z_1, z_2..z_6$). Till this point an homogeneous transformation matrix from base to end effector could be extracted in analytical form. Finally, both of the last mentioned expressions where used to compute the final analytical expression of the Jacobian. A detailed explanation is represented in the developed Algorithm (2.1) where the index "i" refers to the each joint variable and the set of joint angles defined as $q = [\theta_1, \theta_2, ..., \theta_6]$. The parameters vectors are defined as: $\bar{a} = [a_1, ..., a_n]$, $\bar{d} = [d_1, ..., d_n]$ and $\bar{\alpha} = [\alpha_1, ..., \alpha_n]$

Algorithm 2.1 Jacobian Computation

Input : DH parameters $\bar{a}, \bar{d}, \bar{\alpha}, q$

Output: $T_e^b(q)$, $J(q)$

1: $n \leftarrow 6$

2: **while** $i \leq n$ **do**

3: Computation of the homogeneous transformation matrices as Equation (2.7):

$$A_i^{i-1}(q_i) \leftarrow \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4: Extraction of the frame's rotation matrices and position vectors:

$$A_i^0(q_i) = A_1^0 A_2^1 ... A_i^{i-1} = \begin{bmatrix} R_i^0 & p_i^0 \\ 0 & 1 \end{bmatrix}$$

5: Computation of each frame orientation:

$z_i = R_i^0 z_0$

6: **end while**

7: Calculation of the each joint Jacobian as follows:

$$J_i = \begin{bmatrix} z_i \times (p - p_{i-1}) \\ z_{i-1} \end{bmatrix}$$

8: Construction of the complete Jacobian:

$J(q) = [J_0, \dots, J_5] \in \mathbb{R}^{6\times 6}$

9: Construction of the Homogeneous Transformation Matrix from base frame to end effector frame:

$$T_e^b(q) = A_1^0 A_2^1 A_3^2 ... A_6^5 = \begin{bmatrix} R_e^b & p_e^b \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

10: Computation of the ZYZ Euler angles (Equation (2.10)) and transformation matrix(Equation (2.11)):

$T(\varphi_e) \leftarrow T(\phi_E, \theta_E, \psi_E)$

11: Finally, the analytical jacobian can be expressed as:

$$J_A(q) = \begin{bmatrix} I & 0 \\ 0 & T(\varphi_e) \end{bmatrix}^{-1} J(q)$$

The computation of the homogeneous transformation matrix from base to end effector $T_e^b(q)$ and the Analytical Jacobian $J_A(q)$ both in analytical form will be recalled as functions to be evaluated in further algorithms. Moreover, the function to provide the end effector pose in terms of ZYZ Euler angles will be defined as $f_0(q)$.

2.2.3. Analysis of the Feasible Workspace

As mentioned, the feasible workspace of a robot is an intrinsic characteristic of it. The analysis of the workspace is an important aspect of understanding its capabilities and limitations. This exploration focus into determining the spatial coordinate domain within which the kinematic chain can maneuver and operate effectively. Thereby, taking into account the Homogeneous Transformation matrix from base to end effector $T_e^b(q)$ gathered from the Algorithm 2.1, an extension iterative Algorithm 2.2 was developed. The main purpose of the latter is to retrieve a set of points of the feasible workspace $P_{fe} = [p_1, ..., p_{N_{fe}}]$ based on the configuration space and compute the set $\mathcal{W}$ which is the convex hull representation of the feasible workspace. The parameters θ_{min} and θ_{max} represent the joint minimum and maximum limits of the robot,and θ_{step} the minimum step to loop the range of each joint through all the configuration space.

Algorithm 2.2 Computation of the feasible workspace

Input : θ_{step} , $T_e^b(q)$,θ_{min} ,θ_{max}

Output: $\mathcal{W}$

1: $x_e(q) \leftarrow T_e^b(q)_{(4,1:3)}$
2: **for all** $\theta_i = \theta_{min} : \theta_{step} : \theta_{max}$ **do**
3: Compute feasible points matrix:
$P_{fe}^i = x_e(q_i)$
4: **end for**
5: $\mathcal{W} = chull(P_{fe})$

A result of the presented algorithm can be observed in Figure 2.2, where the green dots represents the point cloud of feasible workspace P_{fe} and the light-blue polyhedron represents the convex hull $\mathcal{W}$. Notice that this representation will be remain invariable since it is referred to the world coordinate frame of the robot.

It should be also highlighted that a further development of this algorithm could analyze the singularities related to the kinematic chain based on the Singular Values Decomposition(SVD) of the determinant of the Jacobian which was also computed by Algorithm 2.1. Nevertheless, this is out of the scope of this work.

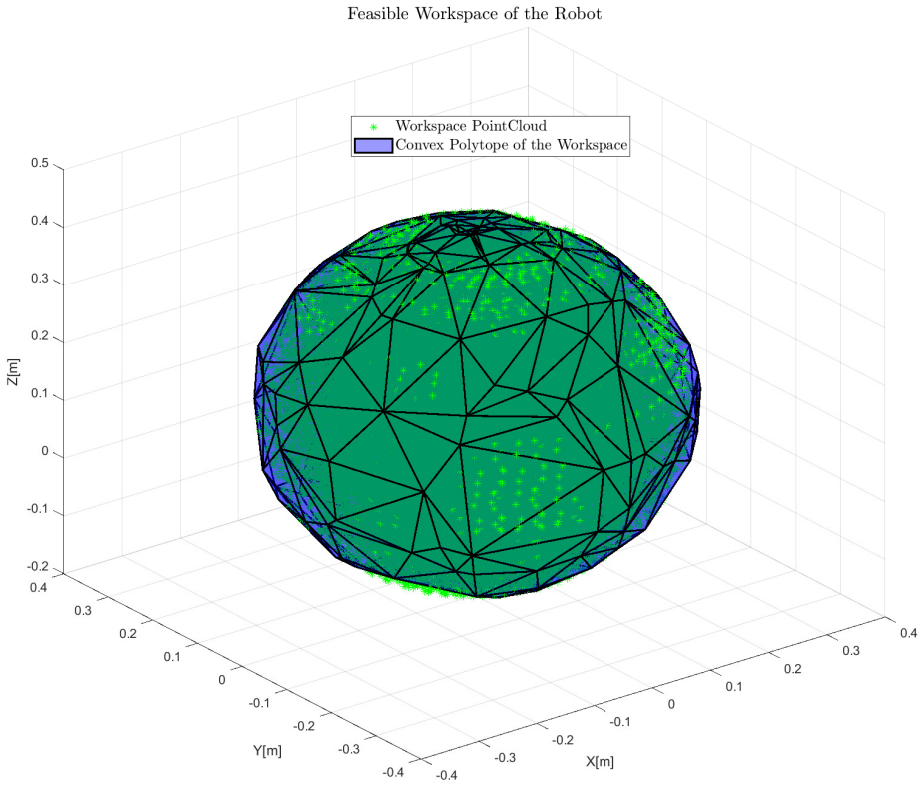


Figure 2.2: Feasible Workspace

3 | Proposed Method

3.1. Introduction

As presented in the last section, the Inverse Kinematics problem is challenging by itself and it can be addressed by different methods; nevertheless, a precondition is required which is the reference trajectory in coordinate space to follow towards a target position. This trajectory is generated by automatic path planners which typically take into account a preliminary description of the environment. On the contrary, given that the scenario of the manipulator does not have any a-prior description of the environment, the automatic path planner will be only account for local information from the exteroceptive sensor therefore this will be referred as a Local Path Planner. Accordingly, the proposed approach can be decomposed in two main structures detailed in the following sections. In detail, the overall control architecture of the proposed approach is shown in Figure 3.1.

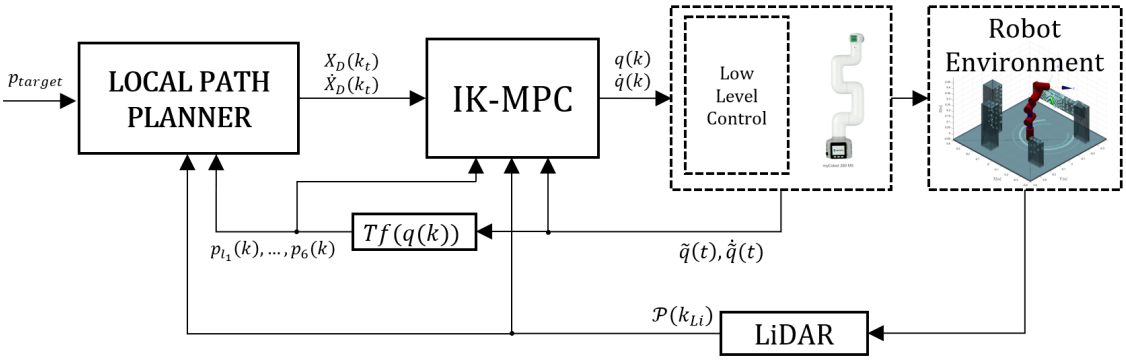


Figure 3.1: Block diagram scheme of the architecture

As it can be observed, the Local Path Planner anticipates a generated trajectory (position X_D and velocity $\dot{X}_D$) to the Inverse Kinematics block, and the latter provides the joint references$(q(k), \dot{q}(k))$ to the low level controllers to perform the required motion. The feedback of the actual position of the robot is computed by its forward kinematic function $Tf(q(k))$ while the point-cloud $\mathcal{P}$ of the position of the obstacles in the environment is provided by the exteroceptive LiDAR sensor. Each of the mentioned variables will be

formally defined in the subsequent sections.

Since for the purpose of this work the motion of the robot will be performed in a simulated environment, some assumptions were taken to mimic as close as possible a real-world scenario:

- The LiDAR sensor is mounted on the top of the robot, that is the end effector position. The sensing from the LiDAR is simulated by an algorithm, that is based on the mounted position.
- The environment will be characterized by three-dimensional obstacles, convex or non-convex, with variable cross-section. Where each obstacle can be described as a compact set.
- The robotic manipulator task is to reach a general target position p_{target} inside the environment which is external to every obstacle.
- The low level controllers are assumed to have a sufficient large gain margin.

The methodology adopted focuses in the development of modular simpler algorithms which together conforms the complete algorithm of the proposed solution. Therefore, this section is meant to present the formulation and algorithms belonging to the Local Path Planning stage and the Inverse Kinematics, both will serve as ground for the understanding of the complete structure presented in the last section.

3.2. Preliminaries

Since the presented control architecture is meant to be evaluated in a simulation environment, it is essential to define additional components of the environment apart from the model of the robot. It is necessary to include a model of the obstacles inside the environment coupled with an algorithm that can emulate the LiDAR sensing. Therefore, this section provides insights in the definition of the obstacles inside the environment and LiDAR measurements which will be used in the formalization of the problem on hands.

3.2.1. Obstacles Representation

Obstacles are represented as rectangular cuboids, as depicted in Figure (3.2). This particular shape is convex which allows various straightforward representations. Specifically, it is highly advantageous to represent obstacles using the six inequalities associated with the planes that describe their faces [9]. This method simplifies the process of determining whether a given point in space is within an obstacle. Although this approach focuses on

simple and convex shapes, it does not restrict the ability to represent more complex obstacles. Complex shapes can be achieved by combining multiple simple objects, including non-convex ones.

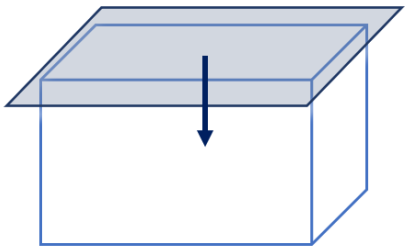


Figure 3.2: Obstacle representation with one plane associated to the upper face

Equation (3.1) corresponds to a three-dimensional linear constraint associated with an obstacle's face. If a point satisfies all six constraints related to the faces of a particular obstacle, it is considered to be inside the obstacle.

$$a_i x + b_i y + c_i z + d_i \geq 0 \ , \ i = 1, ..., 6 \tag{3.1}$$

Each obstacle can be stored as a $6x4$ matrix containing the parameters of the inequalities as shown in Equation (3.2) where j represents the index. While the representation assumes obstacles to be rectangular shape cuboids for simplicity, it is equally suitable for representing generic cuboids and convex polyhedra by increasing the number of inequalities as needed.

$$\mathcal{O}_j = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \\ a_5 & b_5 & c_5 & d_5 \\ a_6 & b_6 & c_6 & d_6 \end{bmatrix} \in \mathbb{R}^3 \tag{3.2}$$

Therefore, the overall set of obstacles can be defined as :

$$\mathcal{O} = \bigcup_{j=1}^{N_{obs}} \mathcal{O}_j \tag{3.3}$$

3.2.2. LiDAR simulation

For the purpose of this study, A simulation algorithm is employed to emulate LiDAR sensing, taking into account that is mounted in the robot end effector position. This assumption is not contradictory of what is a common practice nowadays in the industry. This algorithm generates data information that closely resembles what would be obtained from a physical sensor. The accuracy of the simulations and the execution time significantly relies on this sensor emulation. Thus, an essential requirement for this model is its computational efficiency since LiDAR sensors typically provide data at a high refresh rate, often exceeding 1 kHz. Therefore, the model must be computationally cheap to ensure that simulations can be conducted efficiently without compromising accuracy. To this end, an extension of the Algorithm proposed in [9] is formulated in Algorithm (3.1) for a 3D system environment. The advantage of this algorithm is that exploits the formulation of the obstacles boundaries as inequalities in order to optimize the computations needed to check intersections enabling it to reduce the computational effort and time.

Therefore, the model of the LiDAR sensor must take as input the position of the end effector of the robot, $p_{EE}(k_{Li})$ together with the set of obstacles $\mathcal{O} = [\mathcal{O}_1, ..., \mathcal{O}_{N_{obs}}]$ and the parameters vector $param_{Li} = [\varphi_v, r_{Li}, d_{res}]$ at time k_{Li}. Given the resolution of the vertical(elevation) and horizontal(azimuth) angles, the complete dimensional vectors of the angles to produce the points cloud of the sensor can be respectively defined as $\bar{\varphi} = [-\varphi_v, -\varphi_v + \varphi_{acc}, -\varphi_v + 2\varphi_{acc}, ..., +\varphi_v] \in \mathbb{R}^{N_v}$ and $\bar{\theta} = [0, 2\theta_{acc}, 3\theta_{acc}, ..., 2\pi] \in \mathbb{R}^{N_v}$. Then the algorithm must return as output a set of sensor readings $\mathcal{P}(k_{Li}) \in \mathbb{R}^{3\times N_v N_h}$ and a vector of distances $d_{Li}(k_{Li}) \in \mathbb{R}^{N_v N_h}$ with respect to the center of the sensor(in this case coincides with the end effector position) in the same form that a real sensor would provide.

The chosen values for the LiDAR model parametrization are reported in Table (3.1). As it can be noticed, in order to avoid excessive computational time the resolution of the measurements was increased as well as the distance resolution with respect to standard values(less than 1 deg of precision) that can be encountered in commercial LiDAR products. Nevertheless, the remaining parameters related to the field of view were chosen according to standard specifications.

Parameter	Value
Angular horizontal resolution (θ_r) [deg]	5
Angular vertical resolution (φ_r) [deg]	5
Angular horizontal field of view (θ_h) [deg]	360
Angular vertical field of view (φ_v) [deg]	75
Maximum distance range (r_{Li}) [m]	0.5
Distance resolution (d_{res}) [m]	0.01

Table 3.1: LiDAR Parameters

Algorithm 3.1 LiDAR Simulation Algorithm

Input : $\mathcal{O}$,$param_{Li}$, $p_{EE}(k_{Li})$
Output: $\mathcal{P}(k_{Li})$, $d_{Li}(k_{Li})$

$ind \leftarrow 1$
for $\varphi = -\varphi_v : \varphi_{acc} : \varphi_v$ **do**
  for $\theta = 0 : \theta_{acc} : 2\pi$ **do**
    for $\bar{d_c} = d_{res} : d_{res} : r_{Li}$ **do**
      $p_c \leftarrow p_{EE}(k_{Li}) + [d_c \cos(\varphi)\cos(\theta), d_c \cos(\varphi)\sin(\theta), d\sin(\varphi)]^T$
      for $i \leq N_{obs}$ **do**
        $A_c \leftarrow -\mathcal{O}_{i,(:,1:3)}$
        $b_c \leftarrow \mathcal{O}_{i,(:,4)}$
        if $A_c p_c \leq bc$ **then**
          $d_{Li} \leftarrow min(r_{Li}, d_c)$
          if $d_c \leq r_{Li}$ **then**
            $\mathcal{P}_{ind}(k_{Li}) \leftarrow p_c$
          else
            $\mathcal{P}_{ind}(k_{Li}) \leftarrow p_c - [r_{Li}\cos(\varphi)\cos(\theta), r_{Li}\cos(\varphi)\sin(\theta), r_{Li}\sin(\varphi)]^T$
          end if
        end if
      end for
    end for
    $ind \leftarrow ind + 1$;
  end for
end for

As a graphical representation, a complete overview of the LiDAR sensor measurements $\mathcal{P}(k_{Li})$ (light-blue dots) and four obstacles representations are reported in Figure 3.3. As mentioned before, the LiDAR sensor is assumed to be mounted in the end effector position

of the robot, thereby, the point cloud obtained will always depend on this position.

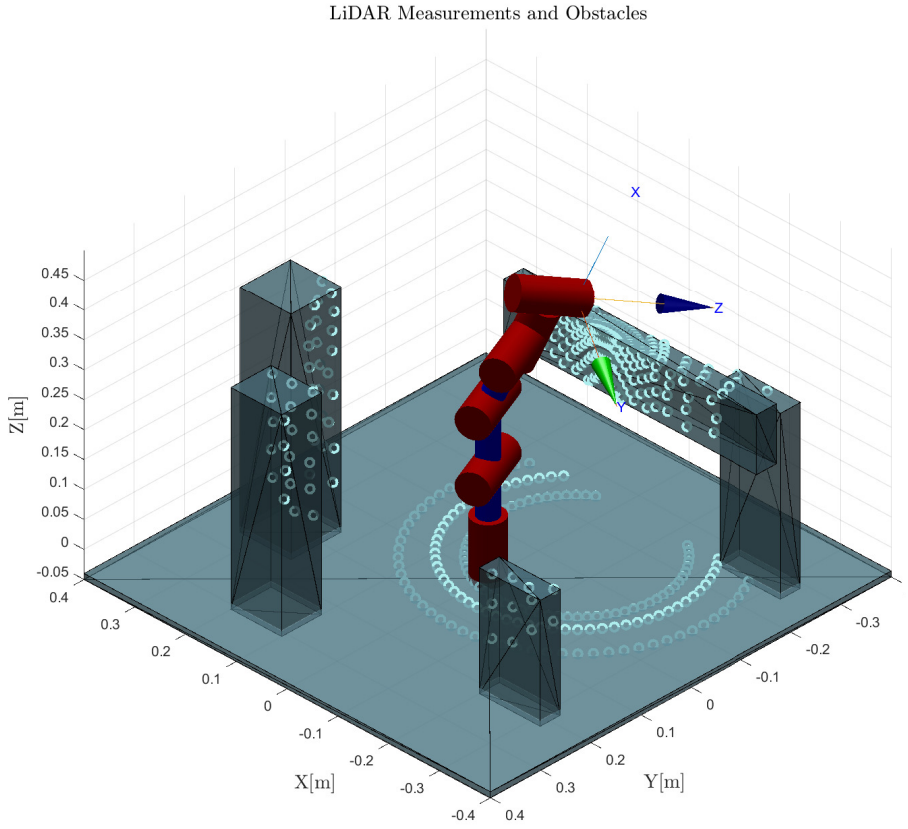


Figure 3.3: LiDAR measurements (light-blue dots) and Obstacles.

3.3. Local Path Planning

This section is divided in three main parts, first a state-of-the-art analysis is presented to ground the solution proposed. Second, the algorithms for the Convex Approximation of the Free Space and Trajectory generation are formulated and deployed.

In this methodology, the trajectory generation is generated every time the robot requires it, in other words, when the robot already performed its last trajectory fed. Therefore, the creation of the obstacle free polytope and the trajectory generation will be performed after a varying time period τ_t which can be defined to happen every k_t step. Specifically, between the time steps of $k_t(t)$ and $k_t(t + \tau_t)$ the trajectory is generated and fed into the Inverse Kinematics algorithm.

3.3.1. Related works

The problem of autonomous navigation of robots has been addressed quite extensively in the literature [3]. In this domain, trajectory planning algorithms are typically approached through two main strategies: Global Planning and Local Planning.

Global planning algorithms require a prior description of the map of the environment, or at least the area between the current and target robot positions. Therefore, are able to generate obstacle free trajectories given the current configuration of the obstacles in the space. Algorithms based on probabilistic techniques such as PRM(*Probabilistic Roadmap*) or RRT (*Rapidly-exploring Random Tree*) are widely used in the industry [34].

However, in new robotics applications with dynamical environments the robot must be able to plan its motion online and without an entire knowledge of its environment. This entails utilizing partial information about the workspace acquired during motion, relying on sensor measurements. Therefore, local methods rely solely on local environmental information, suppressing a previous knowledge of a complete map. As a result, they are well-suited for online and reactive trajectory planning [34]. In this domain, path planning via artificial potential fields is a well know method which aims to drive the robot towards a goal configuration while avoiding obstacles. In detail, each obstacle produces an artificial potential function positioned in the center of the obstacle and the gradient of such function can be interpreted as a repulsive force pushing the robot away from the obstacles. Nevertheless, this method suffers from local minima problem when the pulling and pushing components are equal and opposed, so that the virtual force acting on the robot becomes zero and the robot stops moving. Consequently, the motion planning methods via artificial potentials are not complete in general, because it may happen that the goal configuration is not reached even though a solution exists [35]. Several efforts had been conducted to avoid the local minima problem; nevertheless, most of them does not rely on an offline planned nominal path to be possibly modified/adapted. Furthermore, only a few of them are actually able to include kinematic constraints, such as velocity limits or limited joint ranges, as well as any other kind of constraints to be accounted during task execution [6].

In this regard, a new methodology based on the generation of Convex Regions of Obstacle-Free Space has been acquiring popularity during the last years in several fields such autonomous vehicles [10], unmanned aerial vehicle [31], humanoid robots [26] and specifically in the field of robotic manipulators [39][29][12]. This method allows the generation of regions that are convex and collision free that surpass efficiently the computational burden of collision checking from standard motion planning techniques while driving the agents

through guaranteed safe trajectories. In detail, the trajectory planning under an approximated obstacle free region can be approached in Configuration space or Cartesian space, and can be generated based on convex optimization techniques(were constraints can be enforced) or efficient algorithms. For example in [12] the generation of convex polytopes is based Cartesian space while in [39][29] in configuration spaces, both approaches by means of convex optimization techniques. Nevertheless, the computational time presented for a 7DOF robot in a cluttered environment is in order of the 10^3 seconds which can compromise an online iterative algorithm. On the other hand, new computationally efficient methods based on Cartesian space were developed in [10][31] showcasing its potential for an online real implementation which is the aim of this work.

On top of the latter mentioned approaches, this work proposes a structural division. From one side, the generation of obstacle-free regions in Cartesian space for the purpose of trajectory generation by including the latter computational efficient algorithms in the current field of robotic manipulators. While for the inclusion of kinematic constraints and reactive obstacle avoidance, a trajectory following inverse kinematics its proposed as an optimization problem. The proposed partition guarantees a double layer of safety (obstacle free trajectory generation and reactive obstacle avoidance) for obstacle avoidance which allows a safe motion of the manipulator towards a target position in a obstacle characterized environment .

3.3.2. Convex Approximation of the Free Space

As mentioned before, the manipulator has an exteroceptive sensor used to gather information of its environment. The position and configuration of this LiDAR sensor mounted in the head of the robot recovers all the measurements of the closest obstacles to the robot within the range of measurement. Lets denote $\mathcal{P}(k_t) = [\rho_{Li,k_t}(0), ..., \rho_{Li,k_t}(N_{Li} - 1)] \in \mathbb{R}^{3\times N_{Li}}$ the N_{Li} readings of the LiDAR sensor at time k_t, p_{EE} as the position of the end effector of the manipulator and N_{obs} the number of obstacles in the environment. Additionally, considering user selected quantities $\beta_a > \theta_r$ and $\beta_e > \varphi_r$ i.e. azimuth and elevation angular resolution intervals defining a number n_v of equally spaced candidate vertices on the unit sphere centered at the end effector position. Where for an ordered set of points $S^v = \{S_1^v, ..., S_{n_v}^v\} \in \mathbb{R}^3$ (in this case vertices), the notation $chull(S^v)$ denotes their convex hull. As proposed in [31], the Algorithm 3.2 for the construction of convex sets of obstacle free regions was developed with small modifications.

First, a regular polyhedron with n_v vertices circumscribed in a sphere of radius d_{min} is built. Then, the vertex corresponding to the iteration is radially translated with respect

to the center by the distance d_{step}, and the convex hull is updated. Thus, if the resultant hull does not contains any sensor measurement $\mathcal{P}(k_t)$ and the computed vertices are inside the detection range, the polyhedron $\mathcal{S}^(v)$ is updated and the operation repeats for the next vertex. On the contrary, if the condition does not meet, the last expansion is removed and the position of the vertex is blocked. The cycle concludes when all auxiliary variables γ_i are set to true (as indicated in line 7). This condition guarantees that either all vertices have been blocked or they reached the maximum allowable distance of r_{Li}. When the full process is completed, the algorithm returns the convex set $\mathcal{S}^v$ of the resultant polyhedron. Although the generated polytope $\mathcal{S}^v$ represents an obstacle free region, it does not guaranteed that this region is inside the feasible space of the robot. Moreover, the generation of a obstacle free trajectory towards a target inside the polytope could lead to an unfeasible task for the robot. Therefore, according to the analysis of the feasible workspace developed in Section 2.2.3, this polytope must also lay inside the boundaries of the feasible space of the robot. To this end, an intersection of the convex sets must be carried out. Since the intersection of two convex sets is also convex, the algorithm returns the convex hull $\mathcal{S}(k_t)$ of the intersection of the aforementioned sets (line 16).

Algorithm 3.2 Convex under-approximation of the free space

Input : $\mathcal{P}(k_t)$, $\mathcal{W}$, $p_{EE}(k_t)$, d_{step}, r_{Li}
Output : $\mathcal{S}(k_t)$
Procedure : $polytope\_gen(\mathcal{P}(k_t), \mathcal{W}, p_{EE}(k_t), d_{step}, r_{Li})$

1: Finding the minimum distance to the point cloud:
$d_{min}(k_t) \leftarrow min_{i=0,\dots,N_{Li}-1} |\rho_{Li,k_t}(i)|$
2: Initialization of variable γ as a vector of zeros of dimension n_v:
$\gamma \leftarrow 0_{1\times n_v}$
3: **for** $i = 0 : n_v - 1$ **do**
4: $\mathcal{S}_i^v \leftarrow p_{EE}(k_t) + d_{min}(k_t)[\cos(i\beta_e)\cos(i\beta_a), \cos(i\beta_e)\sin(i\beta_a), \sin(i\beta_e)]^T$;
5: **end for**
6: **for** $i = 0 : n_v - 1$ **do**
7: **while** $\gamma_i \neq 1$ **do**
8: $\bar{\mathcal{S}}_i^v \leftarrow \mathcal{S}_i^v + d_{step}[\cos(i\beta_e)\cos(i\beta_a), \cos(i\beta_e)\sin(i\beta_a), \sin(i\beta_e)]^T$;
9: **if** $\mathcal{P}(k_t) \notin chull(\bar{\mathcal{S}}^v) \wedge (\|\bar{\mathcal{S}}_i^v - p_{EE}(k_t)\|_2^2 < r_{Li})$ **then**
10: $\mathcal{S}^v \leftarrow \bar{\mathcal{S}}^v$
11: **else**
12: $\gamma_i \leftarrow 1$
13: **end if**
14: **end while**
15: **end for**
16: $\mathcal{S}(k_t) = chull(\mathcal{S}^v \cap \mathcal{W})$

A graphic representation of the outcome of the Algorithm is shown in Figure 3.4, where the light-blue dots represents the sensor measurements, the light-red region represents the

obstacle free region $\mathcal{S}^v$ and the green region represent the intersection with the workspace $\mathcal{S}(k_t)$.

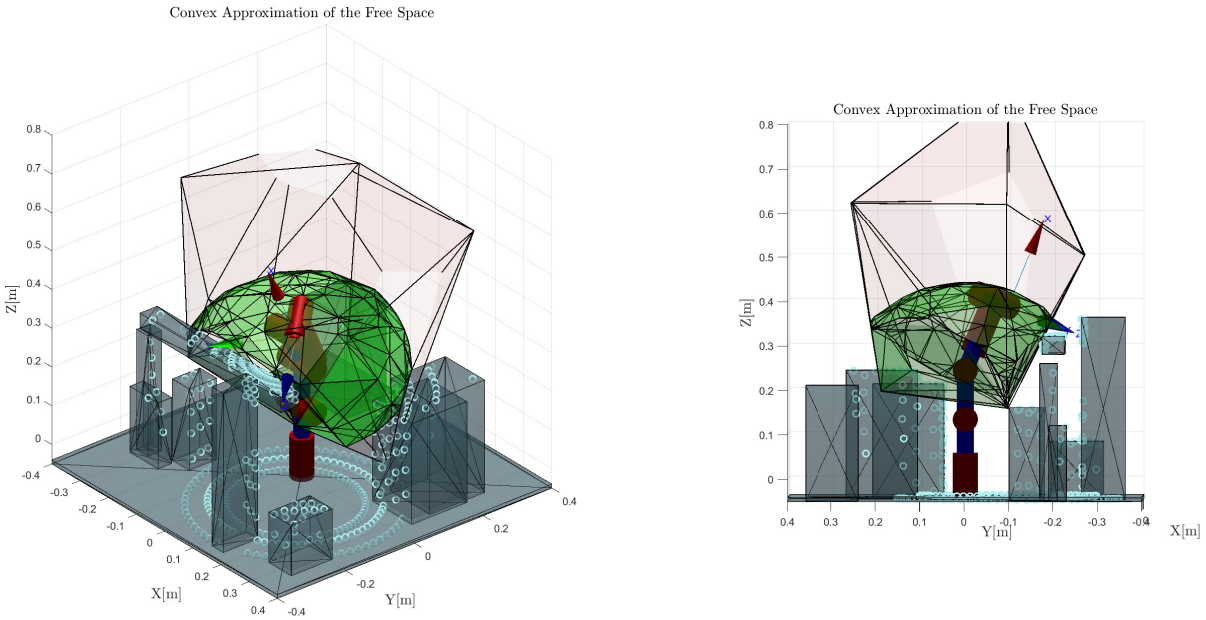


Figure 3.4: Example of construction of the convex approximation of the free region. Left: Isometric View. Right: Side view.

3.3.3. Obstacle Free Trajectory Generation

This section focuses on the proposed strategy for the local path planning of the 6-degree-of-freedom robotic manipulator operating in an unknown environment. The key challenge is to avoid obstacles in the environment that might impede the robot's movement toward its target location. The method involves a multi-step strategy, starting with a Temporary Target Shifting Strategy to the Trajectory Generation of the obstacle-free reference path to be fed to the Inverse Kinematics block.

The Temporary Target Shifting Strategy tackles the problem of finding an obstacle free path when an obstacle obstructs the robot's direct path to its target position. The proposed strategy selects a temporary target from the sensor readings within a specific threshold range in order to avoid trajectories towards an imminent obstacle. The choice of this temporary target involves vector-based strategies.

Following this strategy, the Trajectory Generation phase is deployed. Based on the convex under-approximation of free space, the strategy generates trajectories for the robot's movement from its current location to a temporary target. This process involves defining minimum jerk polynomial trajectories to ensure smooth, controlled, and vibration-free

motion. The orientation of the robot's end effector is also considered, aligning it towards the target position.

Finally, the complete Obstacle Free Trajectory Generation algorithm, detailed in Algorithm 3.5, synthesizes and combine these steps to generate the robot's path, handling obstacle avoidance and orienting the robot towards its target location. The trajectory is developed to guarantee an obstacle-free path characterized by smooth polynomial motions.

Temporary Target Shifting Strategy

A frequently encountered issue in optimization-driven autonomous path planning without a preliminary knowledge of the environment occurs when the agent faces an obstacle situated between the current location and the intended target. As proposed in [31], a target shifting strategy must take place in order to avoid a locally optimal position where the agent could stop moving because any other movement will just increase the distance from the target position.

Contrary to what is proposed in [31] for unmanned aerial vehicles (UAVs) where the temporary target is the obstacle free reading that is closest to the target, in this case, the agent is a 6-DOF robotic manipulator with unknown obstacles in its workspace which makes the problem more sensible to the choice of a temporary target. The sensibility of the chosen target is explained by the fact that the robotic arm has a fixed base frame position with a connected kinematic chain(not only a compact body) that allows different configurations, and that the control architecture includes a reactive obstacle avoidance function. Together, the overall implementation can lead to unstable configurations or repetitive movements of the robot trying to reach the aforementioned target position. Thereby, the target shifting strategy adopted differs mainly in the choice of temporary target.

Considering the sensor readings as $\mathcal{P}(k_t)$ at time k_t, and a specific threshold $\bar{\varepsilon}$, the set of candidate indexes can be defined as:

$$\mathcal{I}(k_t) = \left\{ \bar{i} : \bar{i} = \rho_{Li,k_t}(i) > \bar{\varepsilon} \ , \forall i = 0, .., N_{Li} - 1 \right\}$$

The parameter $\bar{\varepsilon}$ allows to include the sensor readings of distant obstacles but within the range as possible temporary targets. Correspondingly, the temporary target value can be obtained from the set of candidate indexes contained in $\bar{i} \in \mathcal{I}(k_t)$. Therefore, a subset of the point cloud but containing the candidate indexes is defined:

$$\mathcal{I}_s(k_t) = \left\{ \rho_{Li,k_t}(\bar{i}) : \bar{i} \in \mathcal{I}(k_t) \right\}$$

Since the strategy is based on the cosine similarity of the candidate points with respect to the target position, a unitary vector pointing from the actual position towards the target should be defined:

$$\vec{p}_t(k_t) = \frac{p_{target}(k_t) - p_{EE}(k_t)}{|p_{target}(k_t) - p_{EE}(k_t)|}$$

Subsequently, on top of the definitions, the strategy for temporary target shifting is deployed in Algorithm 3.3. First (line 1), a cloud of vectors from the end effector current position towards the candidates point cloud $\mathcal{I}_s(k_t)$ is defined. This cloud of vectors contains all the directions were the trajectory could be shifted. Then, a cosine similarity set S_c is built given the cloud vector and the vector pointing towards the target position $\vec{p}_t(k_t)$ (lines 3-5). Consequently, a check (lines 6-10) of the cosine similarity set is performed with the aim to identify if there is visibility of the target point. The qualitative visibility is defined numerically by the tolerance TOL_{sc} which specifies a high value of cosine similarity (eg. 0.99). On the contrary, if there is not visibility, a candidate point of the point cloud $\mathcal{I}_s(k_t)$ is chosen based on the parameter max_{cos} which will maintain the candidate target far from the obstacle. Finally (lines 12-20), a linear path towards the candidate target $\hat{p}_{Li}$ is constructed with the objective to extract the furthest point belonging to the linear path but within the boundaries of the obstacle free Convex Polytope $\mathcal{S}(k_t)$.

Graphically, this method can be observed in Figure 3.5, where the light-green area represents the obstacle free polytope, the red line represents the unfeasible direct trajectory from the end effector position of the robot towards the target and the blue dotted line represents the shifted target position(represented by a blue star).

Algorithm 3.3 Target Shifting Strategy

Input : $\mathcal{I}_s(k_t)$, $\mathcal{S}(k_t)$, $flag_1$, $p_{target}(k_t)$, $p_{EE}(k_t)$, $TOL_{sc} \in [-1,1]$, $max_{cos} \in [-1,1]$

Output: $p_{temp}(k_t)$

Procedure : $temp\_target(\mathcal{I}_s(k_t), \mathcal{S}(k_t), flag_1, p_{target}(k_t), p_{EE}(k_t), TOL_{sc}, max_{cos})$

$\vec{\mathcal{I}}_s = (\mathcal{I}_s(k_t) - p_{EE}(k_t))./vecnorm(\mathcal{I}_s(k_t) - p_{EE}(k_t))$
$N_I \leftarrow length(\mathcal{I}_s(k_t))$
for $i = 1 : N_I$ **do**
  $S_c(i) = \vec{\mathcal{I}}_s(i) \cdot \vec{p}_t(k_t)$
end for
if $(\exists S_c(i) > TOL_{sc} \forall i) \wedge flag_1$ **then**
  $\hat{p}_{Li} = p_{target}$
else
  $\bar{i} : arg\,max_{\bar{i}=1,...,N_I} S_c(\bar{i}) < max_{cos}$
  $\hat{p}_{Li} = \mathcal{I}_s(k_t)(\bar{i})$
end if
$\vec{\hat{p}}_{Li} = \hat{p}_{Li} - p_{EE}(k_t)$
$res \leftarrow 1000$
for $j = 1 : res$ **do**
  $t = (j-1)/(Res-1)$
  $\hat{P}(i,:) = p_{EE}(k_t) + t\vec{\hat{p}}_{Li}$
end for
$\hat{P}_{in} = \{\hat{P}(i,:) : \hat{P}(i,:) \in \mathcal{S}(k_t)\ , \forall i = 1, ..., res\}$
$N_{P_{in}} \leftarrow length(P_{in})$
$p_{temp}(k_t) = arg\,max_{i=1,...,N_{P_{in}}} \|p_{EE}(k_t) - \hat{P}_{in}(i,:))\|_2^2$

Trajectory Generation

Given the convex under-approximation of the free space $\mathcal{S}(k_t)$, the actual position $p_{EE}(k_t)$ and the temporary target $p_{temp}(k_t)$, it is now possible to proceed with the strategy for trajectory generation in coordinate space. As the robot intermediate task are going to be defined in Cartesian space, the pose of the end effector $P_{EE}(k_t) = [p_{EE}(k_t), \varphi_{EE}(k_t)] \in R^6$ must be taken into account. In practice, this can be achieved with trapezoidal, circular or polynomial trajectories for the path parametrization.

Consequently, without loss of generality, the use of minimum jerk polynomial trajectories was employed given the advantages of this method to suppress vibrations and generation

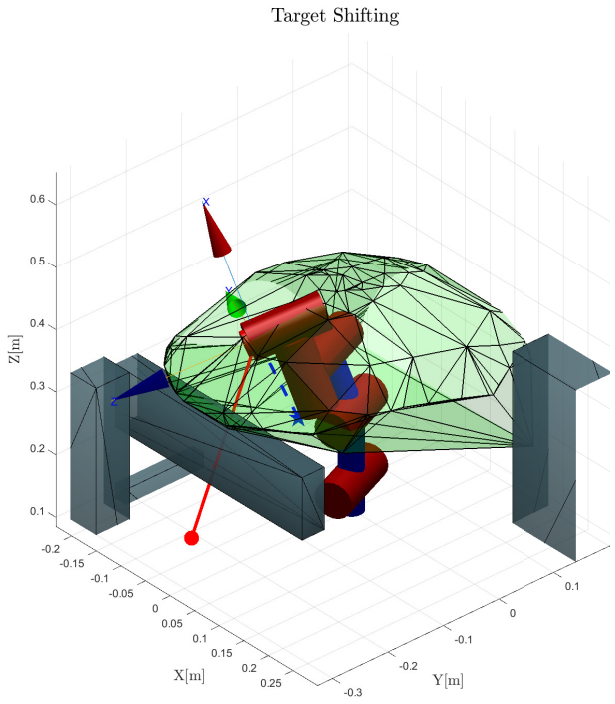


Figure 3.5: Target Shifting Example $max_{cos} = 0.8$. The obstacle free polytope is represented by the light-green geometry, the direct path towards the target is in solid red and the path towards the temporary target is in dashed blue.

of smooth movements while preserving accuracy [33].

Polynomial trajectories serve as a fundamental technique for defining and governing motion profiles within robotic systems. These trajectories, often characterized by their smoothness and controllability, provide a structured framework for shaping and directing the movement of robotic elements. They are represented by mathematical functions that offer a versatile means of controlling motion with precision and accuracy.

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n \tag{3.4}$$

The equation above demonstrates the general form of a polynomial trajectory, where coefficients $a_0, a_1, a_2, \ldots, a_n$ determine the specific shape and behavior of the trajectory. The choice of these coefficients determine the smoothness and shape of the trajectory generated. In order, to assign the polynomial parameters, the initial p_{EE} and final p_{temp} points, the travel time τ and the boundary conditions for velocity, acceleration and jerk, need to be defined.

As mentioned before, the minimum jerk trajectory generation was chosen for the purpose of this work, which apart from considering the resultant trajectory subject to a polyno-

mial parametrization(eg. Equation (3.4)), this method also minimizes the jerk executed through the path. Since the scope of this study is not focused in the trajectory generation algorithm, the algorithm function *minjerkpolytraj* developed by the the Robotic System Toolbox [38] of MATLAB [28] was used. Technical details of the formulation of the aforementioned algorithm are expressed in [33].

The function outputs positions X_D and velocities $\dot{X}_D$, accelerations $\ddot{X}_D$, and jerks $\dddot{X}_D$ at a given number of samples, N_{samp}, initial and final position, and, initial and final time points. It should be pointed out that apart from the position trajectory, an orientation trajectory must also be generated. As referred in Section 2.2.2, the Euler ZYZ orientation representation was chosen, thereby, the trajectory generated should be in term of the set of Euler angles : ϕ_E, θ_E and ψ_E. Nevertheless, the final orientation of the end effector is a sensible choice for the purpose of the task given to the robot and for the performed motions through the environment. Since the choice of the latter is mostly related to the specific task of the robot, without lose of generality, it is assumed that the robot orientation should be always pointing towards the temporary target position. This is represented in Figure 3.6 by following the example temporary target given in Figure 3.5.

For this purpose, Algorithm 3.4 was designed to return the set of Euler angles : ϕ_E, θ_E and ψ_E given the current and desired position of the robot.

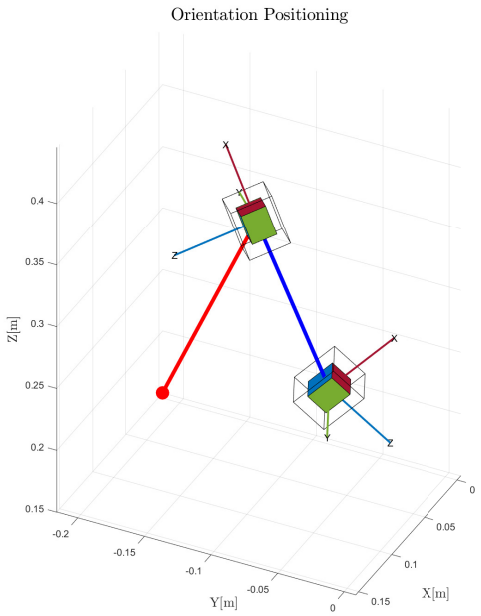


Figure 3.6: Orientation Positioning. The direct trajectory towards the target is represented by the solid red line, the trajectory towards the shifted target in solid blue and the orientation axis(x, y, z) are represented by the red, green and blue lines centered at the initial and final positions.

Algorithm 3.4 Orientation Positioning

Input : $p_{target}(k_t)$, $p_{EE}(k_t)$

Output: $\phi_E(k_t)$, $\theta_E(k_t)$, $\psi_E(k_t)$

Procedure : $get\_orientation(p_{target}(k_t), p_{EE}(k_t))$

1: Getting directional vector:
$\vec{p}_{\varphi} = \frac{p_{target} - p_{EE}(k_t)}{|p_{target} - p_{EE}(k_t)|}$

2: Computation of the first rotation angle:
$\phi_E(k_t) \leftarrow atan2(\vec{p}_{\varphi,(2)}, \vec{p}_{\varphi,(1)})$

3: Creation of intermediate vector to align with Z-axis:
$$R_z = \begin{bmatrix} cos(-\phi_E(k_t)) & -sin(-\phi_E(k_t)) & 0 \\ sin(-\phi_E(k_t)) & cos(-\phi_E(k_t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$\vec{p}_{temp1} = R_z \vec{p}_{\varphi}$

4: Computation of the second Y rotation angle:
$\theta_E(k_t) \leftarrow atan2(\vec{p}_{temp1,(3)}, |\vec{p}_{temp1})|$

5: Rotation of intermediate vector around Y-axis:
$$R_y = \begin{bmatrix} cos(-\theta_E(k_t)) & 0 & sin(-\theta_E(k_t)) \\ 0 & 1 & 0 \\ sin(-\theta_E(k_t)) & 0 & cos(-\theta_E(k_t))) \end{bmatrix}$$
$\vec{p}_{temp2} = R_y \vec{p}_{temp1}$

6: Computation of the third Z rotation angle:
$\psi_E(k_t) \leftarrow atan2(\vec{p}_{temp2,(2)}, \vec{p}_{temp2,(1)}|)$

Algorithm

Exploiting the algorithms detailed in the above sections, now a complete algorithm for Obstacle Free Trajectory Generation as a Local Path Planner can be developed by stating first some preliminary conditions.

First, the problem of generating a trajectory inside a variable volume polytope intrinsically leads to the generation of trajectories of different lengths, thus the time law cannot be assigned with a constant time parameter. Moreover, a concatenation of linear paths cannot be performed since the subsequent temporary target position is not know at iteration k_t. Therefore, as a rule of thumb, this time parameter can be defined proportional to a standard linear distance D_{tot} that the robot can perform for a given fixed time $\bar{t}_f$ (see Line 16).

Second, the variable $flag_1$ is referred to a parameter used to drift even more the temporary target position if the robot gets stuck close to an obstacle more than one k_t iteration (see

Lines 9-13).

Third, when the target position is detected to be inside the convex polytope(see Lines 18-21), the security and influence distances are relaxed(see Section 3.4.3), and the time to reach the target is extended to generated a slow positioning motion.

These both considerations were included in the complete Algorithm 3.5. After the parameters assignment, the algorithm starts by generating an obstacle free polytope $\mathcal{S}(k_t)$ given the current sensor measurements (line 7). Then, the belonging of target position $p_{target}(k_t)$ into the obstacle free polytope is performed. If the condition is not met, then the temporary target shifting strategy is performed (lines 9-15). As mentioned, the max_{cos} parameter it is conditioned if the target needs to be further shifted, this last drift is done by means of a monotonically decreasing function (line 10) which can be tuned with respect to the application. On the contrary, the target position p_{target} belongs to the convex polytope $\mathcal{S}(k_t)$ were a trajectory could be generated directly towards it. Finally, the ZYZ Euler orientation angles are computed pointing towards the temporary target position and the trajectory is generated with zero boundaries conditions(B_{cnd}) for velocity and acceleration (lines 23-27).

A 3D representation of the position and orientation trajectories generated(result of Algorithm 3.5) inside the obstacle free convex polytope is reported in Figure 3.7. Graphically, it can be observed that the orientation trajectory is iteratively moving towards the direction of the temporary target which is the desired behaviour. On the other hand, the generated trajectory vectors X_D and $\dot{X}_D$ are shown in Figure 3.8 respectively.

Algorithm 3.5 Obstacle Free Trajectory Generation

Input : $\mathcal{P}(k_t)$,$\mathcal{I}_s(k_t)$,$\mathcal{W}$,$\varphi_{EE}(k_t)$,$p_{EE}(k_t)$,$p_{target}(k_t)$, p_{obs}, D_{tot},t_{init},T_s,$\bar{t}_f$,$flag_1$

Parameters : d_i , d_s , d_{step} , r_{Li} ,$TOL_{sc} \in [-1,1]$, $max_{cos} \in [-1,1]$, $\bar{d}_{obs}$

Output: X_D ,$\dot{X}_D$, D_{is}

$d_{step} \leftarrow 0.005$

$r_{Li} \leftarrow 0.5$

$TOL_{sc} \leftarrow 0.995$

$max_{cos} \leftarrow 0.6$

$d_i \leftarrow \bar{d}_i$

$d_s \leftarrow \bar{d}_s$

$\mathcal{S}(k_t) \leftarrow polytope\_gen(\mathcal{P}(k_t), \mathcal{W}, p_{EE}(k_t), d_{step}, r_{Li})$

if $p_{target}(k_t) \notin \mathcal{S}(k_t)$ **then**

  if $p^6_{obs} < \bar{d}_{obs}$ **then**

    $max_{cos} = 0.5^{flag_1 * 0.8}$

    $flag_1 = flag_1 + 1$

  else

    $flag_1 = 1$

  end if

  $p_{temp}(k_t) \leftarrow temp\_target(\mathcal{I}_s(k_t), \mathcal{S}(k_t), flag_1, p_{target}(k_t), p_{EE}(k_t), TOL_{sc}, max_{cos})$

  $\tau_t(k_t) = \frac{\|p_{EE}(k_t) - p_{temp}(k_t)\|_2^2}{D_{tot}} \bar{t}_f$

  $D_{is} \leftarrow [\bar{d}_i, \bar{d}_s]$

else

  $p_{temp}(k_t) \leftarrow p_{target}(k_t)$

  $D_{is} \leftarrow [\bar{d}_i - 0.01, \bar{d}_s - 0.01]$

  $\tau_t(k_t) \leftarrow 2$

end if

$\varphi_{temp}(k_t) \leftarrow get\_orientation(p_{target}(k_t), p_{EE}(k_t))$

$B_{cnd} \leftarrow 0$

$t_{temp} = t_{init} : T_s : \tau_t(k_t)$

$N_{tp} \leftarrow length(t_{temp})$

$[X_D \dot{X}_D] \leftarrow minjerkpolytraj(t_{init}, \tau_t(k_t), [p_{EE}(k_t); \varphi_{EE}(k_t)], [p_{temp}(k_t); \varphi_{temp}(k_t)](k_t), N_{tp}, B_{cnd})$

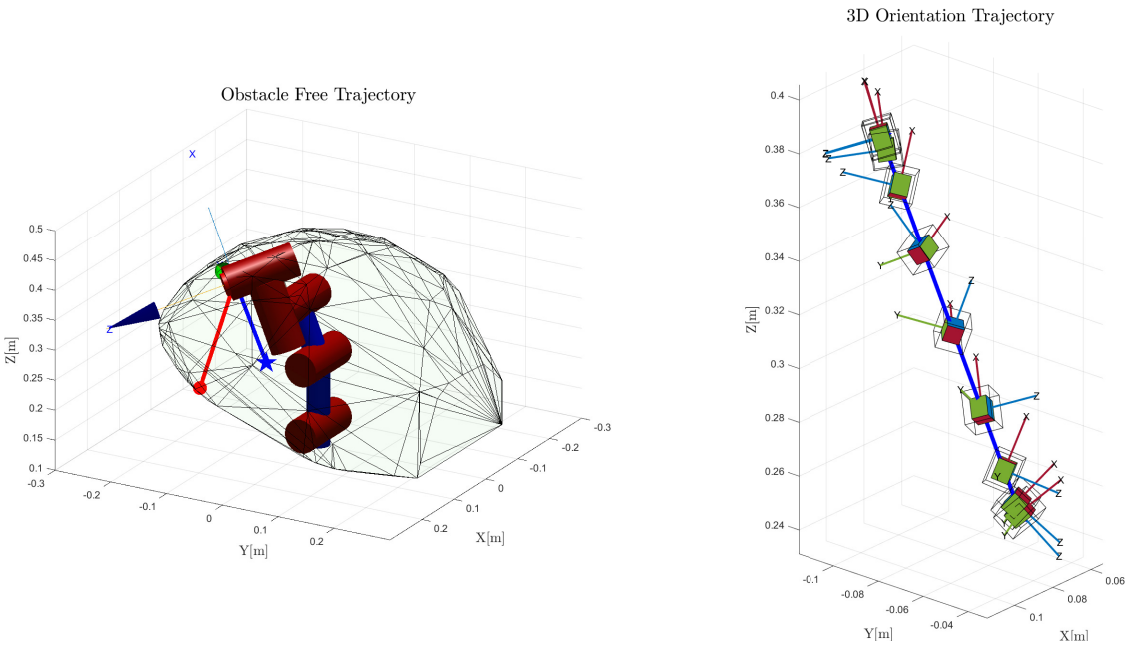
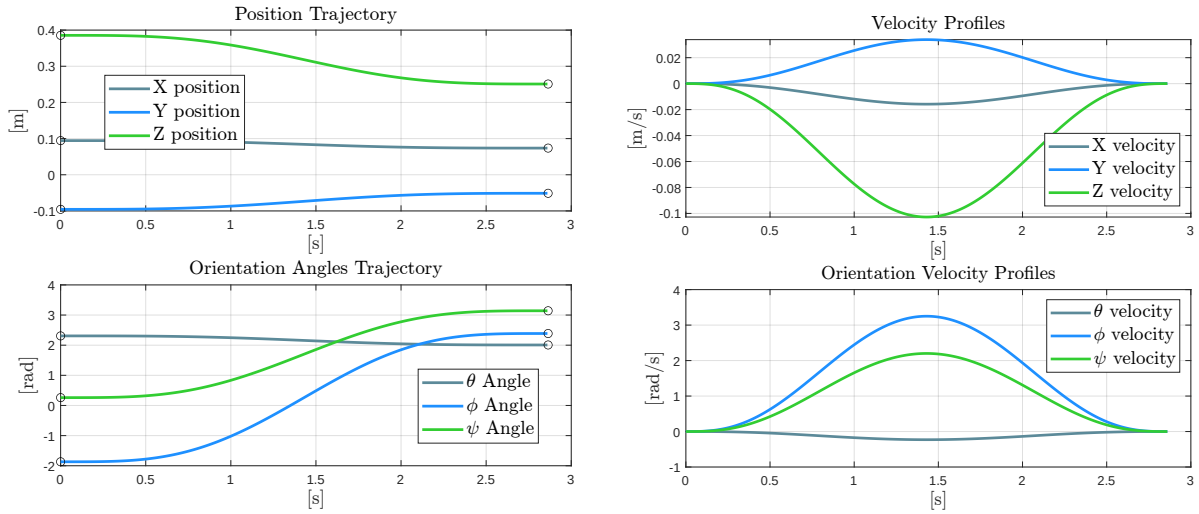


Figure 3.7: Position and Orientation Trajectories 3D Representation.The direct trajectory towards the target is represented by the solid red line, the trajectory towards the shifted target in solid blue and the orientation axis(x, y, z) are represented by the red, green and blue lines centered at the initial and final positions.



(a) Positions. (b) Orientation.

Figure 3.8: Polynomial Trajectory Profiles.

3.4. Inverse Kinematics as an Optimization Problem

As presented in Section 2.1, the standard Closed Loop Inverse Kinematics solution relays on the inverse of the analytical jacobian and the compensation of operational space error but does not accounts for joint constraints if they need to be enforced. Although different methods were developed to include the satisfaction of joint constraints in a Closed Loop solution, the inclusion of higher order constraints(velocity and acceleration bounds) are not straightforward to include [40]. On the contrary, during the last years in the field of robot control, online-solvable instantaneous constrained optimization has become increasingly popular(eg. [18][2][37][40][44][21]) due to its capacity to easily incorporate kinematic functions, position, velocity, and acceleration constraints into the problem formulation.

In light of the above mentioned, the current section will be focused on the discussion, formulation and derivation Inverse Kinematics as an optimization problem.

3.4.1. Related works

There exist few main approaches to define the robot task in the formulation of an optimization problem. One main approach is to define the robot task as a constraint while minimizing a suited cost function that can include velocity or acceleration terms [2]. Although the task error can also be included as a quadratic term in the cost function, just one step ahead prediction of system evolution is normally taken into account [40][44]. In detail, the future state and constraints behaviour are not included in the problem definition, so that a feasible solution at a given time instant can lead to an unfeasible configuration in the future [7].

With the increasing computational power, recent works [1][7] propose Model Predictive Control(MPC) to account for the system evolution over a prediction horizon and including the joint error evolution to account the state feedback. The advantage of this method is that the state feedback controller is optimal over the N future time steps and complies with kinematic constraints. Nevertheless, the aforementioned studies propose an architecture where the Inverse Kinematics solution is given as a reference to the MPC which is formulated only in joint space. Since the Inverse Kinematics is solved independently, the inclusion of tasks related to the operational space(eg. collision avoidance) cannot be directly enforced and complexity of the overall architecture is increased.

In [16] the operational space task as a reference of the MPC is included. Therefore, the inverse kinematics problem is solved at every time instant but allowing to impose not only kinematic constraints of the joints but also secondary tasks in the operational space

such human-robot distance maximization or collision avoidance. Lastly, this approach has gained importance since it also allows to include an online or dynamical motion planning executed by the optimization algorithm while following a nominal Cartesian task and accomplishing safety tasks in dynamical environments [15][13][23]. Even though the latter formulation shows significant advantages, the prediction of the future state of the system over an horizon has been a topic of discussion. Moreover, the inclusion of the task space error and velocity error in the cost function intrinsically contains the evaluation of the jacobian and homogeneous transformation functions which are a non-linear functions of the joint configuration.

With the aim to maintain the linear nature of the optimization problem over an horizon, several methods were developed. For example in [2] a time integration of the joint states is included in the QP problem for the prediction of the system behaviour over the horizon, while the future prediction of the Jacobian is carried out by the Taylor expansion using the Jacobian derivatives. Although this method is linear with respect to the decision variables, the computation of the Jacobian derivatives involves the evaluation of non-linear functions at each time step over the prediction horizon, which increases the total execution time and complexity. Instead, as proposed in [15][1][7] a linearization of the task at each iteration k is formulated by using the solution obtained at time $k-1$ to compute the future trajectory vector of joint positions q_{k+i} so that the overall linear nature of the QP problem is preserved.

While the ultimate quadratic program (QP) is an approximation of the initial non-linear problem, it allows the computational time to be of the same order of magnitude of local methods(in the order of milliseconds), but still conferring the advantages of a predictive strategy. Furthermore, the QP formulation of the IK problem is not limited to manipulators but it can also be used in a wide variety of robots such mobile, humanoids [20], hexapods [22] or a combination of them such as mobile robotic manipulators as shown in [1][7].

In this regard, the contribution of the present work is to propose a QP formulation of the IK-MPC which includes the operational space task as a reference, a predictive strategy, direct acceleration constraints (not only joint and velocity constraints), jerk penalization in the cost function, self-collision and obstacle avoidance as inequality constraints. It should be noticed that the latter acts more as a reactive strategy for obstacle avoidance and trajectory re-planning since the obstacle free reference trajectory is generated by a local path planning algorithm explained in Section 3.3.

3.4.2. Problem Formulation

In this section, the main formulation of the proposed approach for the Inverse Kinematics as an optimization problem is presented. First, the system model is defined, based on the discrete-time Linear Time-Invariant (LTI) system governing robotic kinematic chain. Second, the fundamental objective function is defined, denoted as $\mathcal{L}$, which the optimization routine seeks to minimize. Finally, the methodology for the prediction of the future system states and jacobian is discussed and presented.

System Model

Considering the discrete time LTI system describing the linear kinematics as a dual-integrator system, whose dynamic equation can be written as :

$$\underbrace{\begin{bmatrix} q(k+1) \\ \dot{q}(k+1) \end{bmatrix}}_{z(k+1)} = \underbrace{\begin{bmatrix} I_n & T_s I_n \\ 0_{n\times n} & I_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} q(k) \\ \dot{q}(k) \end{bmatrix}}_{z(k)} + \underbrace{\begin{bmatrix} 0.5{T_s}^2 I_n \\ T_s I_n \end{bmatrix}}_{B} u(k) \tag{3.5}$$

Equation (3.5) can be compacted in the form of :

$$z(k+1) = Az(k) + Bu(k) \tag{3.6}$$

Where $u(k) = \ddot{q}(k) \in \mathbb{R}^{n_u}$ and $z(0) = [q(0)^T, \dot{q}(0)^T]^T \in \mathbb{R}^{2n_q}$. The matrix $I_n \in \mathbb{R}^{n_q \times n_q}$ refers to an identity matrix of the dimension of the joints (in this case $n_q = 6$), T_s is the discrete integration time and k as the discrete time variable.

For the purpose of the state's feedback, equation (3.6) could be divided into :

$$\begin{cases} q(k+1) &= A_0 z(k) + B_0 u(k) \\ \dot{q}(k+1) &= A_1 z(k) + B_1 u(k) \end{cases} \tag{3.7}$$

where the matrices A_0 , A_1 , B_0 and B_1 are:

$$A_0 = \underbrace{\begin{bmatrix} I_{n_q} & 0_{n_q \times n_q} \end{bmatrix}}_{C_1} A \qquad B_0 = \begin{bmatrix} I_{n_q} & 0_{n_q \times n_q} \end{bmatrix} B$$

$$A_1 = \underbrace{\begin{bmatrix} 0_{n_q \times n_q} & I_{n_q} \end{bmatrix}}_{C_2} A \qquad B_1 = \begin{bmatrix} 0_{n_q \times n_q} & I_{n_q} \end{bmatrix} B$$

Cost Function

As discussed, the Inverse Kinematics problem considers a specific coordinate task $x_d(k)$ that the robot needs to reach at every time instant k while complying with motion constraints. First, a reference trajectory that can drive the robot given its current configuration $x_e(k) \leftarrow T_e^b(q(k))_{(4,1:3)}$ to a desired position $x_d(k)$ in M steps can be computed beforehand (e.g. through polynomial trajectories) obtaining a set of references $x_d(k+i), i = 1, ..., M$. Notice that the construction of this trajectory does not enforce any motion constraint in joint space. Then, a MPC structure of the Inverse Kinematics can be used to compute the sequence of control inputs to achieve the desired position in the operational space. The nature of the MPC formulation allows the computation of the control inputs at each control interaction, allowing it to deal with unforeseen events or dynamic scenarios. Moreover, as mentioned in Section 3.4.1, additional constraints linearly dependent on the sequence of control inputs can be easily included. At each time instant k, the MPC problem is defined as finding a M sequence of future control inputs $U_k = [u(0|k)^T, u(1|k)^T, ..., u(M-1|k)^T]^T$ that minimizes an specific cost function. The proposed cost function for the purpose of this work is shown in Equation (3.8) where the operational error and operational error velocity were included as well the joint velocity, joint acceleration and jerk for penalization purposes.

$$\mathcal{L} = \sum_{i=1}^{4} J_i \tag{3.8}$$

Where $J_1, ..., J_4$ are the desired tasks to minimize defined as:

$$J_1 = \sum_{i=1}^{M} \|J_A(q_{k+i})\dot{q}_{k+i} - \dot{x}_{d_{k+i}} + K(x_{d_{k+i}} - x_e(q_{k+i})\|_W^2$$
$$J_2 = \sum_{i=1}^{M} \|\dot{q}_{k+i}\|_S^2$$
$$J_3 = \sum_{i=1}^{M} \|\ddot{q}_{k+i}\|_R^2$$
$$J_4 = \sum_{i=1}^{M-1} \|\Delta\ddot{q}_{k+i}\|_{R_{\Delta\ddot{q}}}^2$$

The jerk is defined as $\Delta\ddot{q}(k+i) = \ddot{q}(k+i) - \ddot{q}(k)$, M is the prediction horizon and K is a positive definite weighting matrix related to the convergence of the error. W, S, R and

$R_{\Delta\ddot{q}}$ are symmetric and positive definite weighting matrices, and the following shorthand notation was employed:

$$\|x\|_Q^2 = x^T Q x \in \mathbb{R} \tag{3.9}$$

Therefore, the formalization of the optimization problem at hand can be written as:

$$\min_{U_k} \mathcal{L}(U_k, \hat{Q}_k, \hat{\dot{Q}}_k, X_d, \dot{X}_d, k) \tag{3.10}$$

Subject to :

$$z(i+1|k) = Az(i|k) + Bu(i|k) \tag{3.11a}$$
$$q(i|k) = C_1 z(i|k) \tag{3.11b}$$
$$\dot{q}(i|k) = C_2 z(i|k) \tag{3.11c}$$
$$A_{ineq} U(k) \leq b_{ineq} \tag{3.11d}$$

Notation $z(i|k)$ denotes the value of z at time $k+i$, predicted at time k. Where $\hat{Q}_k$ and $\hat{\dot{Q}}_k$ are the vectors related to the prediction through the horizon of the future joint position and velocity respectively. X_d and $\dot{X}_d$ are the vectors which refer to the position and velocity of a partial trajectory composed by M steps ahead. A_{ineq} and b_{ineq} are know inequality matrices defining the boundaries of the joint position, velocity and acceleration vectors. Additionally, the reactive Obstacle Avoidance function as well as the self-collision avoidance will be both also enforced as inequality constraints. For a detailed definition of the aforementioned variables refer to Section 3.4.5.

Prediction of the Future States

As mentioned in Section 3.4.1 and formulated in (3.8), the proposed approach includes a predictive strategy which is based in a linearization of the task at each iteration k by using the solution obtained at time $k-1$ to compute the approximation of the future trajectory vector of joint positions $q(k+i)$.

Recalling equation (2.1), in discrete form and employing the analytical jacobian, the relationship between the joint velocities and the task velocities can be expressed as follows:

$$J_A(q(k+1))\dot{q}(k+1) = \dot{x}_d(k+1) \tag{3.12}$$

It should be noticed that $J_A(q_k)$ is non-linear in terms of $u(k)$, therefore, it will depend on the actual configuration $q(k)$ and its evolution that can be obtained by iterating Equation (3.7). Accordingly, the predicted future evolution of the system states $q(k+i)$ and $\dot{q}(k+i)$ through the prediction horizon M results in :

$$\begin{aligned} \hat{q}(k+i) &= C_1 A^i z(k) + C_1 \sum_{j=0}^{i-1} A^{i-j-1} B\hat{u}(k+j) \ , \ i = 1, ..., M+1 \\ \dot{\hat{q}}(k+i) &= C_2 A^i z(k) + C_2 \sum_{j=0}^{i-1} A^{i-j-1} B\hat{u}(k+j) \ , \ i = 1, ..., M+1 \end{aligned} \tag{3.13}$$

Where $\hat{q}(k+i)$ and $\dot{\hat{q}}$ denote for the future prediction of the states evolution. Notice that the evolution of the system states depends on the actual state of the robot $z(k)$ at time k and the optimal vector $U_{k-1} = [\hat{u}(0|k-1)^T, \hat{u}(2|k-1)^T, ..., \hat{u}(M-1|k-1)^T]^T$ computed in the last iteration at time $k-1$ where the first element of the vector is the control input $u(0) = \hat{u}(0|k-1)$ applied to the system model. Moreover, the prediction of the Analytical Jacobian Matrix $J_A(q(k+i))$ and the pose vector $x_e(q_(k+i)) \leftarrow f_0(q_(k+i))$ through the horizon M are performed based on the results of Equation 3.13 starting from $q(k)$. In the following sections, the evolution of the joint configuration for the prediction of these non-linear functions will be exploited in the formulation of the optimization problem which will allow to maintain the linear nature of the cost function and constraints with respect to the vector of decision variables U.

3.4.3. Inequality Constraints

This section presents the formalization of the motion constraints and obstacle avoidance functions enforced into the optimization routine as inequality constraints. According to the Optimization Problem (3.10), the inequality constraints to be enforced can be divided in three main different types : joint boundaries, obstacle avoidance and self-collision avoidance. Accordingly the analysis of each type of constraint is divided as mentioned in the subsequent subsections.

Motion Constraints

As discussed before, the formulation of the optimization problem presented in Equation (3.10) allows to impose direct constraints on joint positions, velocities, and accelerations with the aim to ensure the robot's motion within its feasible kinematic limits. These boundaries should be imposed for the whole prediction horizon M such that:

$$
\begin{aligned}
q_{min} \leq q(k+i) \leq q_{max} \ , \forall i = 1, ..., M-1 \\
\dot{q}_{min} \leq \dot{q}(k+i) \leq \dot{q}_{max} \ , \forall i = 1, ..., M-1 \\
\ddot{q}_{min} \leq u(k+i) \leq \ddot{q}_{max} \ , \forall i = 1, ..., M-1
\end{aligned} \tag{3.14}
$$

The detailed derivation of these constraints with respect to the vector of decision variables U is formulated in Section 3.4.5.

Obstacle Avoidance as Inequality constraint

Given that the robotic arm will operate in dynamic scenarios with unforeseen events, it is not only sufficient to generate a trajectory free of obstacles but to consider an online reactive collision avoidance strategy between the manipulator and the obstacles that can be enforced into the Inverse Kinematics algorithm. There exist different approaches to formulate collision avoidance constraints for convex objects. Between the most popular methods one can find spherical and ellipsoidal approximate representation of the obstacle region, constructive solid geometry (CSG) or dual signed obstacle representation [24]. Nevertheless, most of those methods consider the whole convex set or primitives approximation of the whole set to formulate the collision avoidance constraint. Therefore, in the quest of an efficient mathematical formulation an inequality-based constraint via velocity damper for the collision avoidance task was established [19]. The advantage of this approach resides in its implementation as inequality constraint and the simplicity of using geometric models for the kinematic chain of the robot without any approximation, furthermore this method can be extended to the case of non-strictly convex objects. In the last years, this method was exploited by [41] and [42], which demonstrates its numerical efficiency.

In this work, the implementation of this strategy will account for the collision avoidance of the whole kinematic chain with obstacles of the environment and self-collision avoidance between the links of the manipulator.

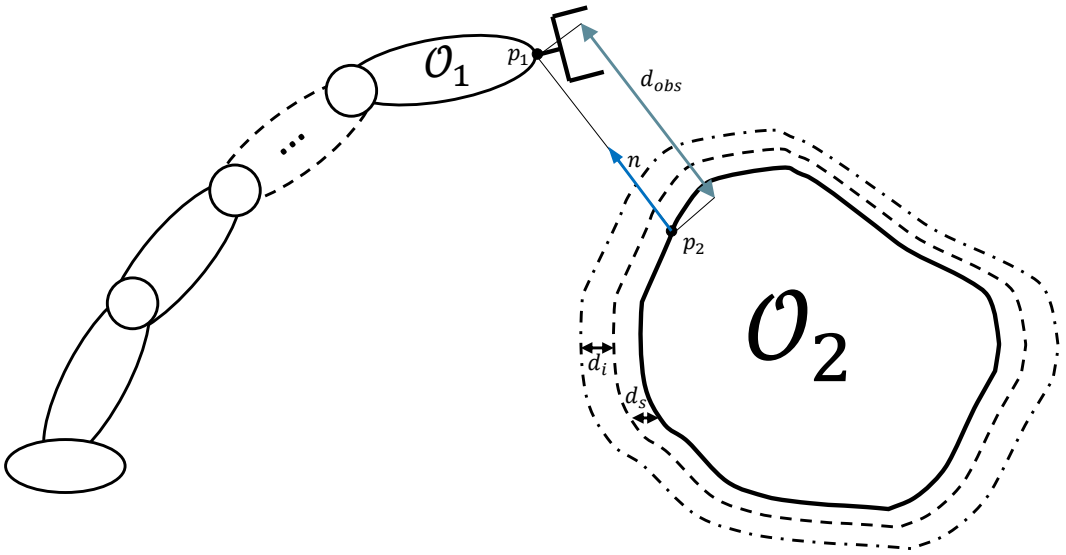


Figure 3.9: Closest distance and velocity damper definition.

Consider two convex sets $\mathcal{O}_1$ and $\mathcal{O}_2$ shown in Figure (3.9). $\mathcal{O}_2$ is a fixed obstacle for the respective time period and $\mathcal{O}_1$ represents one link position of the robot (in this case only the end effector position is represented). The points p_1 and p_2 denote the closest points between $\mathcal{O}_1$ and $\mathcal{O}_2$ so the distance d_{obs} is defined by $\|p_1 - p_2\|$. The derivative of the distance with respect to time, $\dot{d}_{obs}$, is bounded as follows :

$$-\dot{d}_{obs} \leq \zeta \frac{d - d_s}{d_i - d_s} \; ; \; d_i > d_s \tag{3.15}$$

Where ζ is a positive coefficient that regulated the convergence speed, d_s and d_i are positive values called security and influence distances respectively. The inequality (3.15) is called *velocity damper* and once the closest distance d becomes smaller that d_i, the velocity vector is restricted guaranteeing that d is never smaller than d_s.

Since $\mathcal{O}_2$ is fixed, $\dot{d}_{obs}$ can be expressed as a function of the unitary vector $\vec{n}$ and the time derivative of p_1:

$$\dot{d}_{obs} = \vec{n}^T \dot{p}_1 \; ; \; \vec{n} = \frac{p_1 - p_2}{d_{obs}} \tag{3.16}$$

Taking into account that $\mathcal{O}_1$ is belongs to the robotic chain, $\dot{p}_1$ can be expressed as as a function of the robot generalized coordinates q and its derivatives $\dot{q}$. Recalling 2.1, this

relation can be expressed as follows:

$$\dot{p}_1 = J_{p1}(q)\dot{q} \tag{3.17}$$

Where J_{p1} is the relative translation Jacobian matrix of point p_1. By replacing (3.16) in (3.15), it becomes a linear inequality constraint over the robot joint velocity vector:

$$-\vec{n}^T J_{p1}(q)\dot{q} \leq \zeta \frac{d - d_s}{d_i - d_s} \tag{3.18}$$

Thus, the inequality (3.18) is in the form that can be imposed as a linear inequality constraint in (3.8). The detailed derivation of the latter over a prediction horizon and considering self-collision avoidance will be developed in Section 3.4.5 .

To this point, it is necessary to state the closest distance between the points and the method used to define it. Therefore, the closest distance to the obstacles was defined as a function of the measurements of the exteroceptive sensor received at every time instant k_{Li} which happens every τ_{Li} seconds.

Lets denote N_{obs} as the number of obstacles to detect and $\mathcal{S}(k_{Li}) = [s_{k_{Li}}(0), ..., s_{k_{Li}}(N_s - 1)] \in \mathbb{R}^{3\times N_s}$ the N_s filtrated readings of the LiDAR sensor at time k_{Li}. Moreover, $\mathcal{S}(k_{Li}) \in \mathcal{P}(k_{Li})$ and denotes all the readings that are inside the maximum range of measurement of the sensor. Notice that the ground can also be recognized as a close obstacle, this it is also considered in the Convex Approximation of the Free Space in order to avoiding colliding with the low level ground. Once the sub-set $\mathcal{S}(k_{Li})$ is computed in a the related k_{Li} time instant, the proposed routine to get the closest distance from the N_{obs} obstacle to the point p_1 is given by Algorithm 3.6. After initializing values, (line 3) a *K-nearest neighbors*(*kmeans*) algorithm is used in order to classify the sensor readings from each obstacle. The resultant clusters, namely $\mathcal{S}_i$, are sub-spaces of $\mathcal{S}(k_{Li})$; furthermore $\mathcal{S} = \bigcup_{j=1}^{N_{km}} \mathcal{S}_j$. Then, an interactive search of the closest distance from each link to each obstacle cluster is performed (lines 4-13).

Algorithm 3.6 Closest Distance Computation

Input : $[p_{l_1}(k_{Li}), ..., p_{l_6}(k_{Li})]$, $\mathcal{S}(k_{Li})$, N_{km}

Output: $p_{obs}(k_{Li}) \in \mathbb{R}^{3\times N_{obs}}$

Procedure: $closest\_obstacles([p_{l_1}(k_{Li}), ..., p_{l_6}(k_{Li})], \mathcal{S}(k_{Li}), N_{km})$

$N_{obs} \leftarrow 4$
$n_l \leftarrow 6$
Generation of N_c sub-spaces of $\mathcal{S}(k_{Li})$ using *k-means* clustering technique: $\mathcal{S}_i = kmeans(\mathcal{S}(k_{Li}), N_{km})$, $i = 1, ..., N_{km}$
for $ind = 1 : n_l$ **do**
  for $i = 1 : N_{km}$ **do**
    $N_S \leftarrow length(\mathcal{S}_i)$
    for $j = 1 : N_S$ **do**
      $dist(j) = \|S_i^j - p_{l_{ind}}\|_2^2$
    end for
    $\bar{i} : arg\,min_{\bar{i}=1,...,N_S}\, dist(\bar{i})$
    ${p_{obs}}_{ind}^{i} = S_i(\bar{i})$
  end for
end for
$p_{obs}(k_{Li}) = [p_{obs_1}, ..., p_{obs_{n_l}}]$

The representation of the aforementioned method is shown in Figure 3.10 and 3.11, where the light-blue spheres attached to the manipulator represent the link positions, and the remaining colored spheres the closest distances from each link to each cluster. The radius of the spheres represents the desired security distance factor d_s in which the obstacles should not collide with the kinematic chain. Notice that the relative distance between each link position and each obstacle is computed given the actual configuration of the robot $q(k_{Li})$, this means that during motion the obstacle relative positions are time-varying. This dependency can be observed in the different positions between Figure 3.10 and Figure 3.11, where the different colored dots represents each cluster and the respective spheres the closest obstacles from each link to each cluster.

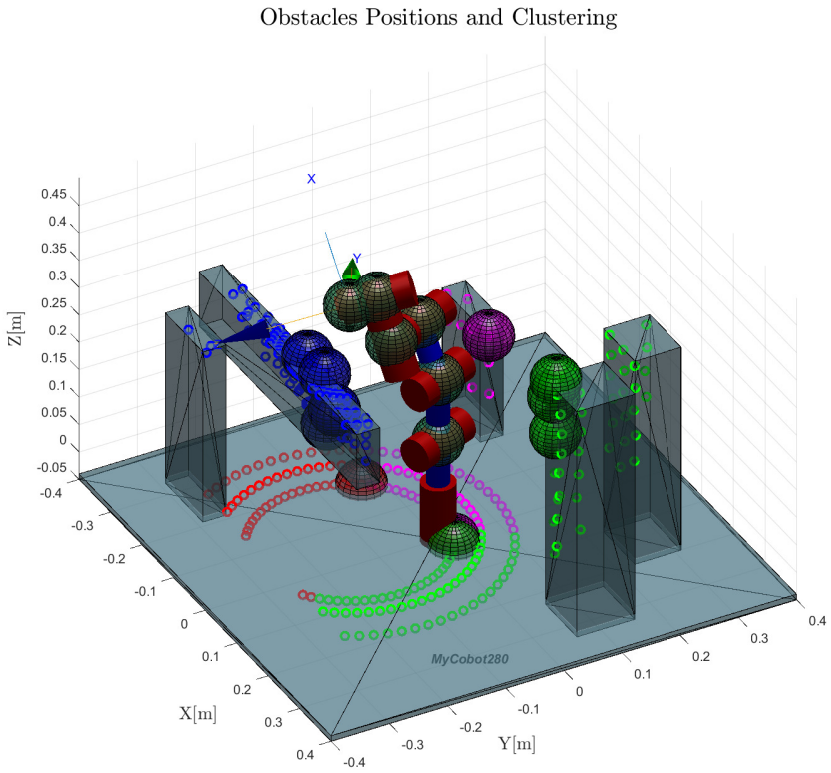


Figure 3.10: Example 1: Obstacles clustering. $q(k_{Li}) = [\frac{-\pi}{8}, 0, \frac{-\pi}{8}, 0, -2.44, 0], d_s = 0.4$

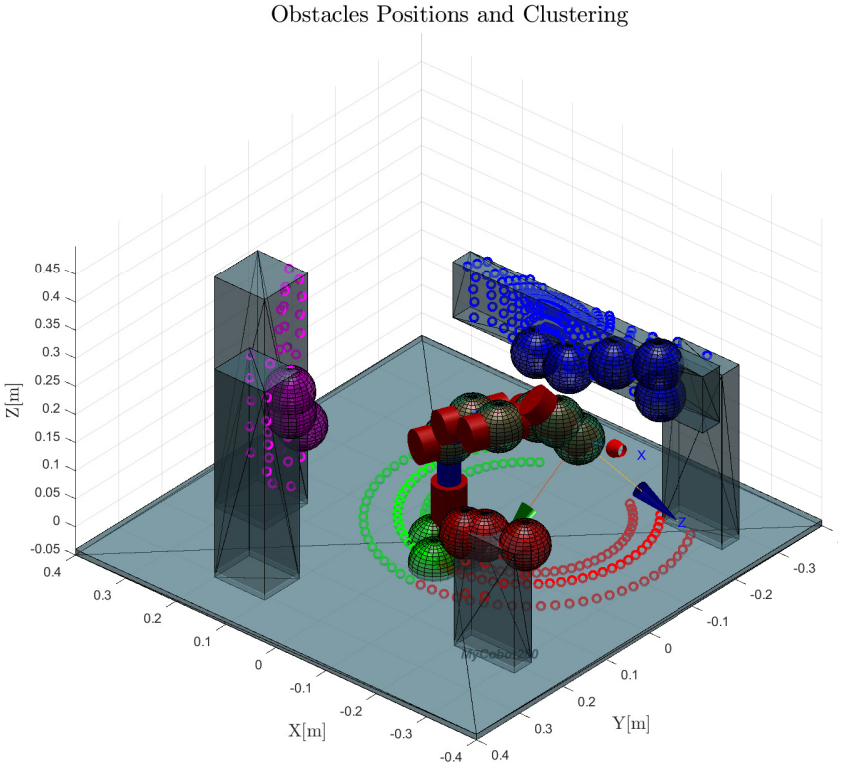


Figure 3.11: Example 2: Obstacles clustering. $q(k_{Li}) = [\frac{-\pi}{8}, \frac{-\pi}{4}, \frac{-\pi}{8}, 0, 2.44, 0], d_s = 0.4$

Self-Collision Avoidance

This constraint, integrated into the optimization framework, aims to prevent the robotic manipulator from colliding with its own structure or links during motion. The inclusion of this constraint aims to prove the capabilities of the proposed method; therefore, only the joint frame positions are considered to cover self-collisions, instead of every part of the whole joint body (which could be done by primitive approximations).

The derivation and inclusion of this constraint will be also based on Equation 3.18, but considering instead the position of the links that are most likely to collide. In other words, although the collision avoidance functionality can be implemented for the whole kinematic chain, this was included only for the joints that are most likely to collide as it can be seen in Figure 3.12.

Thus, the relation between self collision avoidance between each link position can be expressed as follows:

$$p_{link4} \leftrightarrow p_{link1} \qquad p_{link3} \leftrightarrow p_{link1}$$

$$p_{link6} \leftrightarrow p_{link1} \qquad p_{link5} \leftrightarrow p_{link1}$$

$$p_{link6} \leftrightarrow p_{link2} \qquad p_{link5} \leftrightarrow p_{link2}$$

Graphically the position of each link and the relation with the self-collision aforementioned considerations is shown in Figure 3.12 where the green spheres refers to the link position with a radius of 4 cm (d_s security distance), the red arrows represents the self-collision relation between link p_{l_1} and the other links, and the blue arrows represents the self-collision relation between link p_{l_2} and the other links. As mentioned, for the purpose of this work only these relations where assumed, however, other relations for self-collision or obstacle avoidance can be included.

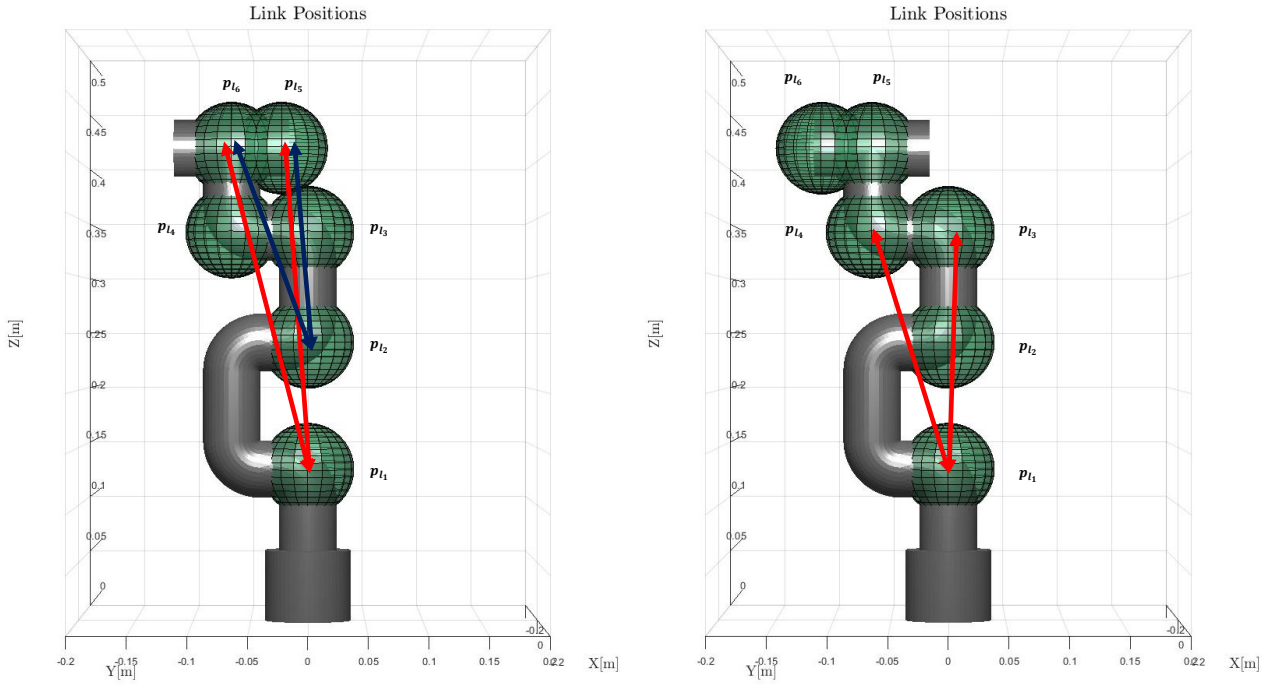


Figure 3.12: Self-collision links relations.

The further derivation of the self-collision avoidance function as linear inequality constraints with respect to the decision variable $u(k)$ is detailed in Section 3.4.5.

3.4.4. Model Predictive Control (MPC)

In order to impose a receding horizon strategy as proposed in Equation 3.8, thus, a Model Predictive Control methodology will be exploited and included in the Inverse Kinematics problem formulation.

Model Predictive Control (MPC) is a powerful control strategy widely used in various engineering applications. It is an advanced control approach that operates by iteratively solving the control problem as an optimization problem over a finite prediction horizon M. MPC incorporates a dynamic model of the system, constraints on inputs and states, and an objective function to determine optimal control inputs. It relies on the Receding Horizon(RH) principle in which at every time instant the main objective is to find the optimal sequence of control inputs $[u(k), ..., u(k+N-1)]$ over a finite time horizon that minimizes a predefined cost function and apply to the system only the first element $u^{\circ}(k)$ of the optimal sequence [25].

To solve the MPC optimization problem, various numerical methods and solvers can be employed depending on the nature of the overall formulation. Common approaches include: Quadratic Programming(QP) and Sequential Quadratic Programming(SQP). As explained before(refer to Section 3.4.1), by maintaining the linear nature of the optimization problem, a quadratic programming solver can be utilized which will also provide computationally efficiency while still conferring the advantages of the receding horizon strategy. Therefore, the QP will be the preferred method for the purpose of this work.

3.4.5. Quadratic Programming Formulation

QP is a mathematical technique used to solve optimization problems that involve quadratic objective functions subject to linear constraints. The use of QP in robotics offers several advantages, including the ability to handle complex or high-dimensional problems. It provides a mathematical framework for incorporating various constraints, and facilitate real-time or real-time solutions. This versatility is crucial in tasks where robots must adapt to changing environments and dynamic scenarios.

The general form of a Quadratic Programming (QP) problem can be expressed as follows:

Minimize:

$$\min_{\mathbf{x}} \ \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} \tag{3.19}$$

subject to:

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}$$

$$\mathbf{C}\mathbf{x} = \mathbf{d}$$

Where:

$\mathbf{x}$: Vector of optimization variables

$\mathbf{H}$: Symmetric positive semi-definite matrix

$\mathbf{c}$: Vector of linear coefficients

$\mathbf{A}$: Matrix of inequality constraint coefficients

$\mathbf{b}$: Vector of inequality constraint bounds

$\mathbf{C}$: Matrix of equality constraint coefficients

$\mathbf{d}$: Vector of equality constraint bounds

More specifically for this work, in the case of inequality-constrained QPs, finding a KKT triplet that satisfies (3.19) is a non-linear and non-smooth problem. Therefore, there

exist two main iterative methods to deal with it *active-set* and *interior-point*. A latter discussion between both methods will be deployed in the results section.

Cost Function

In this section the formal derivation of the Cost Function for the Quadratic Programming problem is presented. The dimension of the vectors presented in this section will be referred in function of dimension n_u of the decision variable $u(k)$, the dimension n_z of the state vector $z(k)$, the dimension n_q of the number of joints $q(k)$ and the prediction horizon M. The related constant diagonal matrices are denoted as:

$$\begin{aligned}
\bar{W} &= diag(\begin{bmatrix} W & W & \dots & W \end{bmatrix}) \in \mathbb{R}^{(M)n_q \times (M)n_q} \\
\bar{S} &= diag(\begin{bmatrix} S & S & \dots & S \end{bmatrix}) \in \mathbb{R}^{(M)n_q \times (M)n_q} \\
\bar{R} &= diag(\begin{bmatrix} R & R & \dots & R \end{bmatrix}) \in \mathbb{R}^{(M)n_u \times (M)n_u} \\
\bar{R}_{\Delta\ddot{q}} &= diag(\begin{bmatrix} R_{\Delta\ddot{q}} & R_{\Delta\ddot{q}} & \dots & R_{\Delta\ddot{q}} \end{bmatrix}) \in \mathbb{R}^{(M)n_u \times (M)n_u} \\
\mathcal{K} &= diag(\begin{bmatrix} K & K & \dots & K \end{bmatrix}) \in \mathbb{R}^{(M)n_q \times (M)n_q}
\end{aligned}$$

As stated in Equation (3.13) the prediction vectors of joint position and joint velocities can be respectively defined as:

$$\begin{aligned}
Q &= [\hat{q}(1|k)^T, \hat{q}(2|k)^T, ..., \hat{q}(M|k)^T]^T \\
\dot{Q} &= [\hat{\dot{q}}(1|k)^T, \hat{\dot{q}}(2|k)^T, ..., \hat{\dot{q}}(M|k)^T]^T
\end{aligned}$$

Thus, the future evolution of the joint positions is characterized by:

$$Q = \Lambda_q + \Lambda_U U \tag{3.20}$$

And the evolution of the joint velocities:

$$\dot{Q} = \Phi_q + \Phi_U U \tag{3.21}$$

Where :

$$\Lambda_q = \overbrace{\begin{bmatrix} C_1A \\ C_1A^2 \\ \vdots \\ C_1A^{M+1} \end{bmatrix}}^{\Lambda_A} z(0) \qquad \Phi_q = \overbrace{\begin{bmatrix} C_2A \\ C_2A^2 \\ \vdots \\ C_2A^{M+1} \end{bmatrix}}^{\Phi_A} z(0)$$

And the associated constant matrices:

$$\Lambda_U = \begin{bmatrix} C_1B & 0 & \dots & \dots & 0 \\ C_1AB & C_1B & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ C_1A^{M-1}B & C_1A^{M-2}B & \dots & \dots & C_1B \end{bmatrix} \quad \Phi_U = \begin{bmatrix} C_2B & 0 & \dots & \dots & 0 \\ C_2AB & C_2B & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ C_2A^{M-1}B & C_2A^{M-2}B & \dots & \dots & C_2B \end{bmatrix}$$

Moreover, if the trajectory position and velocity vectors are compacted as:

$$X_D = \begin{bmatrix} x_d(k) \\ x_d(k+1) \\ \vdots \\ x_d(k+M) \end{bmatrix} \qquad \dot{X}_D = \begin{bmatrix} \dot{x}_d(k) \\ \dot{x}_d(k+1) \\ \vdots \\ \dot{x}_d(k+M) \end{bmatrix}$$

And the evolution of the non-linear functions respectively, future evolution of the jacobian matrix, and the end effector position are defined as:

$$\mathcal{J}_A = \begin{bmatrix} J_A(\hat{q}(k)) & 0 & \dots & 0 \\ 0 & J_A(\hat{q}(k+1)) & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & J_A(\hat{q}(k+M)) \end{bmatrix} \qquad \dot{X}_E = \begin{bmatrix} f_0(\hat{q}(k)) \\ f_0(\hat{q}(k+1)) \\ \vdots \\ f_0(\hat{q}(k+M)) \end{bmatrix}$$

It should be remarked that the aforementioned matrices are a prediction based on Equation (3.13), therefore, to maintain the linear nature of the problem they remain constant at every time iteration and recalculated at the next iteration.

In addition, the cost function proposed in (3.8) can be rewritten in matrix form:

$$J_1 = \left\| \dot{X}_D - \mathcal{J}_A\dot{Q} + \mathcal{K}(X_D - X_E) \right\|_W^2 \tag{3.22}$$

If we denote the task space error as $\mathcal{E} = X_D - X_E$, replacing (3.21) in matrix form and operating, we obtain:

$$J_1 = \left\| \underbrace{\dot{X}_D - \mathcal{J}_A\Phi_q + \mathcal{K}\mathcal{E}}_{\Lambda_X} \underbrace{-\mathcal{J}_A\Phi_U}_{\Gamma_U} U \right\|_W^2 \tag{3.23}$$

Thus, by performing the norm, the cost function J_1 can be defined in matrix form as :

$$J_1 = [\Lambda_X + \Gamma_U U]^T \bar{W} [\Lambda_X + \Gamma_U U] \tag{3.24}$$

Operating and eliminating the terms that does not depend on the vector of decision variables U, we obtain:

$$J_1 = 2[\Lambda_X^T \bar{W} \Gamma_U]^T U + U^T [\Gamma_U \bar{W} \Gamma_U] U \tag{3.25}$$

Similarly for the other terms of the cost function :

$$\begin{aligned} J_2 &= [\Phi_q + \Phi_U U]^T \bar{S} [\Phi_q + \Phi_U U] = 2(\Phi_q^T \bar{S} \Phi_U)^T U + U^T (\Phi_U \bar{S} \Phi_U) U \\ J_3 &= U^T \bar{R} U \end{aligned} \tag{3.26}$$

As for the term related to the penalization of the vector of the joint jerk which was stated as $\Delta\ddot{q}(k+i) = \ddot{q}(k+i) - \ddot{q}(k)$,and since $\ddot{q}(k)$ is the decision variable $u(k)$, the formulation of J_4 in matrix form can be rewritten as:

$$\begin{aligned} J_4 &= \left\| \underbrace{\begin{bmatrix} -I_{n_u} & I_{n_u} & 0 & 0 & \dots & 0 \\ 0 & -I_{n_u} & I_{n_u} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \dots & 0 \\ 0 & 0 & \dots & 0 & I_{n_u} & -I_{n_u} \end{bmatrix}}_{\Lambda_T} U \right\|_{\bar{R}_{\Delta\ddot{q}}}^2 \\ J_4 &= U^T (\Lambda_T^T \bar{R}_{\Delta\ddot{q}} \Lambda_T) U \end{aligned} \tag{3.27}$$

Thus, composing all the terms of the cost function, we have:

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^{4} J_i \\ \mathcal{L} &= \left[2(\Phi_q^T \bar{S} \Phi_U)^T + 2(\Lambda_X^T \bar{W} \Gamma_U)^T \right] \mathbf{U} \\ &\quad + \mathbf{U}^T \left[(\Gamma_U \bar{W} \Gamma_U) + (\Phi_U \bar{S} \Phi_U) + \bar{R} + (\Lambda_T^T \bar{R}_{\Delta\ddot{q}} \Lambda_T) \right] \mathbf{U} \end{aligned} \tag{3.28}$$

Finally, diving the cost by two, the form proposed in 3.10, is derived as the following QP:

$$\min_{U} \underbrace{(\Phi_q^T \bar{S} \Phi_U + \Lambda_X^T \bar{W} \Gamma_U)^T}_{c^T} U + \frac{1}{2} U^T \underbrace{(\Gamma_U \bar{W} \Gamma_U + \Phi_U \bar{S} \Phi_U + \bar{R} + \Lambda_T^T \bar{R}_{\Delta \ddot{q}} \Lambda_T)}_{H} U \tag{3.29}$$

Inequality Constraints

Motion Limits

Being the problem on hand a constrained optimization problem, appropriate bounds have been chosen for the variables through the prediction horizon. Replacing Equation (3.21) in (3.14)

$$\begin{aligned} Q_{min} &\leq \Lambda_q + \Lambda_U U \leq Q_{max} \\ \dot{Q}_{min} &\leq \Phi_q + \Phi_U U \leq \dot{Q}_{max} \\ \ddot{Q}_{min} &\leq U \leq \ddot{Q}_{max} \end{aligned} \tag{3.30}$$

Where the boundary matrices are defines as:

$$\begin{aligned} Q_{min} &= \begin{bmatrix} {q_{min}}^T & {q_{min}}^T & \dots & {q_{min}}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \\ Q_{max} &= \begin{bmatrix} {q_{max}}^T & {q_{max}}^T & \dots & {q_{max}}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \\ \dot{Q}_{min} &= \begin{bmatrix} \dot{q}_{min}^T & \dot{q}_{min}^T & \dots & \dot{q}_{min}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \\ \dot{Q}_{max} &= \begin{bmatrix} \dot{q}_{max}^T & \dot{q}_{max}^T & \dots & \dot{q}_{max}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \\ \ddot{Q}_{min} &= \begin{bmatrix} \ddot{q}_{min}^T & \ddot{q}_{min}^T & \dots & \ddot{q}_{min}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \\ \ddot{Q}_{max} &= \begin{bmatrix} \ddot{q}_{max}^T & \ddot{q}_{max}^T & \dots & \ddot{q}_{max}^T \end{bmatrix}^T \in \mathbb{R}^{Mn_u} \end{aligned}$$

Thus, the inequalities related to the boundaries of joints can be rewritten in terms of the vector of decision variables as:

$$\underbrace{\begin{bmatrix} \Lambda_U \\ -\Lambda_U \\ \Phi_U \\ -\Phi_U \\ I_{n_u \times n_u} \\ -I_{n_u \times n_u} \end{bmatrix}}_{AL_{ineq}} U \leq \underbrace{\begin{bmatrix} Q_{max} - \Lambda_q \\ -Q_{min} + \Lambda_q \\ \dot{Q}_{max} - \Phi_q \\ -\dot{Q}_{min} + \Phi_q \\ \ddot{Q}_{max} \\ -\ddot{Q}_{min} \end{bmatrix}}_{bL_{ineq}} \tag{3.31}$$

Obstacle Avoidance

Regarding the obstacle avoidance function, the formulation as inequality constraint will be based on Equation (3.18).

Defining the vector of one step ahead prediction states :

$$\hat{z}(k+1) = \begin{bmatrix} \hat{q}(k+1) \\ \dot{\hat{q}}(k+1) \end{bmatrix}$$

Moreover, in order to formulate the obstacle avoidance to the full kinematic chain (each link), the partial Jacobian functions $Jp_1, ..., Jp_6$ will be needed. Additionally, as a result of Algorithm (3.6), if we define the location of the position of just one obstacle $p_{obs_1}(k_{Li})$ at time k_{Li} and the positions of all joints $[p_{l_1}(k), ..., p_{l_6}(k)] = Tf(q(k))$ at every time k (see Figure 3.13). Then, distances from each joint to the respective obstacle can be computed as :

$$d_i = \|p_{l_i}(k) - p_{obs_i}(k_{Li})\|_2^2 \ , \ \forall i = 1, ..., 6 \tag{3.32}$$

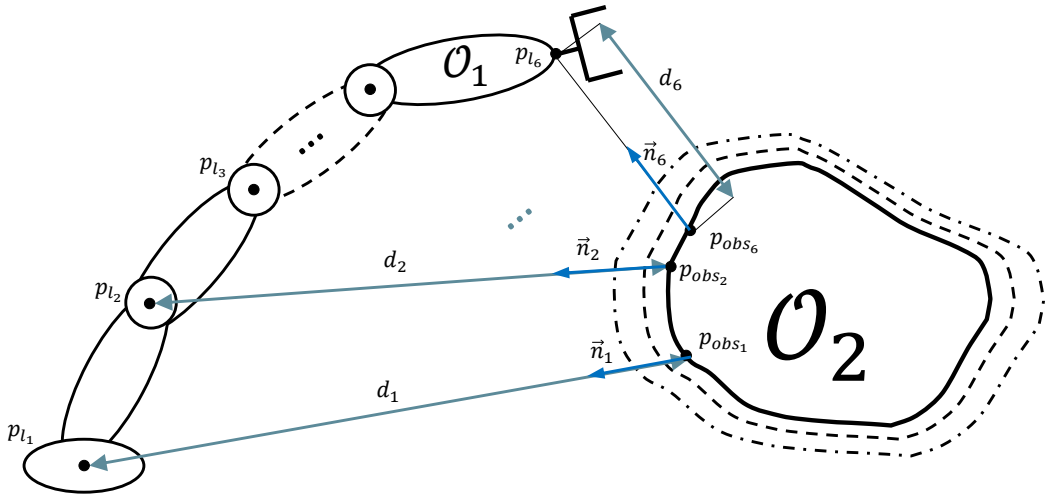


Figure 3.13: Joint to Obstacle Distances at time k_{Li}.

Should be noticed that the distances and their respective unit vectors $\vec{n}_i$ as well as the Jacobian functions are dependent on the actual configuration of the robot $q(k)$.

Thus, based on (3.6), it can be derived for only one step ahead prediction :

$$-\Lambda_J \bar{B}_1 \mathbf{u}(\mathbf{k}) \leq \Psi_1 \bar{D}(q(k+1)) + \Lambda_J(q(k+1)) \bar{A}_1 \hat{z}(k+1) \tag{3.33}$$

Where :

$$\bar{D}(q(k+1)) = \begin{bmatrix} d_1^T & d_2^T & \dots & d_6^T \end{bmatrix}^T \in \mathbb{R}^6$$
$$\bar{B}_1 = = \begin{bmatrix} B_1 & B_1 & \dots & B_1^T \end{bmatrix}^T \in \mathbb{R}^{6n_q \times n_u}$$
$$\bar{A}_1 = = \begin{bmatrix} A_1 & A_1 & \dots & A_1^T \end{bmatrix}^T \in \mathbb{R}^{6n_q \times n_z}$$

And

$$\Lambda_J(q(k+1)) = \begin{bmatrix} \vec{n}_1^T J_{p1} & 0 & \dots & 0 \\ 0 & \vec{n}_2^T J_{p2} & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \vec{n}_6^T J_{p6} \end{bmatrix} \qquad \Psi_1 = \begin{bmatrix} \zeta_1 & 0 & \dots & 0 \\ 0 & \zeta_1 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \zeta_1 \end{bmatrix}$$

Consequently, Equation (3.33) can be extended trough the prediction horizon M and considering the evolution of the future states denoted in (3.20) :

$$-\Phi_J \mathcal{B}_1 \mathbf{U} \leq \overline{\Psi} \mathcal{D} \Phi_J + \mathcal{A}_1 \widetilde{Z} \tag{3.34}$$

Where the state dependent matrices are:

$$\mathcal{D} = \begin{bmatrix} \bar{D}(\hat{q}_{k+1}) \\ \bar{D}(\hat{q}_{k+2}) \\ \vdots \\ \bar{D}(\hat{q}_{k+M}) \end{bmatrix} \qquad \widetilde{Z} = \begin{bmatrix} \dot{\hat{z}}_{k+1} \\ \dot{\hat{z}}_{k+2} \\ \vdots \\ \dot{\hat{z}}_{k+M} \end{bmatrix} \qquad \Phi_J = \begin{bmatrix} \Lambda_J(\hat{q}_{k+1}) & 0 & \dots & 0 \\ 0 & \Lambda_J(\hat{q}_{k+2}) & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \Lambda_J(\hat{q}_{k+M}) \end{bmatrix}$$

And the constant matrices:

$$\mathcal{A}_1 = \begin{bmatrix} \bar{A}_1 & 0 & \dots & 0 \\ 0 & \bar{A}_1 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \bar{A}_1 \end{bmatrix} \qquad \mathcal{B}_1 = \begin{bmatrix} \bar{B}_1 & 0 & \dots & 0 \\ 0 & \bar{B}_1 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \bar{B}_1 \end{bmatrix} \qquad \overline{\Psi} = \begin{bmatrix} \Psi_1 & 0 & \dots & 0 \\ 0 & \Psi_2 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \Psi_6 \end{bmatrix}$$

For clarity, the dimensions of the described matrices are denote as follows: $\mathcal{D} \in \mathbb{R}^{nq \times M}$, $\mathcal{B}_1 \in \mathbb{R}^{n_q^2 M \times n_u M}$, $\mathcal{A}_1 \in \mathbb{R}^{n_q^2 M \times 2n_q M}$, $\Phi_J \in \mathbb{R}^{n_q M \times n_q^2 M}$, $\overline{\Psi} \in \mathbb{R}^{n_q M \times n_q M}$, $\widetilde{Z} \in \mathbb{R}^{n_q M \times 2n_q M}$

Moreover, with the aim to relax the constraints, the convergence rate ζ is defined with decreasing values trough the horizon, this is the reason why the matrix $\overline{\Psi}$ refers to different matrices Ψ.

As mentioned, Equation (3.34) shows the formulation for the inequality constraints in the case of just one obstacle with reference to all the joint positions. Therefore, the notation for the i obstacle can be simplified as:

$$\begin{aligned} AO_i &= -\Phi_J \mathcal{B}_1 \\ bO_i &= \overline{\Psi}\mathcal{D} + \Phi_J \mathcal{A}_1 \widetilde{Z} \end{aligned}$$

Thereby, this can be extended for a specific number of N_{obs} to include as Inequality constraints in the optimization problem. Thus, the complete inequality matrices for N_{obs} can be re-written in the form of :

$$\begin{aligned} AO_{ineq} &= \begin{bmatrix} AO_1^T & AO_2^T & \dots & AO_{N_{obs}}^T \end{bmatrix}^T \\ bO_{ineq} &= \begin{bmatrix} bO_1^T & bO_2^T & \dots & bO_{N_{obs}}^T \end{bmatrix}^T \end{aligned} \tag{3.35}$$

Self-Collision Avoidance

Similarly, for the case of inequalities related to the self-collision avoidance it is necessary to define the positions of the obstacles. In this case, the obstacle position it is the joint position of the link from which a collision want to be avoided. Therefore, as mentioned in Section 3.4.3, if we define the positions of all joints $[p_{l_1}(k), ..., p_{l_6}(k)] = Tf(q(k))$ at every time k. Then, the distances between each pair of joints can be computed as :

$$\begin{aligned} d_{s1} &= \|p_{l_4}(k) - p_{l_1}(k)\|_2^2 & d_{s2} &= \|p_{l_3}(k) - p_{l_1}(k)\|_2^2 \\ d_{s3} &= \|p_{l_6}(k) - p_{l_1}(k)\|_2^2 & d_{s4} &= \|p_{l_5}(k) - p_{l_1}(k)\|_2^2 \\ d_{s5} &= \|p_{l_6}(k) - p_{l_2}(k)\|_2^2 & d_{s6} &= \|p_{l_5}(k) - p_{l_2}(k)\|_2^2 \end{aligned}$$

If we compact the terms into :

$$\bar{D}_s(q(k+1)) = \begin{bmatrix} d_{s1}^T & d_{s2}^T & \dots & d_{s6}^T \end{bmatrix}^T \in \mathbb{R}^6$$

And, correspondingly in the same order, the relationship for the computation of the unit vectors $\vec{n}_{s_i}$ and Jacobian functions should be maintained such as:

$$\Lambda_{J_s}(q_{k+1}) = diag(\begin{bmatrix} \vec{n}_{s_1}^T J_{p4} & \vec{n}_{s_2}^T J_{p3} & \vec{n}_{s_3}^T J_{p6} & \vec{n}_{s_4}^T J_{p5} & \vec{n}_{s_5}^T J_{p6} & \vec{n}_{s_6}^T J_{p5} \end{bmatrix})$$

Considering the same constant matrices $\bar{B}_1$ and $\bar{A}_1$, then it can be derived for only one step ahead prediction :

$$-\Lambda_{J_s}\bar{B}_1\mathbf{u}(\mathbf{k}) \leq \Psi_{s1}\bar{D}_s(q(k+1))) + \Lambda_{J_s}(q(k+1))\bar{A}_1\hat{z}(k+1) \tag{3.36}$$

Where :

$$\Psi_{s1} = \begin{bmatrix} \zeta_{s1} & 0 & \dots & 0 \\ 0 & \zeta_{s1} & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \zeta_{s1} \end{bmatrix}$$

Comparably, taking into account Equation (3.20) regarding the evolution of the joint positions, for M steps ahead we have:

$$-\Phi_{J_s}\mathcal{B}_1\mathbf{U} \leq \overline{\Psi}_s\mathcal{D}_s\Phi_{J_s} + \mathcal{A}_1\widetilde{Z} \tag{3.37}$$

Where :

$$\overline{\Psi}_s = \begin{bmatrix} \Psi_{s1} & 0 & \dots & 0 \\ 0 & \Psi_{s2} & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \Psi_{s6} \end{bmatrix} \qquad \Phi_{J_s} = \begin{bmatrix} \Lambda_{J_s}(\hat{q}_{k+1}) & 0 & \dots & 0 \\ 0 & \Lambda_{J_s}(\hat{q}_{k+2}) & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & \Lambda_{J_s}(\hat{q}_{k+M}) \end{bmatrix}$$

Analogously with the aim to relax the constraints, the convergence rate ζ_{s1} is defined with decreasing values trough the horizon. Finally, for the inclusion into the inequality constraints the notation can be simplified as:

$$\begin{aligned} AS_{ineq} &= -\Phi_{J_s}\mathcal{B}_1 \\ bS_{ineq} &= \overline{\Psi}_s\mathcal{D} + \Phi_{J_s}\mathcal{A}_1\widetilde{Z} \end{aligned}$$

Complete Inequality Constraints

Finally, the formalization of the inequality constraints stated in (3.11) regarding the motion limits, obstacle avoidance and self-collision avoidance it is presented:

$$\begin{aligned} A_{ineq} &= \begin{bmatrix} AL_{ineq}^T & AO_{ineq}^T & AS_{ineq}^T \end{bmatrix}^T \\ b_{ineq} &= \begin{bmatrix} bL_{ineq}^T & bO_{ineq}^T & bS_{ineq}^T \end{bmatrix}^T \end{aligned} \tag{3.38}$$

It should be noticed that the dimension of constraints related to the motion limits will increase accordingly to the prediction horizon, while the constraints related to obstacle avoidance function are dependent on the number of obstacles to detect and the prediction horizon. The efficiency of this technique resides in the computational effort required when more obstacles are needed to be included. Although this can increase the dimension of the inequality constraints, it does not compromise significantly the complexity of the optimization problem since the decision variables dimension is unchanged.

Problem Formalization

At this point, the formalization of the optimization problem in (3.10) as a Quadratic Programming problem can be stated as follows:

$$\min_U \underbrace{(\Phi_q^T \bar{S} \Phi_U + \Lambda_X^T \bar{W} \Gamma_U)^T}_{c^T} U + \frac{1}{2} U^T \underbrace{(\Gamma_U \bar{W} \Gamma_U + \Phi_U \bar{S} \Phi_U + \bar{R} + \Lambda_T^T \bar{R}_{\Delta \ddot{q}} \Lambda_T)}_{H} U \tag{3.39}$$

Subject to:

$$z(i+1|k) = Az(i|k) + Bu(i|k) \tag{3.40a}$$

$$q(i|k) = C_1 z(i|k) \tag{3.40b}$$

$$\dot{q}(i|k) = C_2 z(i|k) \tag{3.40c}$$

$$\underbrace{\begin{bmatrix} AL_{ineq}^T & AO_{ineq}^T & AS_{ineq}^T \end{bmatrix}^T}_{A_{ineq}} U(k) \leq \underbrace{\begin{bmatrix} bL_{ineq}^T & bO_{ineq}^T & bS_{ineq}^T \end{bmatrix}^T}_{b_{ineq}} \tag{3.40d}$$

3.4.6. Iterative Algorithm

In order to solve the optimization program in (3.10), as presented above, some preliminaries operations(acquisition of sensor measurements and computation of the closest distances to obstacle) must take place. Moreover, in order to integrate the receding horizon strategy, a numerical simulation of the system model is required as well as a recursive implementation of the overall Inverse Kinematics-Model Predictive Control. Therefore, a sequential iterative strategy is presented in Algorithm 3.7. Notice that the iterations are

related to the discrete time variable T_s which its numerical value will be defined in the subsequent sections.

Algorithm 3.7 Quadratic Programming - Inverse Kinematics

Input: X_D, $\dot{X}_D$, $q(0)$, $\dot{q}(0)$, $\ddot{q}(0)$, τ_{Li}
Output: $q(k)$, $\dot{q}(k)$
Procedure: $inverse\_kinematics(X_D, \dot{X}_D, q(0), \dot{q}(0), \ddot{q}(0), \tau_{Li})$

1: Compute the positions of each link given the current configuration $([p_{l_1}(k), ..., p_{l_6}(k)] \leftarrow Tf(q(k))$
2: Acquire the sensor measurement $\mathcal{S}(k_{Li}) \in \mathcal{P}(k_{Li})$ every τ_{Li} (See Algorithm 3.1).
3: Compute the closest distance from each joint to the obstacles: $p_{obs}(k_{Li}) = closest\_obstacles([p_{l_1}(k_{Li}), ..., p_{l_6}(k_{Li})], \mathcal{S}(k_{Li}), N_{km})$
4: Acquire the last evolution of the manipulated variables $[q(k), \dot{q}(k)]$ and the last optimal vector $U(k-1)$.
5: Solve the QP formulated in (3.39) subject to the inequality constraints(3.40d) and obtain $U(k)$;
6: Apply the first input $u(0|k)$ of the optimal sequence to the plant (3.5).
7: Set k = k+1 , go to 1.

3.5. Algorithm

In the present chapter, modular algorithms were developed and presented in order to build the proposed hierarchical architecture. In summary, in Algorithm 3.5, which refers to the Local Path Planner, the following Sub-algorithms 3.2 , 3.3 and 3.4 were integrated. Later, for Algorithm 3.7 the problem formulation of the IK-MP as a QP was developed, this included the integration of following Sub-algorithm 3.6. As stated, both main Algorithms are subjected to the LiDAR sensor readings that are simulated in Algorithm 3.1. A graphical scheme representing the interaction of the main Algorithms is deployed in Figure 3.14.

As it can be observed, the proposed structure allows the inclusion of other components such an offline or sequential high level planner which can compute the referenced target p_{target} based on the specific application, or a customized low level control system which can run at a different discrete time. This characteristic makes the presented architecture adaptable and capable to be extended with other motion or control techniques.

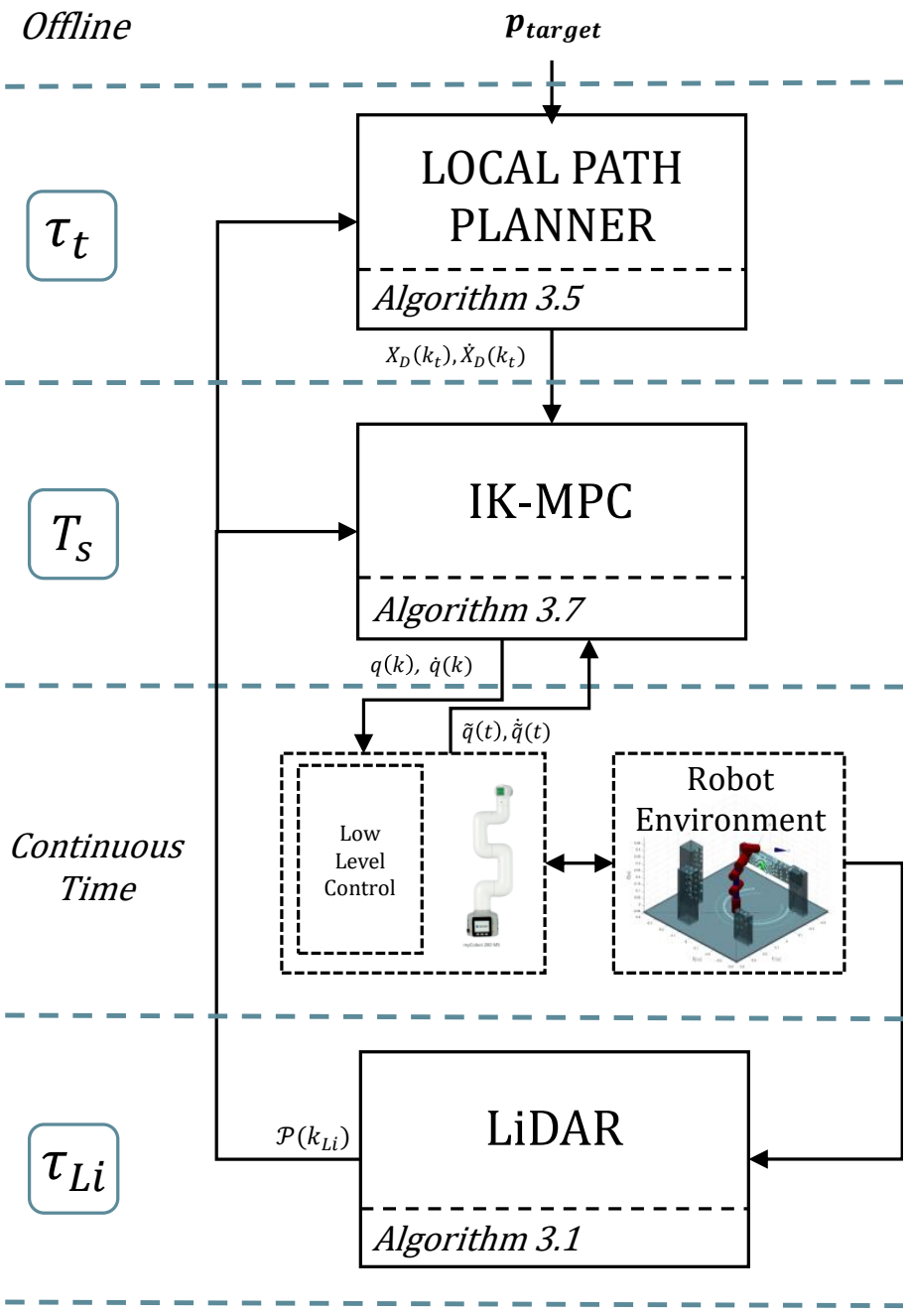


Figure 3.14: Block diagram scheme of the architecture

4 | Simulation results

4.1. Robotic arm simulation environment

The evaluation of the proposed method for a 6 degrees of freedom robotic arm in unknown environments provides insights into the effectiveness and versatility of the developed framework. The results are presented in three main sections, each of them focusing on a specific aspect of the presented system's architecture .

For the purpose of this study, there is no requirement to simulate a highly complicated structure. Instead, the preferred tests are focused in generate new environments and conduct rapid simulations of the kinematic robot behavior. In fact, with this aim certain assumptions should be stated:

- The low level controllers are assumed to be unitary gain with a sufficiently large gain margin.
- Ideal communication scenario, which does not takes into account any delays related to communication.
- The environment is limited to finite number of cuboid shape obstacles and the ground is defined to be at 5 cm under (Z-axis) the base frame of the robot.
- The LiDAR measurements are received every τ_{Li} seconds and it does not sense the body links of the manipulator.
- The trajectory is generated every τ_t seconds which means every time the robot performed the last partial trajectory fed.
- Only one target position p_{target} will be provided to the robot.

Therefore the system model used for the simulation is the one reported in Equation 3.5. For the purpose of a graphical representation of the results, several toolboxes will be used such as *Simulink-Simscape*[36](for dynamical motion simulation) and *PeterCorke Robotic Toolbox*[11](for a kinematic representation of the robot).

The simulations will be conducted under some initial conditions such as an initial configuration of the robot, fixed obstacles and a target position to reach. An example representation of an initial state of the system is shown in Figure 4.1, where the red star represented the target to reach p_{target} and the light-blue dots the LiDAR point-cloud.

Finally, the reported tests have been executed on a laptop equipped with an Intel core i7-11800H CPU, 16 GB of RAM , and MATLAB [28] R2023a version.

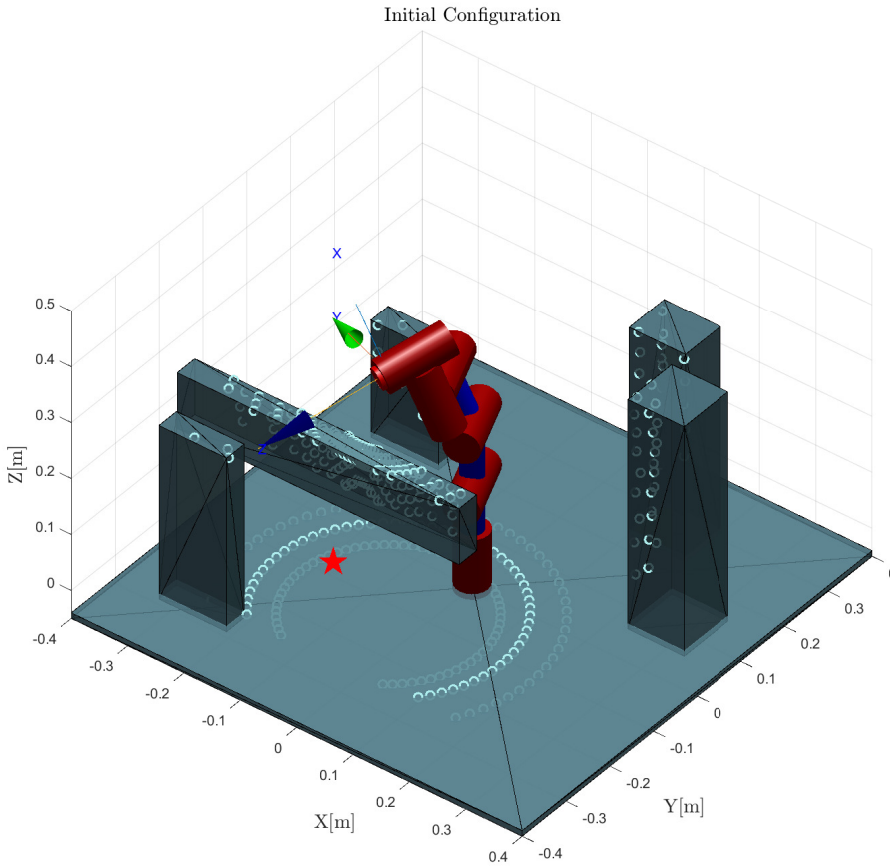


Figure 4.1: Example of Initial Configuration $q(0) = [\pi/8, 0, \pi/8, 0, -2.44, 0]$. Red-star: Target position p_{target}. Light-blue dots: Initial LiDAR readings.

4.2. Simulation Results

4.2.1. Trajectory Generation Results

This section delves into the trajectory generation component of the hierarchical control system. Various scenarios are explored to assess the execution time under different conditions. Additionally, the effectiveness of the obstacle clustering technique and the impact of the target shifting strategy, with variations in the parameter max_{cos}, are thoroughly analyzed. The outcomes provide a comprehensive understanding of the trajectory generation module's capabilities and limitations.

To this end, an initial case scenario is built and deployed in Figure 4.2. The clustered LiDAR readings are shown as well as the convex polytope of the obstacle free region in which the trajectory generation should take place. The large red dot refers to the target position and its line to the direct trajectory towards it. Additionally, the spheres represents the closest distances of each cluster to each link position.

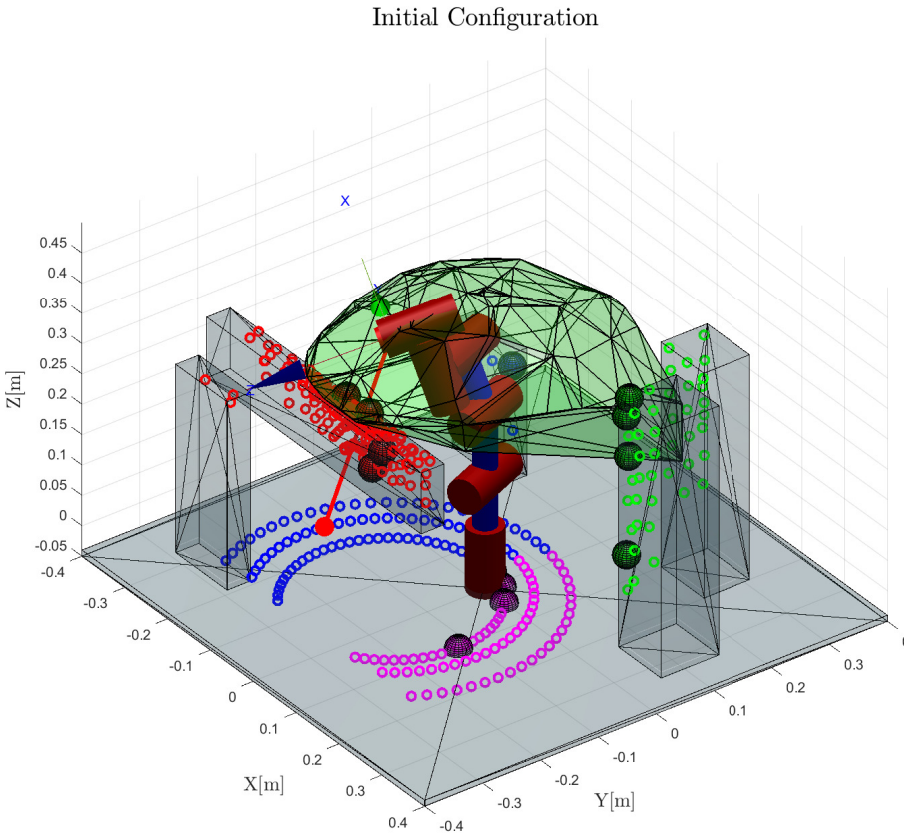


Figure 4.2: Trajectory Results: Initial Configuration. Light-green geometry: Obstacle free convex polytope. Circular dots: Represent each cluster of the sensor readings. Spheres: Closest distances of each cluster to each link position.

Thus considering this scenario, before computing a possible trajectory, the parameter max_{cos} related to shifting strategy should be analyzed. As mentioned before, this parameter is related on *how much* the target should be shifted away from the imminent obstacle. A representative analysis for different values of max_{cos} (0.2, 0.4, 0.6 and 0.8) is reported in Figure 4.3, where the blue stars represents the shifted targets and the parameter value. As it can be observed, lower values will produce a very shifted targets which, in some cases can be exploited to allow the robot for a reconfiguration. Indeed, this reconfiguration, as stated in Algorithm 3.5(line 10), is defined as a function dependent on whenever the manipulator tries to keep passing an obstacle but it always arrives closer to it.

It is reasonable to conclude that this value will be dependent on how limited is the free space of the robot, workspace and how close the manipulator could be around an obstacle. Without, any further assumption, for this work, an iterative experimental tunning resulted in an initial value for $max_{cos} = 0.8$ which showed a performance for the overall target shifting strategy.

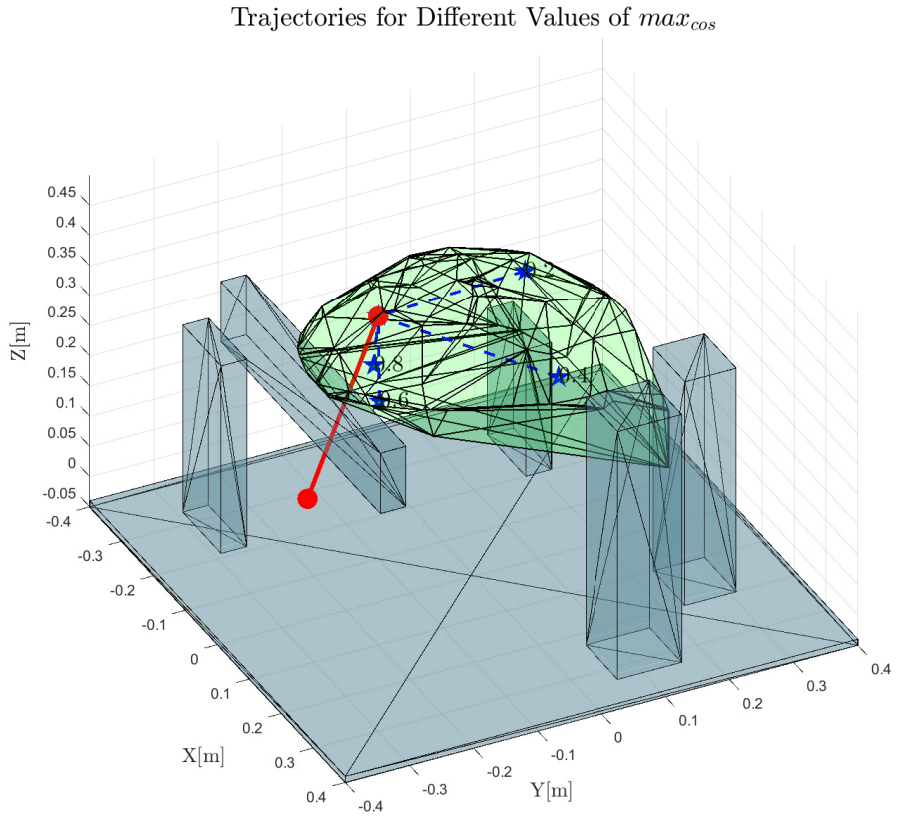


Figure 4.3: Blue-dashed: Possible trajectories for different values of max_{cos}. Blue-stars: Shifted target positions. Solid red: Direct trajectory towards the target.

For this initial configuration there is no need of reconfiguration of the robot, then for

a given settled parameter as $max_{cos} = 0.8$, a polynomial trajectory generation can take place as shown in Figure 4.4. The complete execution of Algorithm 3.5 reported a total execution time of **0.3695** seconds, with **0.1061** seconds for the Convex Polytope Generation and **0.2634** seconds for the Trajectory Generation and remaining arrangements. Additionally, the resultant polynomial trajectories position X_D and velocity $\dot{X}_D$ are shown in Figure 4.5. Although a general purpose library of MATLAB is being used for the trajectory generation, the time efficiency of the method is proven, with a possibility to be improved by personalized functions for the polynomial trajectory generation.

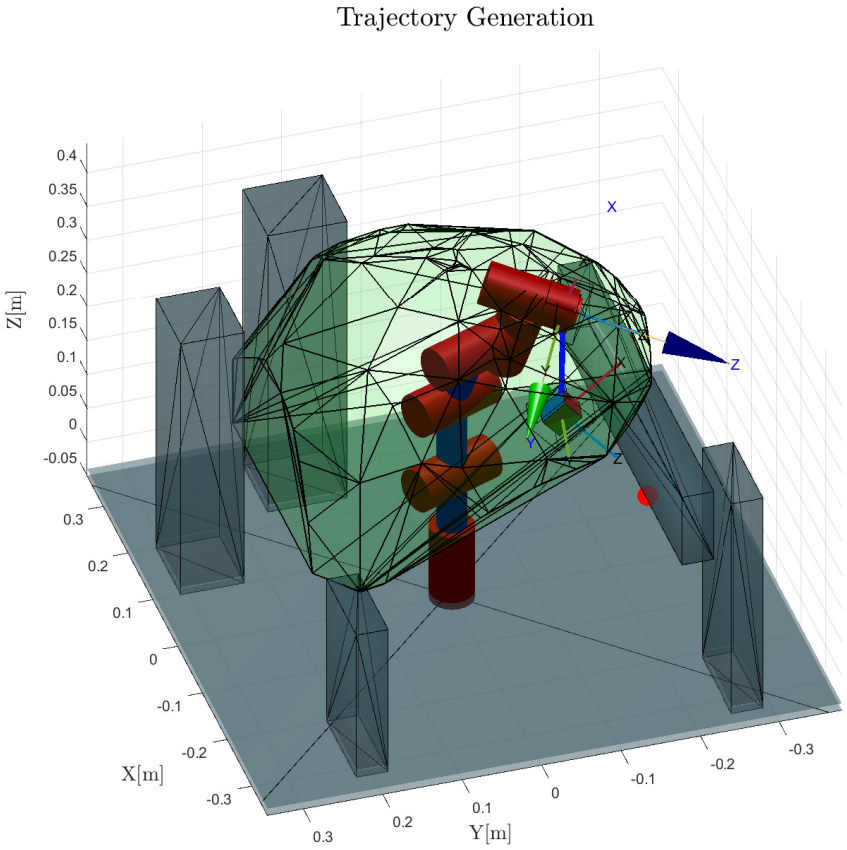
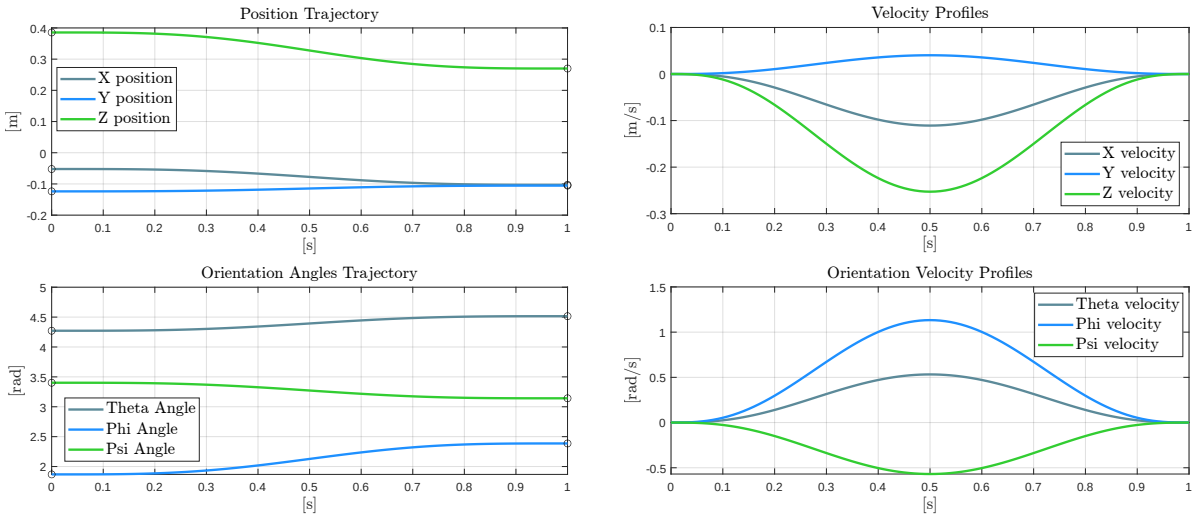


Figure 4.4: The polynomial trajectory generated $max_{cos} = 0.8$ towards the shifted target is represented in solid blue and the orientation axis(x, y, z) are represented by the red, green and blue lines centered at the initial and final positions.



(a) Positions. (b) Orientation.

Figure 4.5: Polynomial Trajectory Profiles.

4.2.2. IK- Optimization Results

In this section, the inverse kinematics optimization results are presented, comparing the standard one-step-ahead formulation with a multi-step-ahead approach. Metrics such as execution time, trajectory following accuracy (measured by Mean Squared Error), and the performance of optimization methods, specifically *interior-point* and *active-set*, are evaluated. This section aims to highlight the advantages of the proposed Inverse Kinematics-Model Predictive Control (IK-MPC) in dynamic scenarios.

Optimization Parameters

Recalling the QP optimization problem stated in (3.39), it is necessary to parameterize the weights and related coefficients in order to evaluate overall performance.

Regarding the weights considered in the cost function of the optimization routine, were adjusted in for trajectories without obstacles and updated through different simulation scenarios. Since the orientation of the end effector during the trajectory following it is less crucial that the position the matrix W and K were tuned accordingly. Moreover, to be consistent with the established limits of velocity and acceleration the matrices S and R were adjusted respectively together with the jerk penalization matrix $R_{\Delta\ddot{q}}$ to avoid drastic movements. Thereby, a final experimental tuning accounting for these effects led to the following values:

$$
\begin{aligned}
W &= diag(\begin{bmatrix} 5000 & 5000 & 5000 & 0.5 & 0.5 & 0.5 \end{bmatrix})1e12 \\
S &= diag(\begin{bmatrix} 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix})1e9 \\
R &= diag(\begin{bmatrix} 9 & 9 & 9 & 9 & 9 & 9 \end{bmatrix})1e8 \\
R_{\Delta\ddot{q}} &= diag(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix})1e-4
\end{aligned}
$$

The convergence rate of the error instead was chosen to maintain the stability of the Closed Loop [40] accordingly to the following relationship :

$$K < \frac{2}{T_s} \tag{4.1}$$

Where, $T_s = 0.01$ is the chosen discrete time period. Therefore, the following values for the position error K_p and orientation K_o coefficient were chosen:

$$
\begin{aligned}
K_p &= diag(\begin{bmatrix} 10 & 10 & 10 \end{bmatrix}) \\
K_o &= diag(\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}) \\
K &= diag(\begin{bmatrix} K_p & K_o \end{bmatrix})
\end{aligned}
$$

Additionally, for the case of the IK-MPC, as mentioned before (see Section 3.4.5), the convergence rate coefficient ζ in the obstacle avoidance function should be relaxed through the prediction horizon. This relaxation is denoted by a linear relationship as follows:

$$\zeta_i = \zeta_{init} - \frac{\zeta_{init} - \zeta_{end}}{M-1} i \ , \ \forall i = 1, ..., M \tag{4.2}$$

Thus, after a practical tunning the values were established as $\zeta_{init} = 0.5$ and $\zeta_{init} = 0.1$. The same values were also considered for the self-collision avoidance.

Lastly, since the problem on hand has only few inequality constraints(max 4 clusters) and a small dimension of the prediction horizon, the *active set method* was chosen as the preferred optimizer algorithm. The chosen algorithm was also particularly advantageous as it allowed to exploit the last solution found as a initial guess for the following iteration in the Inverse Kinematics algorithm, leading to faster convergence in subsequent optimizations and a substantial reduction in the overall computation. In contrast, the *interior-point method*, with its approach of navigating through the interior of the feasible region, seemed less influenced by the initial estimates. A comparison of the two methods is shown Table 4.2.

In fact, the MATLAB solver *quadprog* for QP optimization problems was employed and the values for the optimizer options were chosen based on performance required, these details are shown in Table 4.1.

Description	Value
Algorithm	Active Set
Optimality Tolerance	$1e^{-14}$
Step Tolerance	$1e^{-8}$
Constraint Tolerance	$5e^{-4}$
MaxIterations	200

Table 4.1: Optimization Parameters.

Simulation Results

In order to analyze the IK-MPC results, an offline generated circular trajectory with an intermediate point was fed as a reference with the aim to compare the results for different horizons and optimization algorithms. As shown in Figure 4.6, the robot is positioned in a starting configuration $q(0) = [\pi/8, 0, \pi/8, 0, -2.4435, 0]$, the blue line represents selected circular trajectory which first reaches a target position $p_{target} = (-0.1522, -0.2238, 0.2000)$ and the returns to the initial position $p_{init} = (-0.0522, -0.1238, 0.3855)$ with a constant orientation over the whole trajectory.

Given that the trajectory tracking is very accurate for all the cases which is difficult to appreciate graphically, the results will be measured considering the Mean Squared Error(MSE) between the reference trajectory and the performed trajectory for different Prediction Horizons. The results are reported in Table 4.2. From the table can be inferred that the MSE is improved when the Prediction Horizon is longer but that implies a trade-off with a higher execution time per iteration. Between both algorithms, as expected, the *Active-Set* had better results in terms of execution time while achieving the same MSE as the *Interior-Point* for all horizons. Additionally, the MSE improvement between the 1-step and 5-steps horizon are significant, but not the case from the 5-steps to 10-steps which also implies a higher computational effort (almost the double per iteration). Furthermore, although both methods demonstrated their capabilities to solve the problem within the discretization time $T_s = 0.01$s, the reported average time per iteration of the *Active-Set* algorithm exhibited a better performance validating its feasibility to be implemented with a sampling time as lower as $1ms$. Therefore, given that the proposed method is meant to be computationally efficient, from now on, the chosen method for testing and results will be **Active-Set** with a **5-steps** ahead prediction horizon.

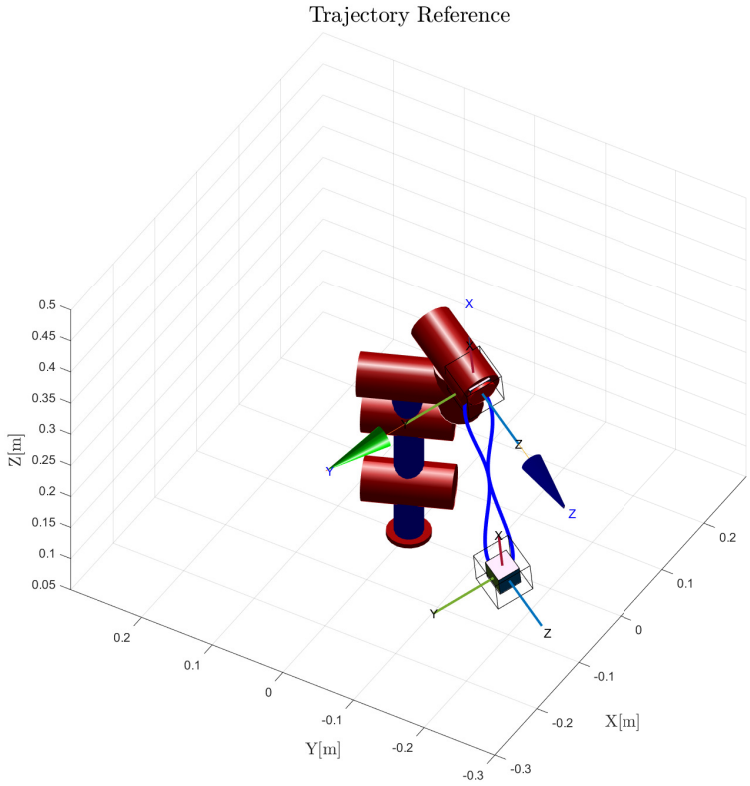


Figure 4.6: Circular Trajectory Reference. The trajectory is represented by the solid blue line and the orientation axis(x, y, z) are represented by the red, green and blue lines centered at the initial and final position

Moreover, the choice of the prediction horizon is coherent with a good practice since for a sampling time of $T_s = 0.01$s it will cover 0.05s which given the small changes between iterations(in the order of $10^{-2}[rad]$) for a common high-gain lower level controller it should be sufficient to cover the entire settling time.

The resultant joint positions, velocities and acceleration of the performed trajectory as a result of the IK-MPC are shown in Figure 4.7. As observed, the joint velocity and acceleration vectors are characterized by smooth motions without discontinuities, and as a consequence a smooth trajectory.

Table 4.2: Task Space Results.

	Active-Set			***Interior-Point***		
Prediction Horizon	1-step	5-steps	10-steps	1-step	5-steps	10-steps
MSE $x_{EE}[10^{-9}]$	2.8373	1.5524	1.5668	2.8373	1.5524	1.5668
MSE $y_{EE}[10^{-9}]$	8.8543	1.4406	1.4383	8.8543	1.4406	1.4383
MSE $z_{EE}[10^{-9}]$	6.4671	3.5911	3.6316	6.4671	3.5911	3.6316
MSE $\phi_{EE}[10^{-9}]$	6.8249	4.5419	4.6066	6.8249	4.5419	4.6066
MSE $\theta_{EE}[10^{-9}]$	9.5563	7.3995	7.6011	9.5563	7.3995	7.6011
MSE $\psi_{EE}[10^{-9}]$	6.7244	3.2923	3.1278	6.7244	3.2923	3.1278
Total Exec. Time $[s]$	0.3858	0.4486	0.8844	1.7843	3.8132	8.0320
Avg. Time per Iter. $[ms]$	0.3870	0.4518	0.8951	1.7897	3.8401	8.1296

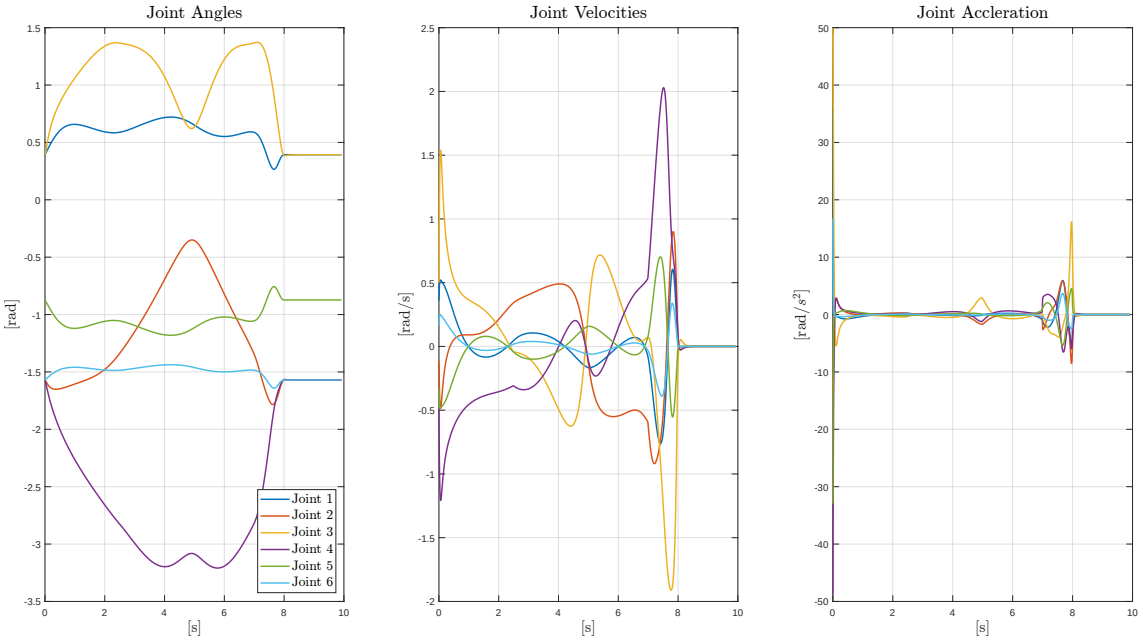


Figure 4.7: Joints Motion. Algorithm: *Active-set*. Prediction horizon: $M = 5$.

Now, the system is able to be tested with obstacles obstructing the generated trajectory. To this end, a linear polynomial trajectory was generated to the same target(returning to the same position) with a sphere shape obstacle in the middle of the path (see Figure 4.8). The resultant performed trajectory is presented in 3D-space in Figure 4.9, while the task space(position and orientation) results with the evolution of the constraint(distance to obstacle) are shown in Figure 4.10.

Additionally, the joint motions are presented in Figure 4.11, where the most oscillating behaviour can be observed when the robot is avoiding the obstacle.

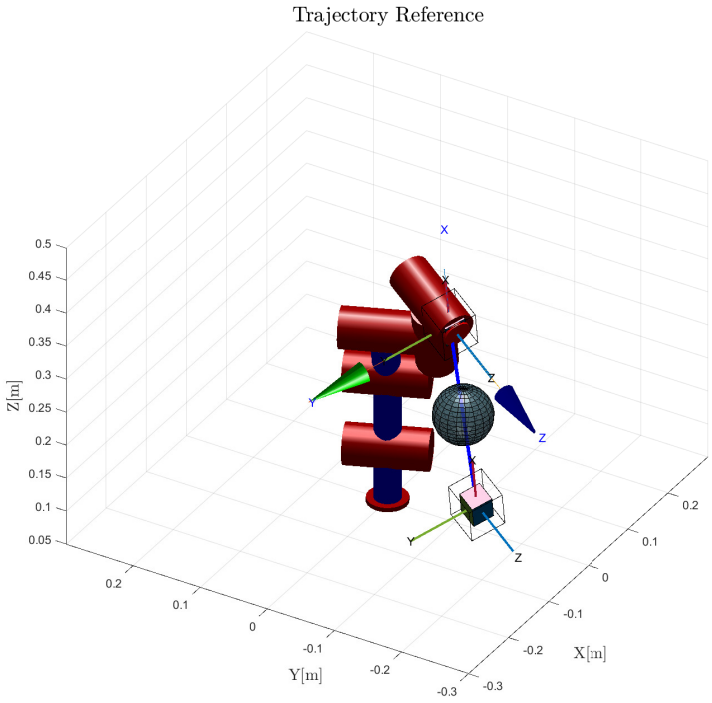


Figure 4.8: Trajectory reference with obstacle. The trajectory is represented by the solid blue line and the orientation axis(x, y, z) are represented by the red, green and blue lines centered at the initial and final position.

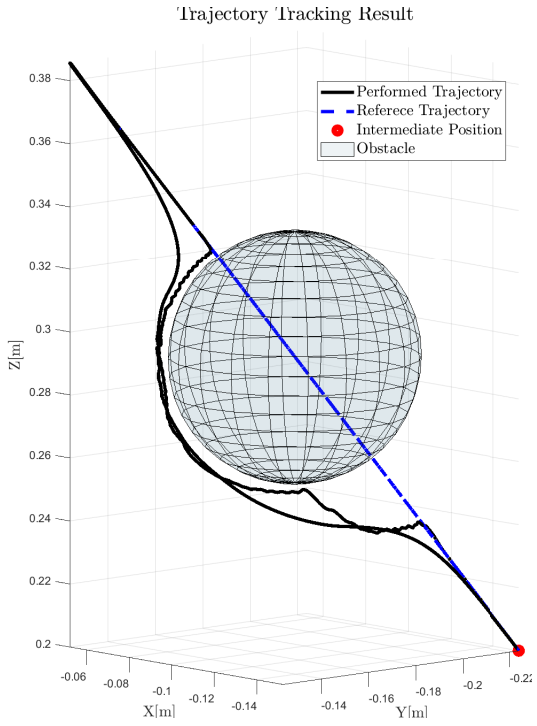


Figure 4.9: 3D Trajectory Tracking Results

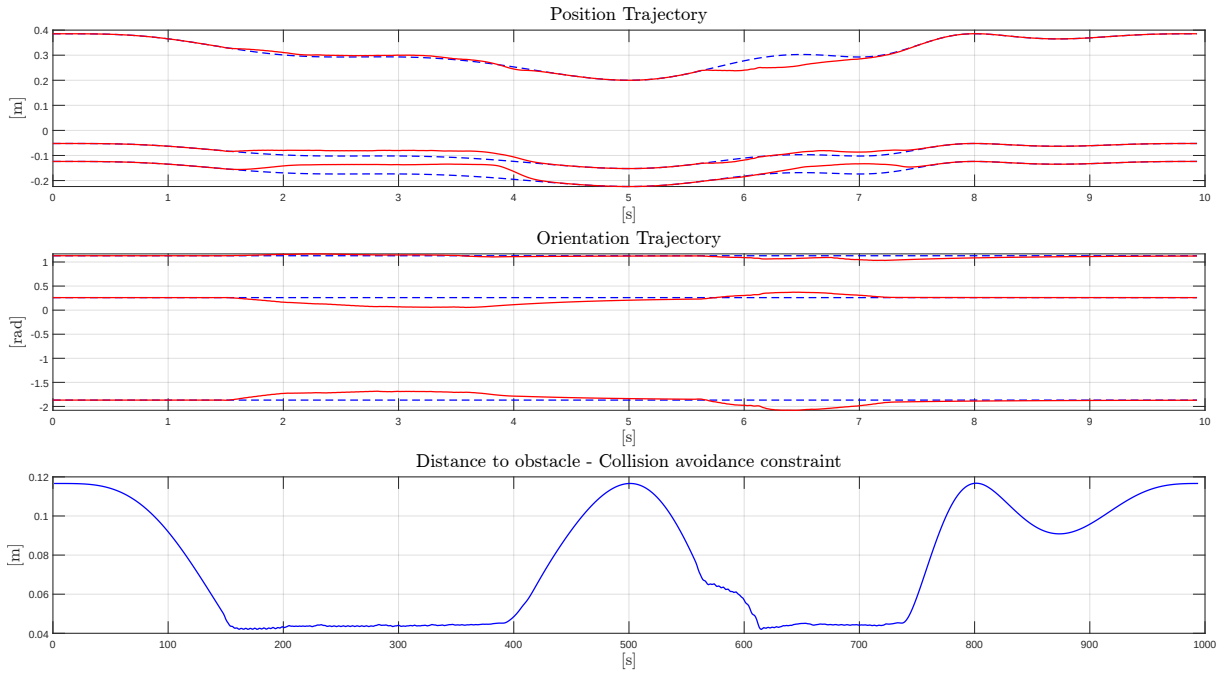


Figure 4.10: Performed Trajectory(solid red), Generated Trajectory(blue dotted) and Distance to obstacle constraint(solid blue).

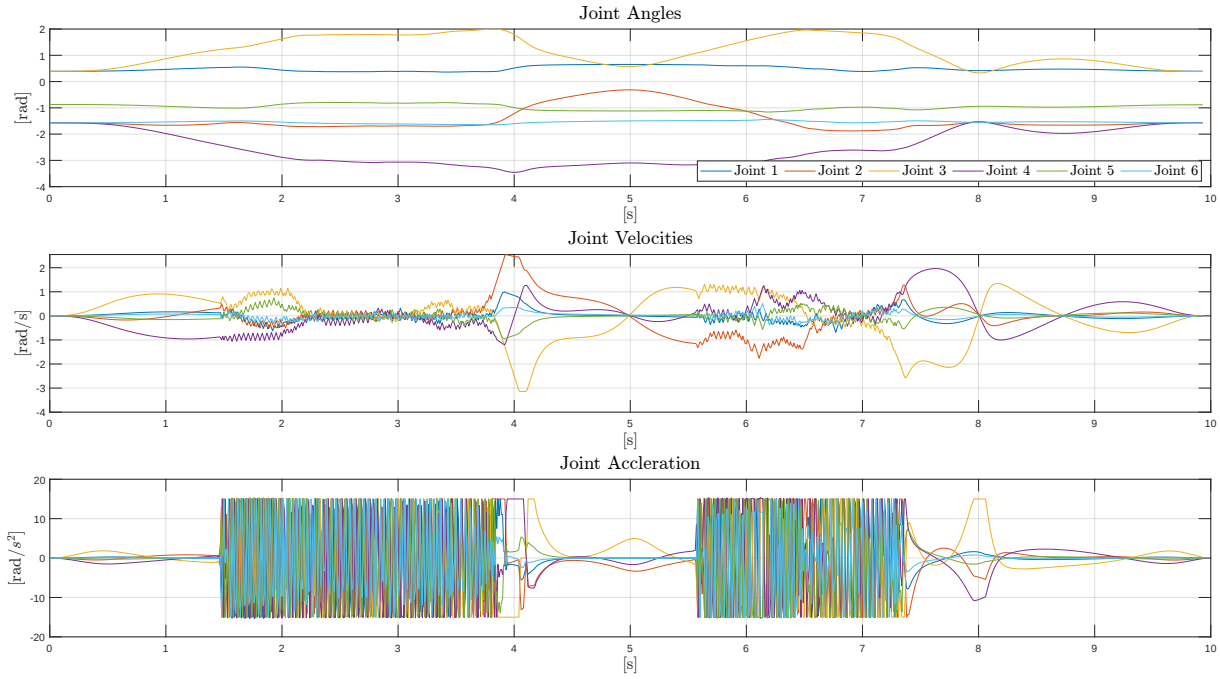


Figure 4.11: Trajectory with obstacle: Joints Motion.

4.2.3. Overall Framework Results

To this point, both algorithms, Trajectory Generation and IK-MPC, are shown independently a satisfactory performance according to what was required in the problem formulation. Now it is possible to test the coupled performance of the entire architecture in challenging environments; thus, in this section, the overall performance of the developed system is assessed through two distinct scenarios simulating real-world robotic applications. Each scenario represents a complex environment, challenging the robotic arm to navigate and execute tasks while respecting the physical constraints and obstacle avoidance functions. The use of a Digital Twin of a MyCobot280 Robotic Arm in the simulation, was implemented using Simulink, Matlab, and Peter Corke's Robotic Toolbox, this ensures a realistic representation of the system's kinematic behavior. The results provide insights into the system's robustness, safety, and efficiency in accomplishing tasks within dynamic and unpredictable conditions. Although these scenarios were selected for its complexity, additional simpler tests were performed as reported in Appendix A. In all cases, the manipulator will try to reach a target position p_{target} (with no direct visibility of the target) and then return to its initial position p_{init}. Although it is reasonable that the solution to reach the target position could be exploited to return to the initial position, it will not be the case for this set-up. All with the aim to show the capabilities of the proposed method even when robot starts moving(when returning) in a cluttered environment.

As outlined in the preceding sections, it is necessary to provide a summary of fixed parameters for simulating the algorithms, these are reported in Table 4.3 (for the LiDAR configuration see Table 3.1).

Description	Value	Related Algorithm
$\bar{\varepsilon}$	0.4	$temp\_target()$
max_{cos}	0.7	$temp\_target()$
TOL_{sc}	0.99	$temp\_target()$
D_{tot}	0.05	Trajectory Generation
t_f	1 s	Trajectory Generation
$\bar{d}_i$	0.05	$inverse\_kinematics()$
$\bar{d}_s$	0.04	$inverse\_kinematics()$
τ_{Li}	0.5 s	General
T_s	0.01 s	General
N_{obs}	4	General

Table 4.3: Simulation Algorithm Parameters.

Finally, the 3D representation of every figure in this section will obey the following con-

vention : Light-blue dots LiDAR readings at time k_t, the light-green polytope the obstacle free region at time k_t,the red stars represents the initial and target positions, the blue dotted line are the initial and final Z orientation of the end effector, the cyan triangles represents the shifted target position, the blue continue line the trajectory generated and the black line (triangle markers) is the final performed trajectory .

First Scenario

This first scenario aims to challenge the robotic manipulator with a common task such as "picking an object under a table". The environment will be also characterized with two additional aleatory obstacles generated within the workspace apart from the table. Although the task of "picking" is out of the scope of this work, the automatic motion of robot will try to mimic this situation by reaching a target position starting from a random configuration. The initial state of the robot is reported in Table 4.4 and graphically represented in Figure 4.12.

Description	Value
Initial Configuration [rad]	$[\pi/2, 0, \pi/8, 0, -2.443, 0]$
Initial Position [m]	(0.0944,-0.0956 ,0.3855)
Target Position [m]	(-0.0056,-0.2156, 0.1500)

Table 4.4: First Scenario: Initial Configuration.

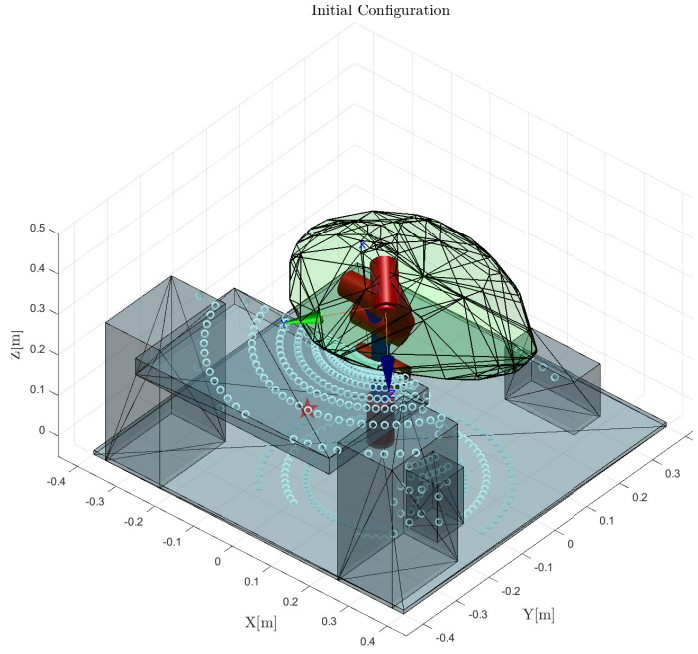


Figure 4.12: First Scenario: Initial Configuration

The results of the aforementioned scenario will be analyzed iteratively to understand every step made by the system in specific situations. First, (see Figure 4.13) since there is no direct visibility, the robot system starts by shifting its trajectory towards an obstacle free direction by reconfiguring its position, then a second trajectory is generated but closer to the target. Consequently (see Figure 4.14), a third trajectory is generated but not with direct visibility to the target because condition in line 6 - Algorithm 3.3 was not accomplished. Finally(4th iteration), this condition was accomplished a last trajectory towards the target was generated. Subsequently, the target was shifted towards the initial position, and for the first iteration(retraction) the orientation was maintained.

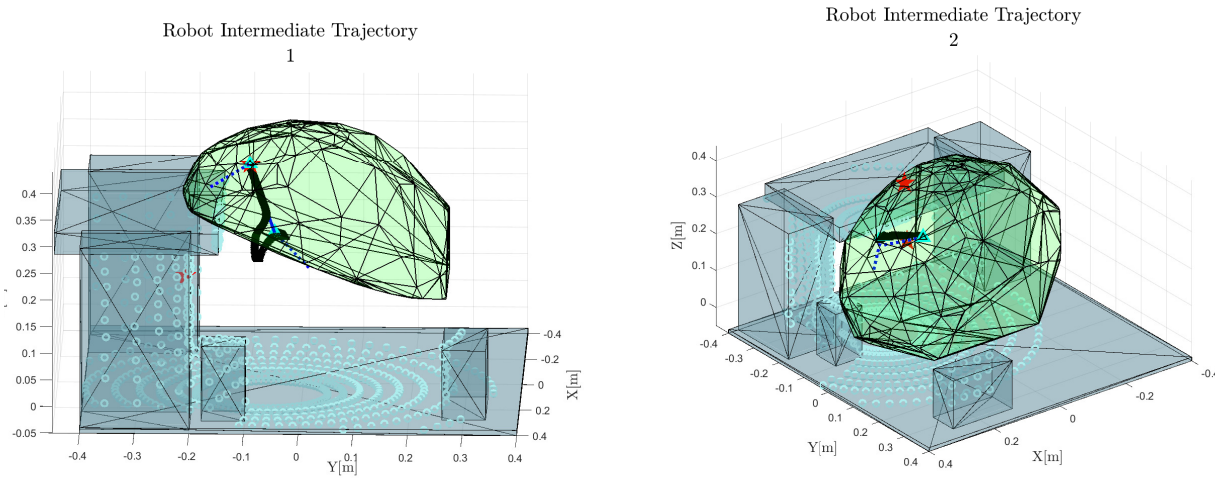


Figure 4.13: Intermediate Trajectories 1 and 2

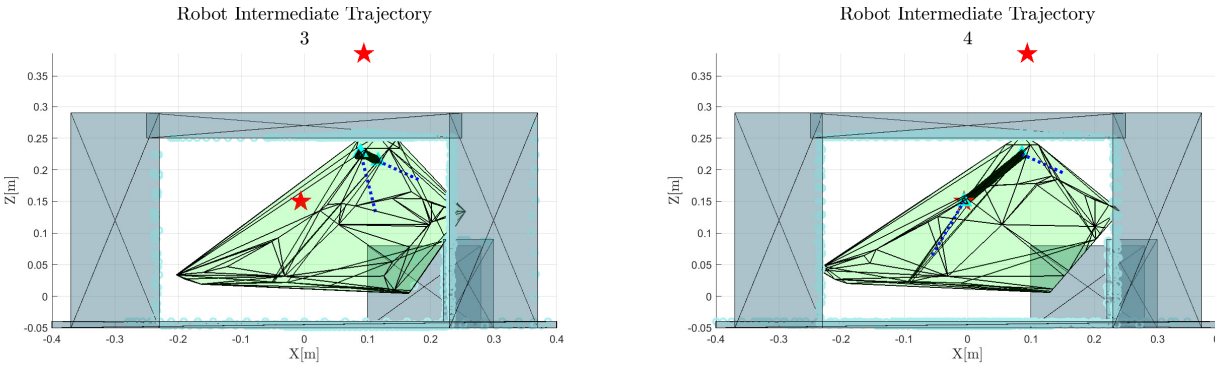


Figure 4.14: Intermediate Trajectories 3 and 4

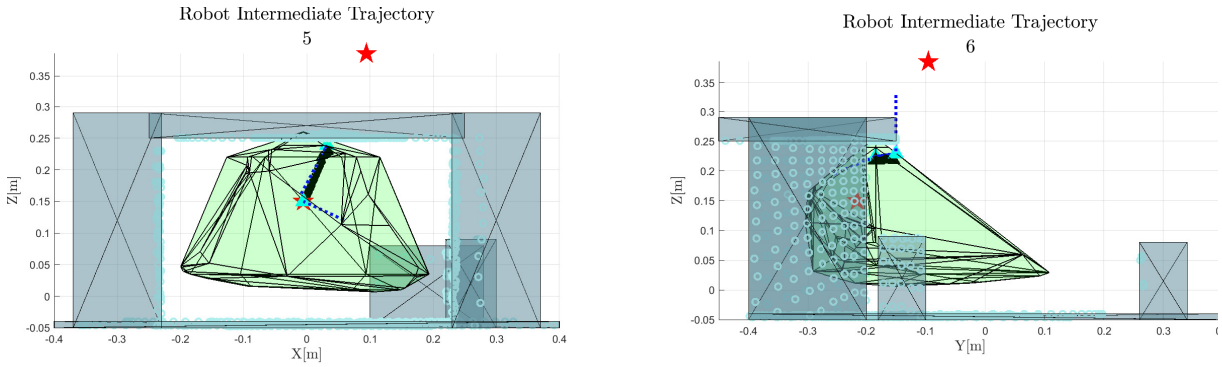


Figure 4.15: Intermediate Trajectories 5 and 6

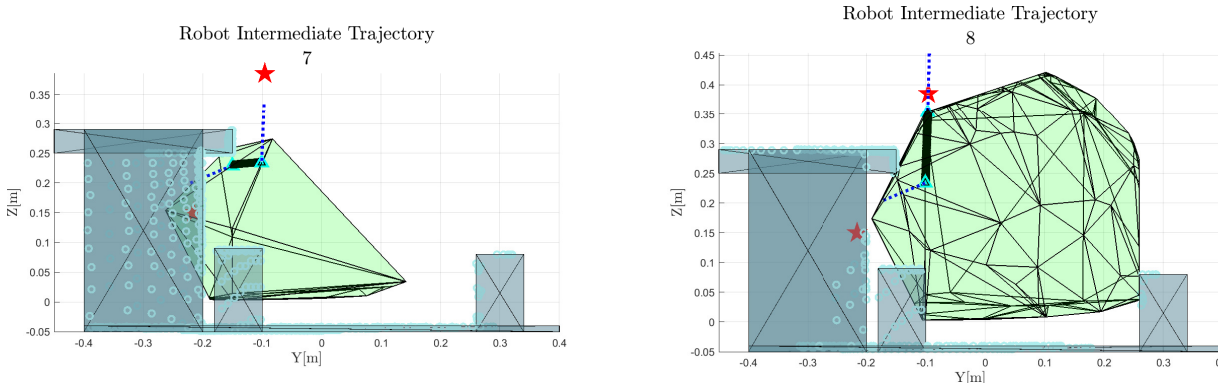


Figure 4.16: Intermediate Trajectories 7 and 8

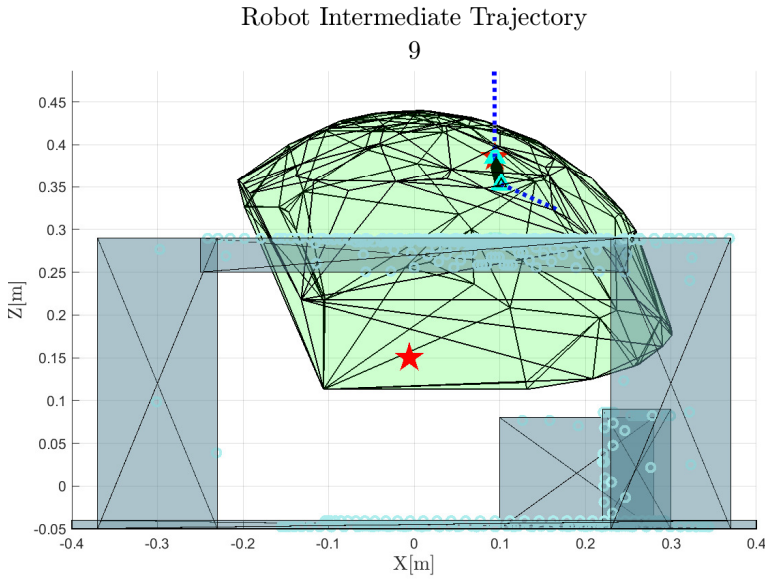


Figure 4.17: Intermediate Trajectories 9

The complete 3D trajectory performed by the manipulator is shown in Figure 4.18 while a detailed task space trajectory tracking is shown in Figure 4.19. It can be observed that the reference orientation is not tracked as much as the position, this was expected since the cost function weights were focused most on the positioning part.

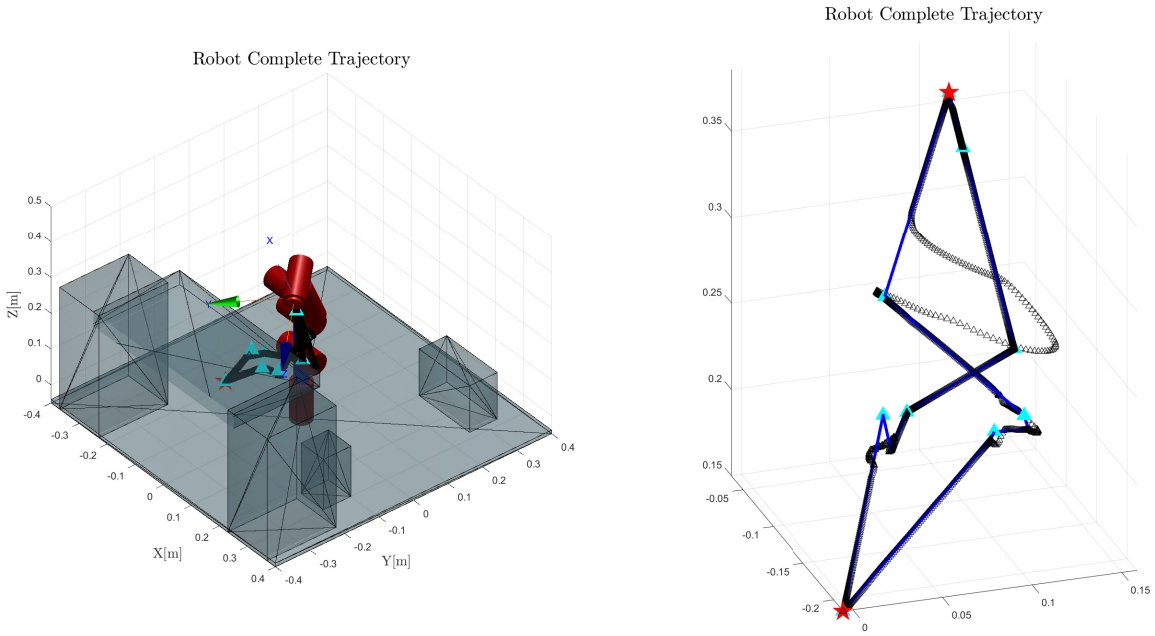


Figure 4.18: First Scenario: Complete Performed Trajectory.Red-Stars: Initial and target positions. Black triangle-dashed line: Trajectory performed. Blue-line: intermediate trajectories. Cyan triangles: Intermediate Shifted targets.

Consequently, the resultant joint positions, velocities and acceleration of the performed trajectory are reported in Figure 4.20

Finally, the distances related to the obstacle avoidance constraint are reported in Figure 4.21 while the ones related to the self-collision constraint in Figure 4.22. It should be noticed that the new position of the obstacles is updated every $\tau_{Li} = 0.5$ s, which leads in some cases to a sudden violation of the obstacle constraints, therefore, the control input(acceleration) saturates trying to return to the feasible region.

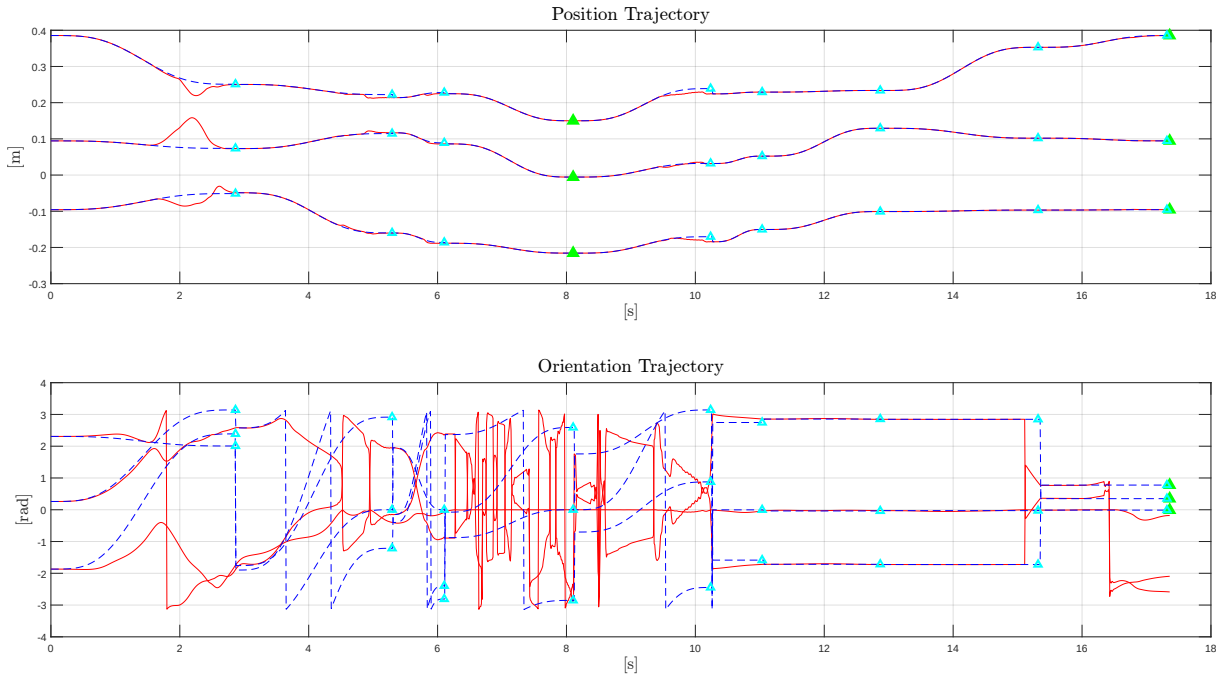


Figure 4.19: Performed Trajectory(solid red) and Generated Trajectory(blue dashed).

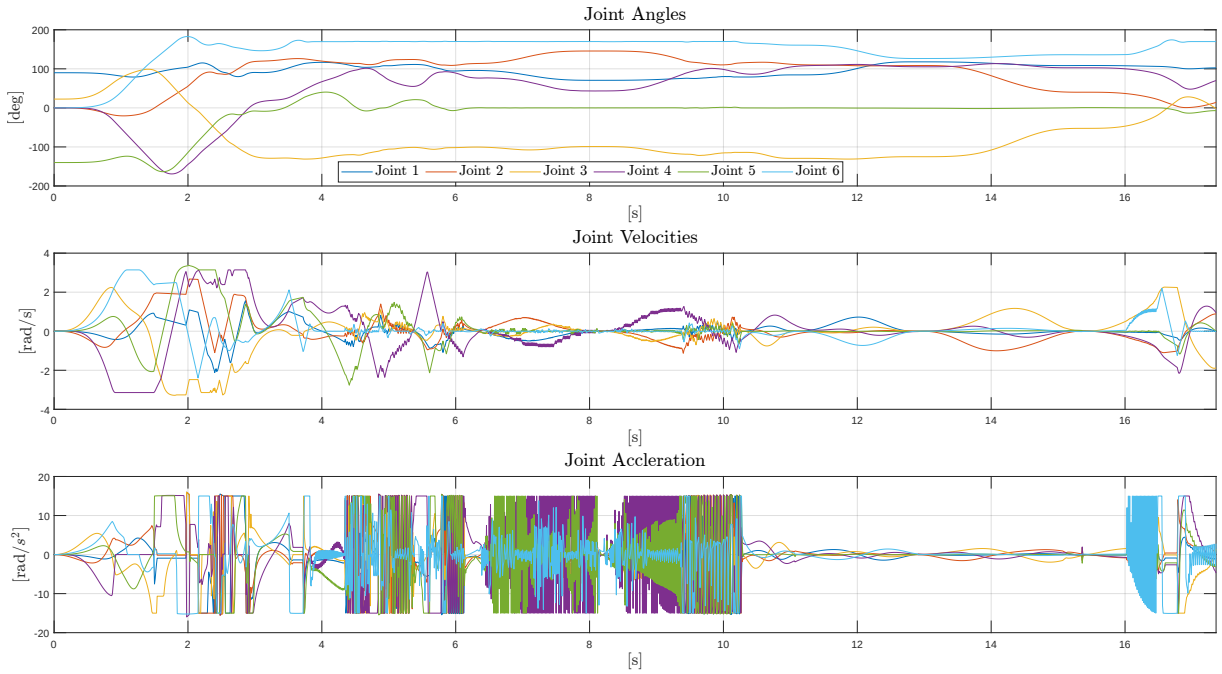


Figure 4.20: Joints Motion

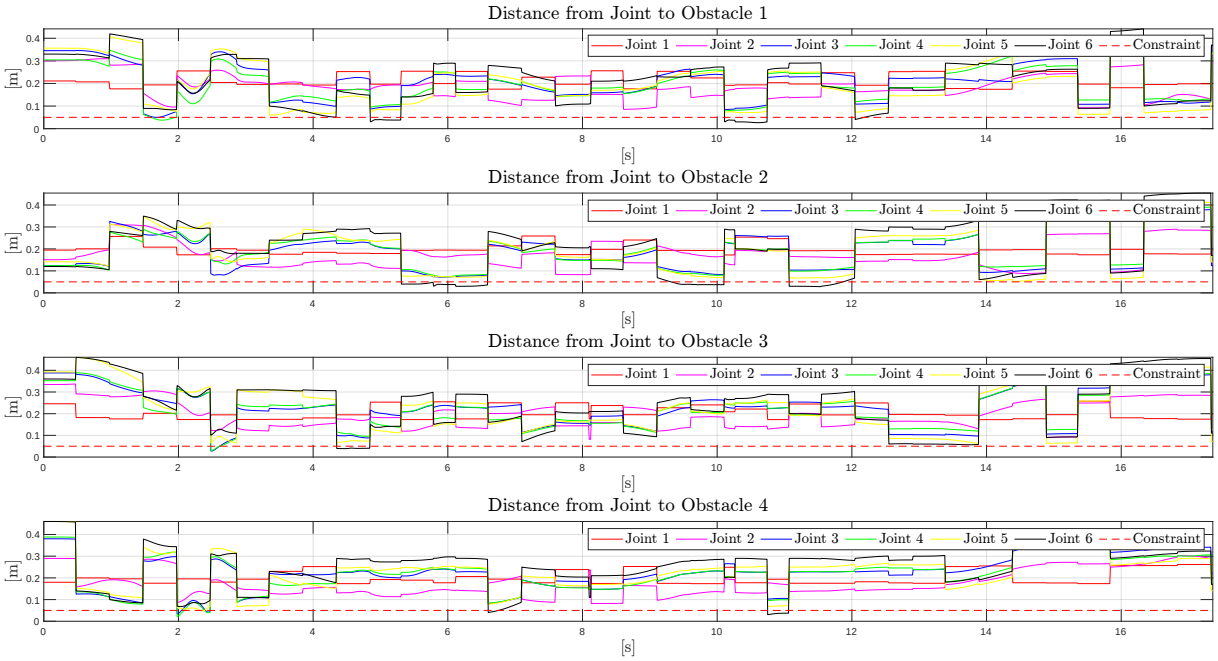


Figure 4.21: Distance from each link to each obstacle

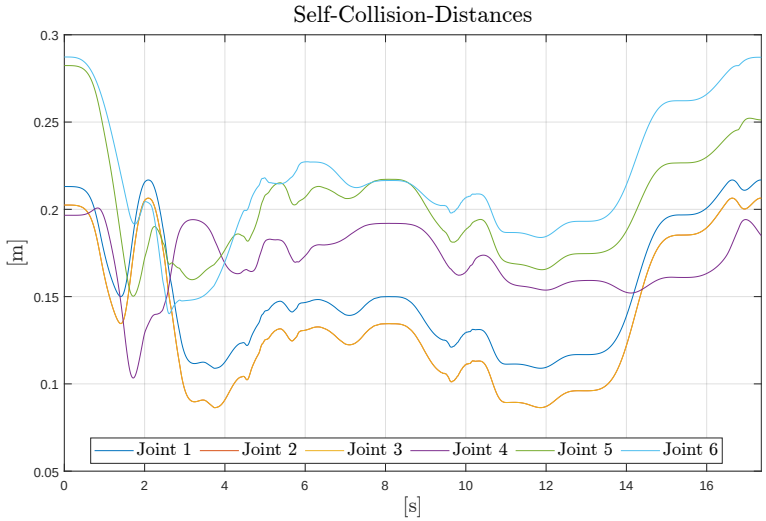


Figure 4.22: Self-Collision Distances. Constraint at 0.045 m

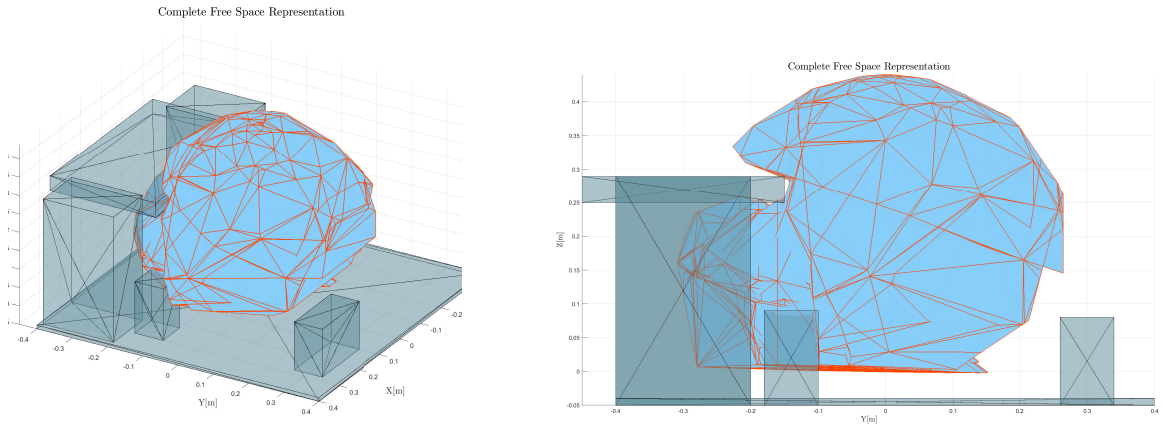
Additionally, as know, each trajectory had a different time-scaling, a time allocated to generate the obstacle free polytope and generate the trajectory, and a time to solve the IK. Those values are reported in Table 4.5 respectively. From the table can be concluded that the collected time results show the efficiency of the method with a lower than $2ms$ for the IK time period. This shows the capability of the method to be used for an online implementation since the IK is solved much faster than the discrete time T_s. While a maximum of $1s$ for the trajectory generation, trajectory that can be computed in parallel

while the robot is still reaching its target.

Iteration	Traj. Time $[s]$	Traj. Gen. Time $[s]$	Iter. Avg. Time IK-MPC $[s]$
1	2.8641	1.0055	1.5326×10^{-3}
2	2.4302	0.4278	0.7157×10^{-3}
3	0.8092	0.1409	0.9011×10^{-3}
4	2.0000	0.1742	0.5413×10^{-3}
5	2.1329	0.3018	0.6899×10^{-3}
6	0.7996	0.1303	0.9900×10^{-3}
7	1.8325	0.1863	0.4813×10^{-3}
8	2.4488	0.8454	0.4748×10^{-3}
9	2.0000	0.1931	0.4990×10^{-3}

Table 4.5: First Scenario: Time Execution Results.

Moreover, although any map generation was included in this work, it is essential to mention the possibilities of generating an obstacle free local map while the robot is reaching the target. This feature can be exploited by means of the the union of the intermediate convex polytopes generated, for this example, graphically this is represented in Figure 4.23.



(a) Isometric view.

(b) Side view.

Figure 4.23: First Scenario: Complete Obstacle Free Region.

Second Scenario

This second scenario aims to challenge the robotic manipulator in a highly cluttered environment with a target close and between obstacles. The environment will be also characterized with fourteen additional aleatory obstacles generated within the workspace. This scenario is particularly challenging since the goal target is laying between obstacles and in positioned backwards, situation which will require the robot to reconfigure without colliding with itself. The initial state of the robot is reported in Table 4.6 and graphically represented in Figure 4.24.

Description	Value
Initial Configuration [rad]	$[1.396, 0, \pi/8, 0, -\pi/2, 0]$
Initial Position [m]	(0.0971,-0.0829 ,0.3983)
Target Position [m]	(-0.1861,0.2109, 0.1500)

Table 4.6: Second Scenario: Initial Configuration.

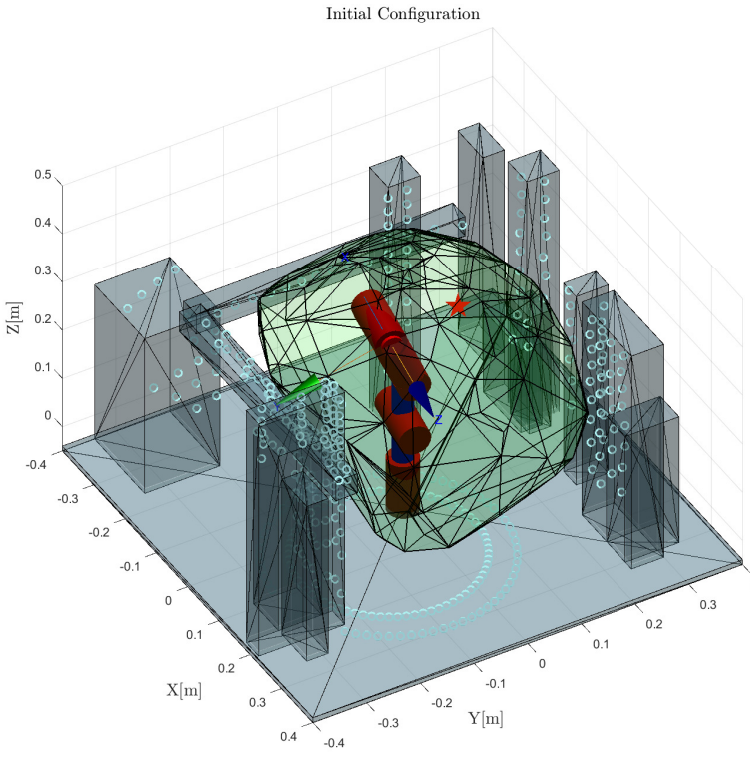


Figure 4.24: Second Scenario: Initial Configuration

First (see Figure 4.25), a trajectory with orientation towards the target is generated. Since the target position is behind the robot this trajectory involves a reconfiguration of the manipulator which took place during the performed trajectory. Then (right image),

a second trajectory is generated towards a obstacle free direction, nevertheless, this trajectory was unfeasible since an approximation of the robot will cause a collision with the closer obstacle. Therefore, the robot IK-MPC anticipated the collision and performed a different trajectory but still closer to the temporary target.

Finally (see Figure 4.26), the manipulator reached the target while respecting the obstacle avoidance constraints. Followed by a retracting stage to return to its initial position (also Figure 4.27).

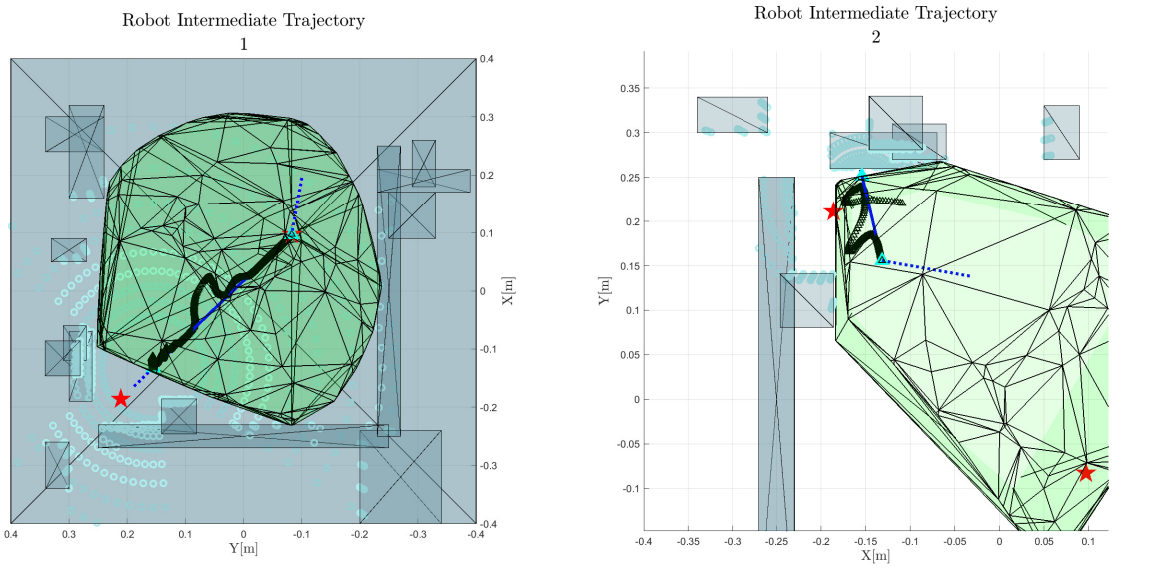


Figure 4.25: Intermediate Trajectories 1 and 2

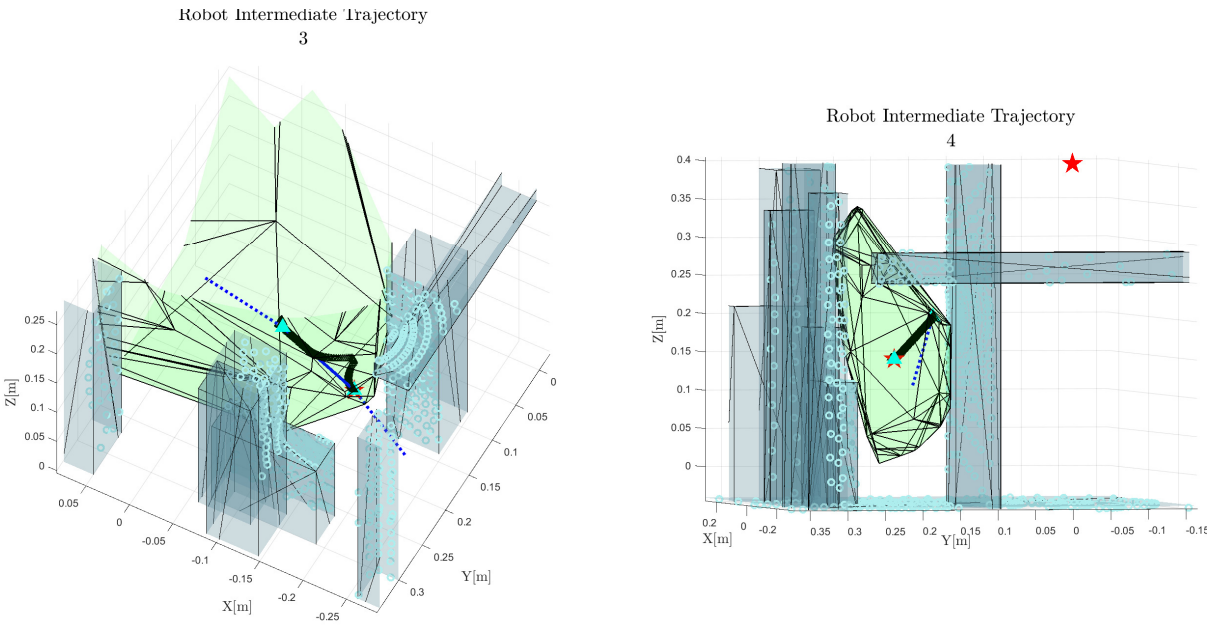


Figure 4.26: Intermediate Trajectories 3 and 4

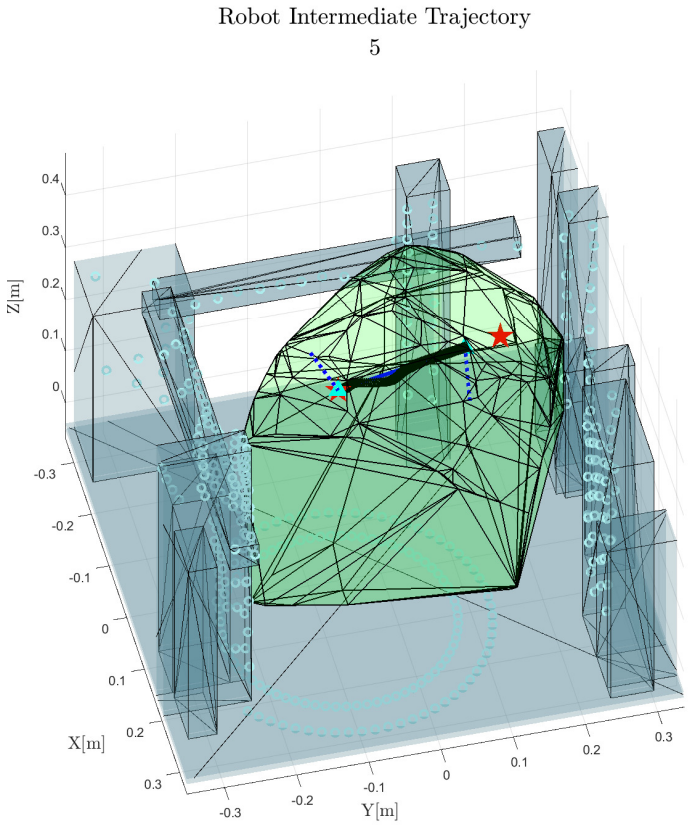


Figure 4.27: Intermediate Trajectory 5

The entire 3D trajectory performed by the manipulator is shown in Figure 4.28 while a detailed task space trajectory tracking is shown in Figure 4.29. As expected, the robot deviates from its referenced trajectory to reconfigure and point towards the objective. Additionally, when the end effector its getting closer to the obstacles, the LiDAR readings are updating the obstacles positions which led to a sudden violation of the constraints as shown in Figure 4.31, in those time instants the control input saturates by trying to go back to the feasible region as it can be observed in the acceleration behaviour in Figure 4.30. In the other hand, despite the fact that the robot needed to reconfigure, it performed without violating the self-collision constraint as presented in Figure 4.32.

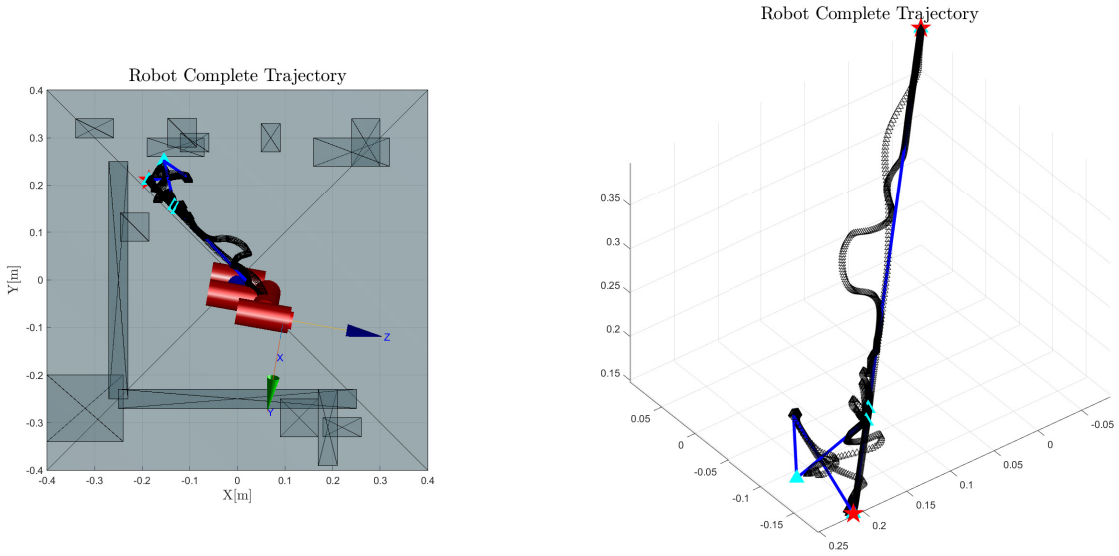


Figure 4.28: Second Scenario: Complete Performed Trajectory. Black triangle-dashed line: Trajectory performed. Blue-line: intermediate trajectories. Cyan triangles: Intermediate Shifted targets.

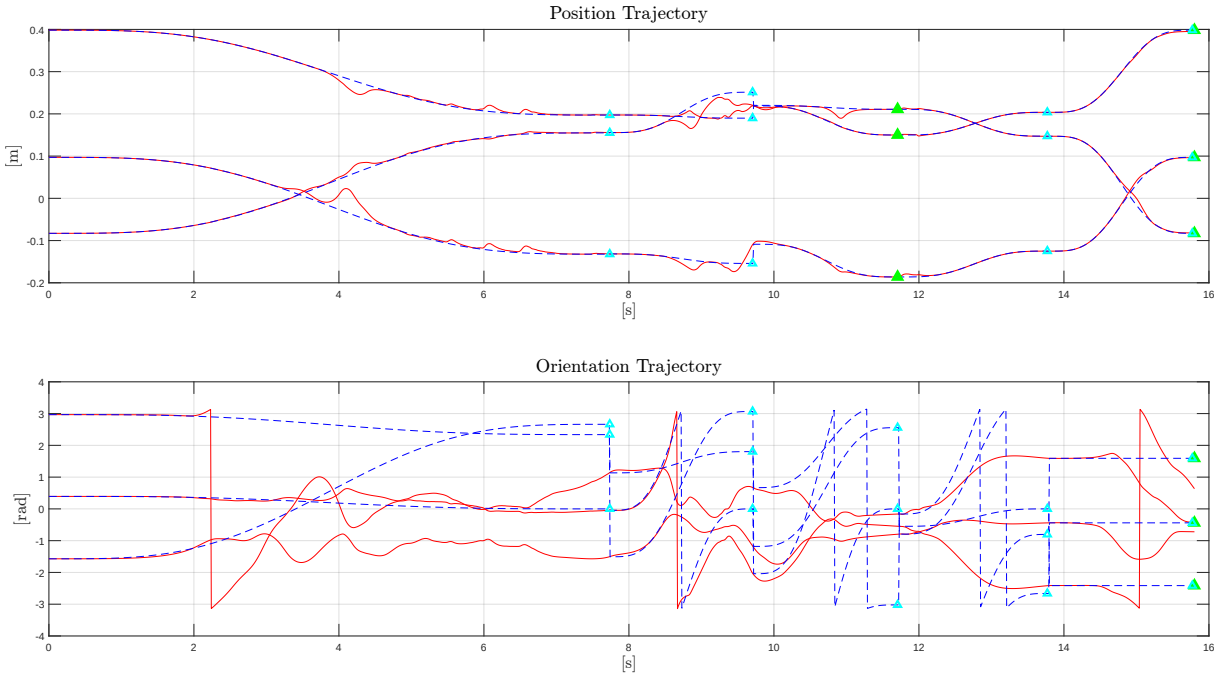


Figure 4.29: Performed Trajectory(solid red) and Generated Trajectory(blue dashed).

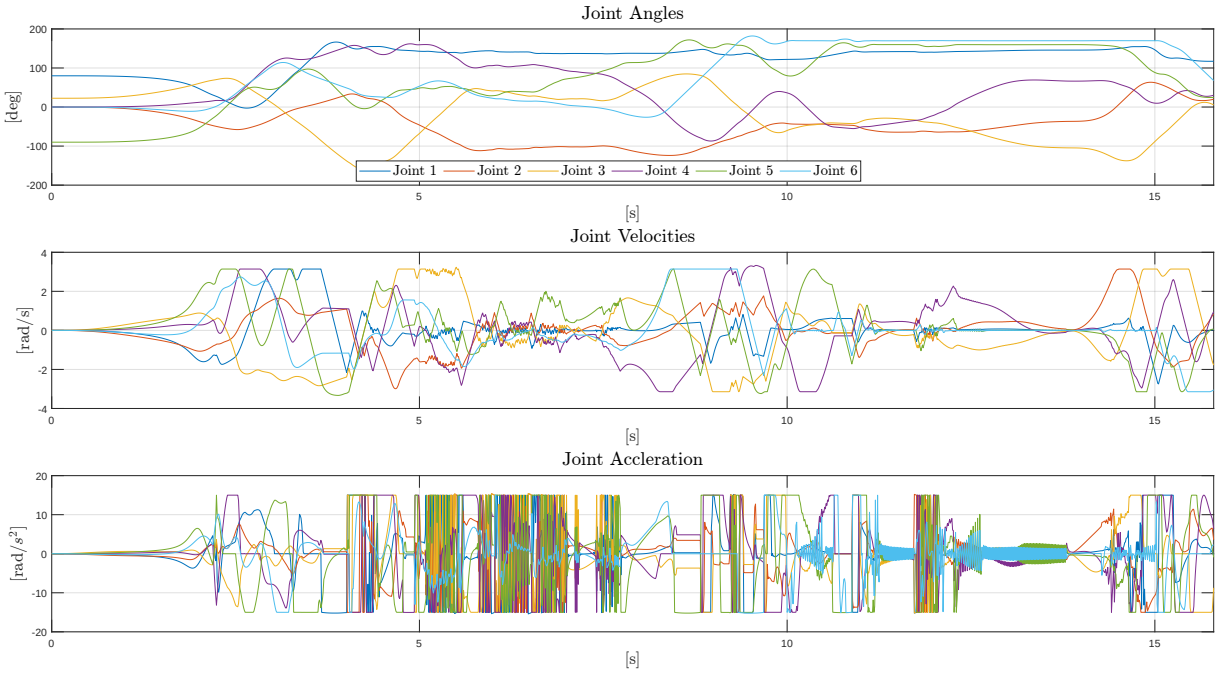


Figure 4.30: Joints Motion

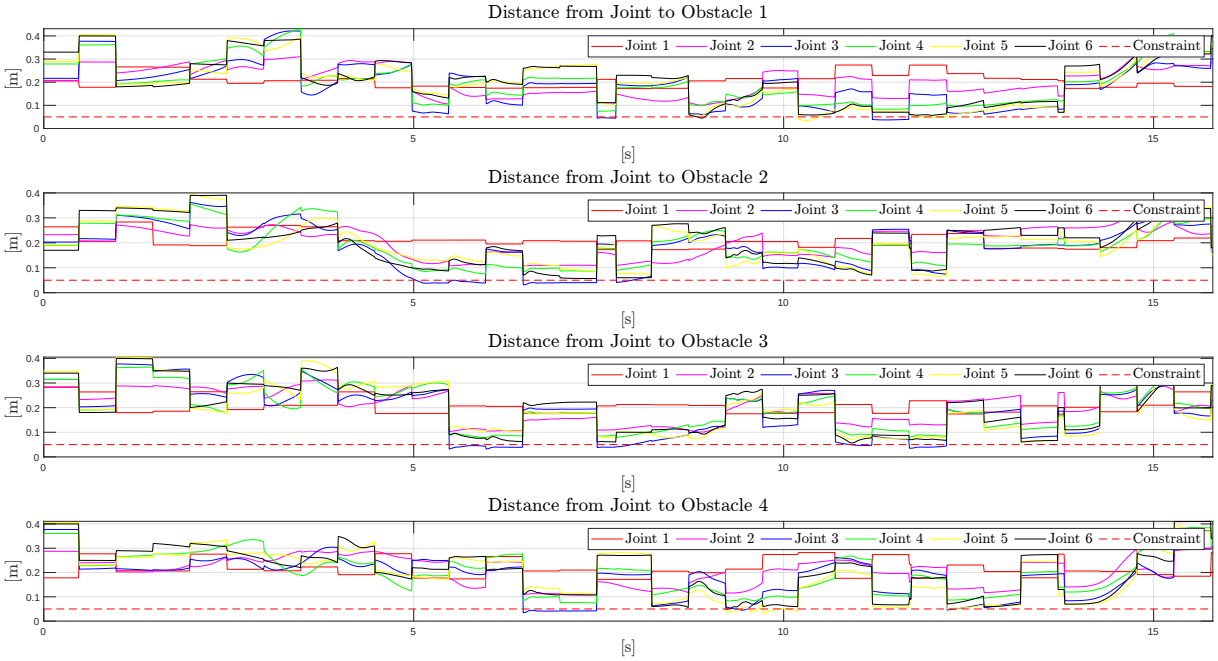


Figure 4.31: Distance from each link to each obstacle

Once again, the execution time results collected show the efficiency of the method for an online implementation with less than $1ms$ for the average time per iteration in the IK-MPC and small trajectory generation times. In addition, as before, the entire obstacle free region built as a union of all generated Polytopes is reported in Figure A.10.

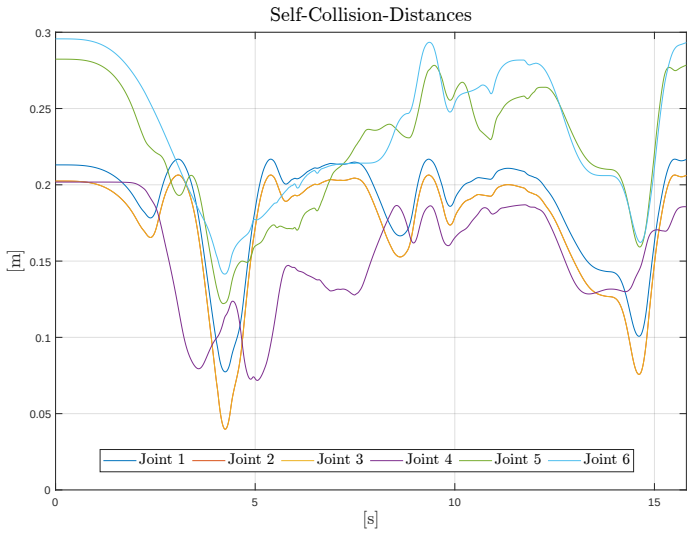
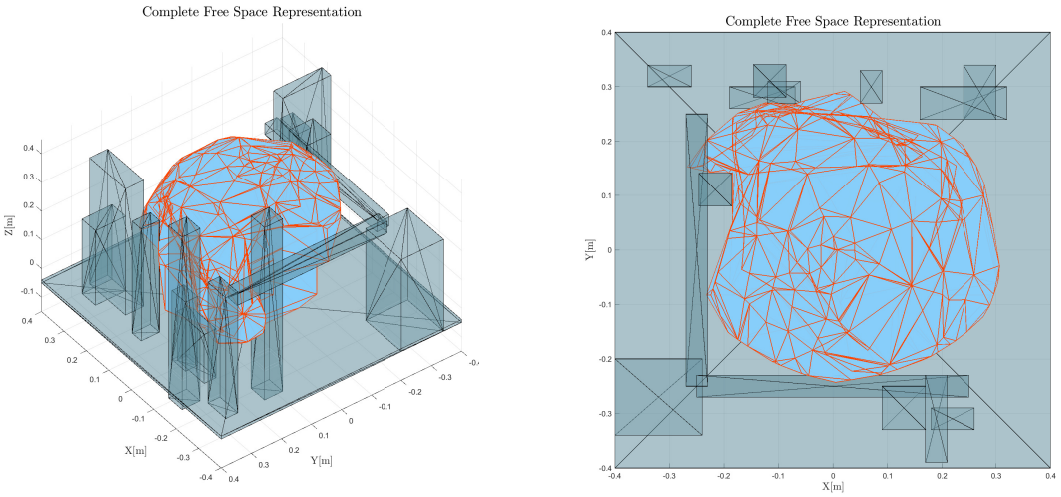


Figure 4.32: Self-Collision Distances. Constraint at 0.045 m.

Iteration	Traj. Time $[s]$	Traj. Gen. Time $[s]$	Iter. Avg. Time IK-MPC $[s]$
1	7.7356	1.6417	0.9652×10^{-3}
2	1.0515	0.6014	0.9100×10^{-3}
3	2.0000	0.1903	0.7186×10^{-3}
4	2.0650	0.2597	0.8244×10^{-3}
5	2.0000	0.3423	0.9167×10^{-3}

Table 4.7: Second Scenario: Time Execution Results.



(a) Isometric view. (b) Top view.

Figure 4.33: Second Scenario: Complete Obstacle Free Region.

5 | Conclusions and future developments

This thesis presented a novel hierarchical control framework for a 6-DOF robotic manipulator operating in unknown and dynamic environments. Starting from an understanding of the state-of-the-art, this work proposed novel strategies for Local Path Planning and a constrained Inverse Kinematics. The proposed partition guaranteed a double layer of safety (obstacle free trajectory generation and reactive obstacle avoidance) for obstacle avoidance which allows a safe motion of the manipulator towards a target position in an obstacle characterized environment .

In the Local Path Planning block, a Convex Approximation of the Free Space was proposed with the aim to generate obstacle free trajectories towards a target. An efficient strategy for target shifting was presented and tested. The results indicated its significance in achieving successful trajectory planning in intricate environments. The generated polynomial trajectories exhibited satisfactory performance, meeting the desired specifications.

The Inverse Kinematics used a Model Predictive Control approach, which exhibited improved performance compared to traditional one-step-ahead formulations. Through extensive parameter tuning, the framework achieved accurate trajectory following while satisfying the joint motion, obstacle and self-collision avoidance constraints. Furthermore, despite the fact the robot is not redundant, the self-collision avoidance constraint allowed different reconfiguration of the robot while operating. In addition, the average time per iteration for IK-MPC was consistently below $2ms$ for the presented scenarios that show its potential for an online real implementation. A comparison between the optimization algorithms were presented and tested to aim the best performance.

The integration of the Local Path Planning strategy and Inverse Kinematics into a unified hierarchical control system was evaluated in specific challenging scenarios with aleatory obstacles. The proposed system demonstrated robustness, safety, and efficiency in accomplishing tasks while satisfying the enforced constraints.

Nevertheless, the completeness of the method was not proven. During the tuning of

parameters the results of the system showed an inefficient path towards the target or even not being able to reach it (which was not the case after the tuning). These tests revealed that the system behavior is sensible to the choice of those parameters to achieve the target in a greater or lesser number of iterations. However, this characteristic is not unique of this method, well know methods such as planning via artificial potentials[34] are also not complete in general. Overall, the presented framework represents a viable and efficient solution to the considered problem, and its applicable in real world scenarios.

In conclusion, the developed hierarchical control framework exhibited promising capabilities for robust and adaptable robotic manipulator operation in complex environments. In fact, the proposed method was developed as general as possible to be able to include redundant, humanoids or hexapods robots in future works. Additionally, the proposed structure allows the inclusion of other components such an offline or sequential high level planner which can compute the referenced target p_{target} based on the specific application, or a customized low level control system which can run at a different discrete time. The results obtained elucidates a path for continued advancements in the field of robotic control systems, with potential applications in diverse industrial and research domains.

5.1. Future Developments

While the current research provides a solid foundation, there are still points for future research lines and enhancement. One of them, which was also proven in the results, is the generation of a constantly updated Local Map based on the obstacle free convex regions. This local map can be characterized as an occupancy map or other required techniques. Additionally, since the choice of the parameter max_{cos} is sensible, a future research can approach the target shifting as an optimization problem taking into account more variables than just the direction towards the target.

As for the case of the IK-MPC, the framework showcased its adaptability and versatility to include more requirements. The extension could point towards redundant manipulators or the inclusion of a mobile base in order to exploit redundancy and dexterity in the solution, which are important in obstacle characterized environments. In this regard, a manipulability index as well as other terms could be also included in the cost function. The obstacle avoidance functions can be extended for "N_{obs}" obstacles depending on the application. Moreover, the approximation of the links of the manipulator with primitives shapes can be done and enforced as inequality constraints to cover the whole kinematic chain. Additionally, the complete formulation as QP could be compared in terms of performance and computational time by solving the entire non-linear problem with a

SQP solver.

Finally, the extension of the proposed method to a real-world application is encouraged for experimental validation to evidence the capabilities of this work.

Bibliography

[1] G. B. Avanzini, A. M. Zanchettin, and P. Rocco. Constraint-based model predictive control for holonomic mobile manipulators. pages 1473–1479, 2015. doi: 10.1109/IROS.2015.7353562.

[2] K. Ayusawa, W. Suleiman, and E. Yoshida. Predictive inverse kinematics: optimizing future trajectory through implicit time integration and future jacobian estimation. pages 566–573, 2019. doi: 10.1109/IROS40897.2019.8968110.

[3] A. T. Azar, I. K. Ibraheem, and A. J. Humaidi. *Mobile Robot Motion Control and Path Planning*. Springer, 2023.

[4] B. Behroozpour, P. A. M. Sandborn, M. C. Wu, and B. E. Boser. Lidar system architectures and circuits. *IEEE Communications Magazine*, 55(10):135–142, 2017. doi: 10.1109/MCOM.2017.1700030.

[5] A. Bouman, M. Ginting, N. Alatur, M. Palieri, D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. Burdick, and A.-a. Agha-mohammadi. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. 10 2020.

[6] G. Buizza Avanzini. *Control strategies for redundant and mobile robotic manipulators subject to multiple constraints*. PhD thesis, Politecnico di Milano, 2015.

[7] G. Buizza Avanzini, A. M. Zanchettin, and P. Rocco. Constrained model predictive control for mobile robotic manipulators. *Robotica*, 36(1):19–38, 2018. doi: 10.1017/S0263574717000133.

[8] J. Castaño-Amorós, I. de Loyola Páez-Ubieta, P. Gil, and S. T. Puente. Visual-tactile manipulation to collect household waste in outdoor; [manipulación visual-táctil para la recogida de residuos domésticos en exteriores]. *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, 20(2):163 – 174, 2023. doi: 10.4995/riai.2022.18534.

[9] L. Cecchin. Graph-based exploration and mapping controller for mobile robots. Master's thesis, Politecnico di Milano, 2020.

[10] L. Cecchin, D. Saccani, and L. Fagiano. G-beam: Graph-based exploration and mapping for autonomous vehicles. pages 1011–1016, 2021. doi: 10.1109/CCTA48906.2021.9659296.

[11] P. Corke. *Robotics, vision and Control: Fundamental Algorithms in MATLAB*. Springer, 2023.

[12] R. Deits and R. Tedrake. *Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming*, pages 109–124. Springer International Publishing, Cham, Switzerland, 2015.

[13] M. Eckhoff, R. J. Kirschner, E. Kern, S. Abdolshah, and S. Haddadin. An mpc framework for planning safe and trustworthy robot motions. pages 4737–4742, 2022. doi: 10.1109/ICRA46639.2022.9812160.

[14] *MyCobot280: English Manual*. Elephant Robotics, 2020. URL `https://docs.elephantrobotics.com/docs/pdf/myCobot-en.pdf`.

[15] M. Faroni, M. Beschi, and A. Visioli. Predictive inverse kinematics for redundant manipulators: Evaluation in re-planning scenarios. *IFAC-PapersOnLine*, 51(22):238–243, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.11.548. 12th IFAC Symposium on Robot Control SYROCO 2018.

[16] M. Faroni, M. Beschi, and N. Pedrocchi. An mpc framework for online motion planning in human-robot collaborative tasks. pages 1555–1558, 2019. doi: 10.1109/ETFA.2019.8869047.

[17] W. Gong, H. Xu, Q. Li, H. Gu, C. Han, S. Song, and M. Q.-H. Meng. Mobile robot manipulation system design in given environments. pages 609–613, 2017. doi: 10.1109/ICInfA.2017.8078980.

[18] A. A. Hassan, M. El-Habrouk, and S. Deghedie. Inverse kinematics of redundant manipulators formulated as quadratic programming optimization problem solved using recurrent neural networks: A review. *Robotica*, 38(8):1495–1512, 2020. doi: 10.1017/S0263574719001590.

[19] F. Kanehiro, F. Lamiraux, O. Kanoun, E. Yoshida, and J.-P. Laumond. A local collision avoidance method for non-strictly convex polyhedra. *Robotics: Science and Systems IV*, pages 151–158, 2009. doi: 10.15607/rss.2008.iv.020.

[20] O. Kanoun, F. Lamiraux, and P.-B. Wieber. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4):785–792, 2011. doi: 10.1109/TRO.2011.2142450.

[21] D. Khudher and R. Powell. Quadratic programming for inverse kinematics control of a hexapod robot with inequality constraints. pages 1–5, 2016. doi: 10.1109/RCTFC.2016.7893402.

[22] D. Khudher and R. Powell. Quadratic programming for inverse kinematics control of a hexapod robot with inequality constraints. pages 1–5, 2016. doi: 10.1109/RCTFC.2016.7893402.

[23] M. Lei, L. Lu, A. Laurenzi, L. Rossini, E. Romiti, J. Malzahn, and N. G. Tsagarakis. An mpc-based framework for dynamic trajectory re-planning in uncertain environments. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 594–601, 2022. doi: 10.1109/Humanoids53995.2022.10000159.

[24] M. Lutz and T. Meurer. Efficient formulation of collision avoidance constraints in optimization based trajectory planning and control. *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 228–233, 2021. doi: 10.1109/CCTA48906.2021.9658663.

[25] L. Magni and R. Scattolini. *Advanced and multivariable control.* Pitagora Editrice S.r.l, 2014.

[26] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake. Motion planning around obstacles with convex optimization. 2022. doi: https://doi.org/10.48550/arXiv.2205.04422.

[27] R. Mascaro, M. Wermelinger, M. Hutter, and M. Chli. Towards automating construction tasks: Large-scale object mapping, segmentation, and manipulation. *Journal of Field Robotics*, 38(5):684–699, 2021. doi: https://doi.org/10.1002/rob.22007.

[28] MATLAB. *version 9.14.0 (R2023a).* The MathWorks Inc., Natick, Massachusetts, 2023.

[29] M. Petersen and R. Tedrake. Growing convex collision-free regions in configuration space using nonlinear programming. 2023. doi: https://doi.org/10.48550/arXiv.2303.14737.

[30] M. Rahman, H. Liu, M. Masri, I. Durazo-Cardenas, and A. Starr. A railway track reconstruction method using robotic vision on a mobile manipulator: A proposed

strategy. *Computers in Industry*, 148:103900, 2023. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2023.103900.

[31] D. Saccani, L. Cecchin, and L. Fagiano. Multitrajectory model predictive control for safe uav navigation in an unknown environment. *IEEE Transactions on Control Systems Technology*, 31(5):1982–1997, 2023. doi: 10.1109/TCST.2022.3216989.

[32] M. Satoh, A. Bhat, T. Usami, and K. Tatsumi. A multi-purpose autonomous mobile robot as a part of agricultural decision support systems. pages 1–7, 2023. doi: 10.1109/CASE56687.2023.10260483.

[33] A.-N. Sharkawy. Minimum jerk trajectory generation for straight and curved movements: Mathematical analysis. In *Advances in Robotics and Automatic Control: Reviews*, pages 187–201. International Frequency Sensor Association Publishing (IFSA Publishing, S. L.), 2021. ISBN 978-84-09-25863-5. doi: 10.48550/arXiv.2102.07459.

[34] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, planning and control*. Springer, 2010.

[35] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Motion planning. In *Robotics: Modelling, planning and control*, pages 523–559. Springer, 2010.

[36] Simulink. Simulation and model-based design, 2023. URL `https://www.mathworks.com/products/simulink.html`.

[37] W. Suleiman. On inverse kinematics with inequality constraints: new insights into minimum jerk trajectory generation. *Advanced Robotics*, 30(17-18):1164–1172, 2016. doi: 10.1080/01691864.2016.1202136.

[38] I. The MathWorks. *Robotics System Toolbox*. Natick, Massachusetts, United State, 2021. URL `https://www.mathworks.com/help/robotics/`.

[39] P. Werner, A. Amice, T. Marcucci, D. Rus, and R. Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. 2023. doi: https://doi.org/10.48550/arXiv.2310.02875.

[40] U. Wolinski and M. Wojtyra. A novel qp-based kinematic redundancy resolution method with joint constraints satisfaction. *IEEE Access*, 10:41023–41037, 2022. doi: 10.1109/access.2022.3167403.

[41] S. Yang, Y. Zhang, H. Wen, and D. Jin. Coordinated control of dual-arm robot on space structure for capturing space targets. *Advances in Space Research*, 71(5): 2437–2448, 2023. ISSN 0273-1177. doi: https://doi.org/10.1016/j.asr.2022.10.027.

[42] W. Zhang and H. Wen. Motion planning of a free-flying space robot system under end effector task constraints. *Acta Astronautica*, 199:195–205, 2022. ISSN 0094-5765. doi: https://doi.org/10.1016/j.actaastro.2022.07.005.

[43] S. Zimmermann, R. Poranne, and S. Coros. Go fetch! - dynamic grasps using boston dynamics spot with external robotic arm. pages 4488–4494, 2021. doi: 10.1109/ICRA48506.2021.9561835.

[44] Łukasz Woliński and M. Wojtyra. An inverse kinematics solution with trajectory scaling for redundant manipulators. *Mechanism and Machine Theory*, 191:105493, 2024. ISSN 0094-114X. doi: https://doi.org/10.1016/j.mechmachtheory.2023.105493.

A | Appendix A

A.1. Additional Tests

A.1.1. Test 1

Test with visibility of the target but with obstacle obstructing the path planned.

Description	Value
Initial Configuration [rad]	$[1.3962, 0, \pi/8, \pi/8, 2.2689, 0]$
Initial Position [m]	(0.0137,-0.1124 ,0.3626)
Target Position [m]	(-0.0500,-0.2500, 0.1500)

Table A.1: Test 1: Initial Configuration.

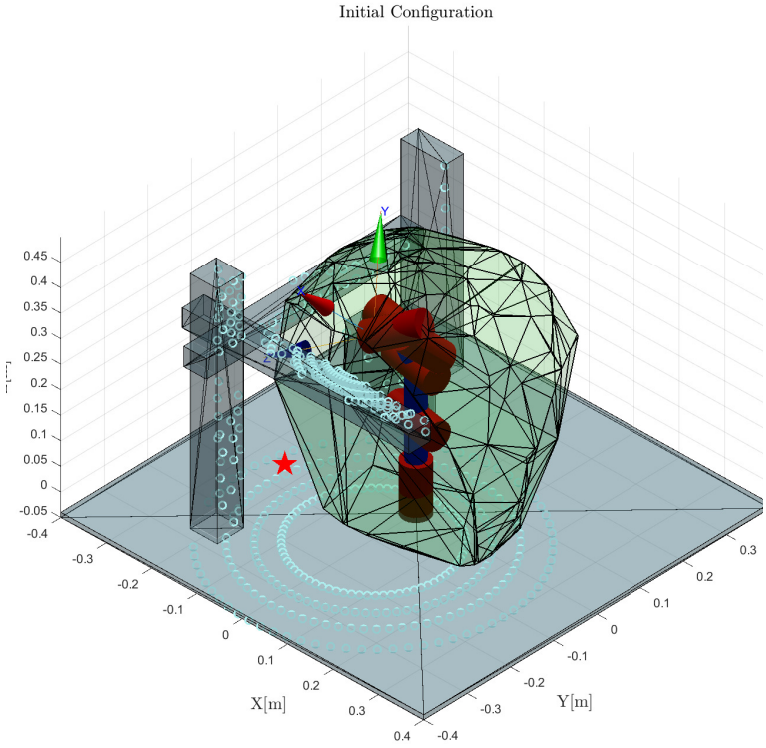


Figure A.1: Test 1: Initial Configuration

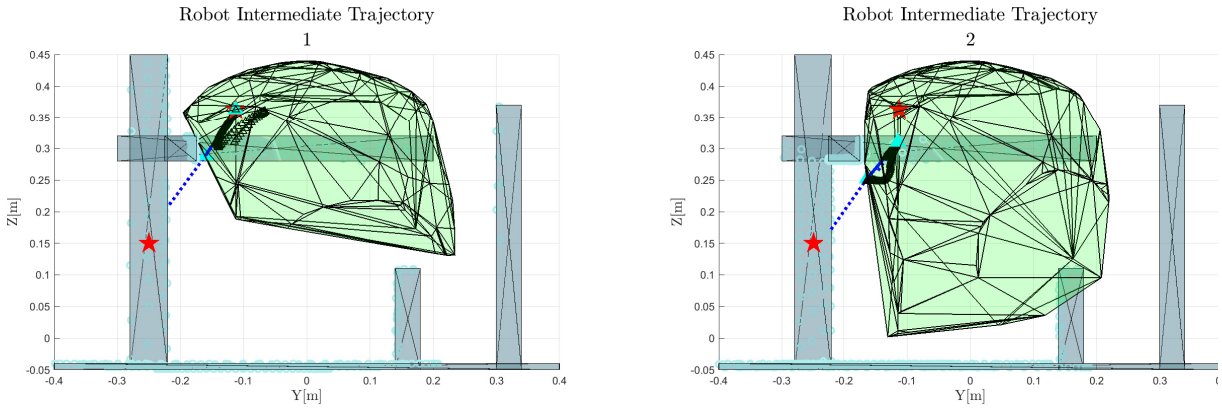


Figure A.2: Intermediate Trajectories 1 and 2

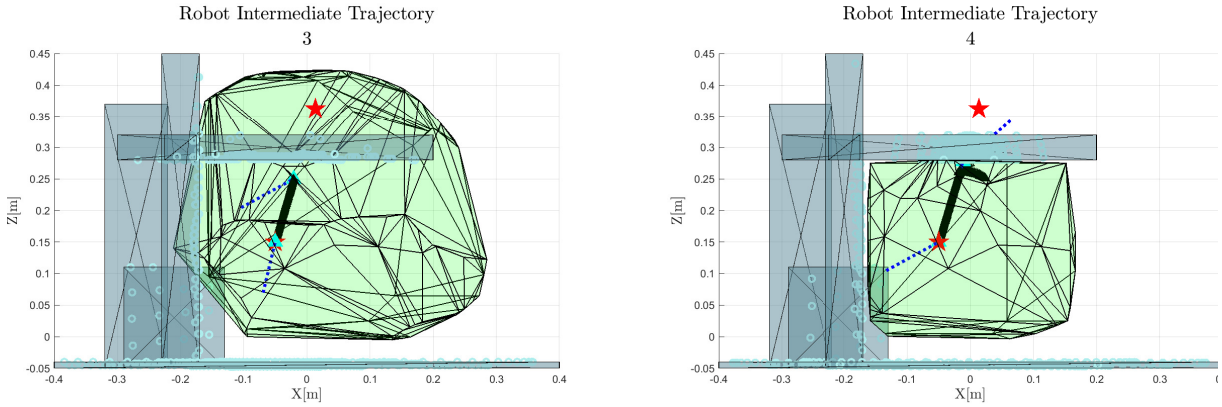


Figure A.3: Intermediate Trajectories 3 and 4

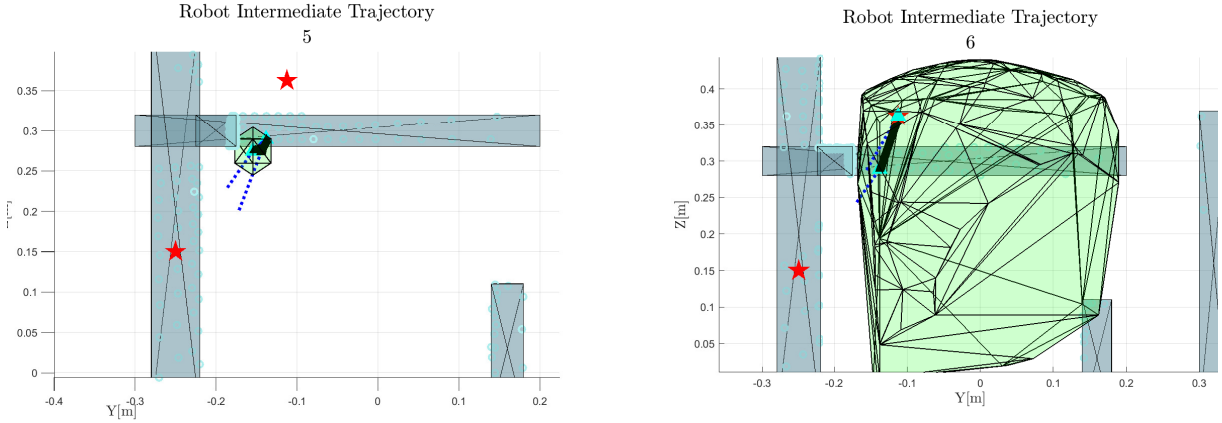


Figure A.4: Intermediate Trajectories 5 and 6

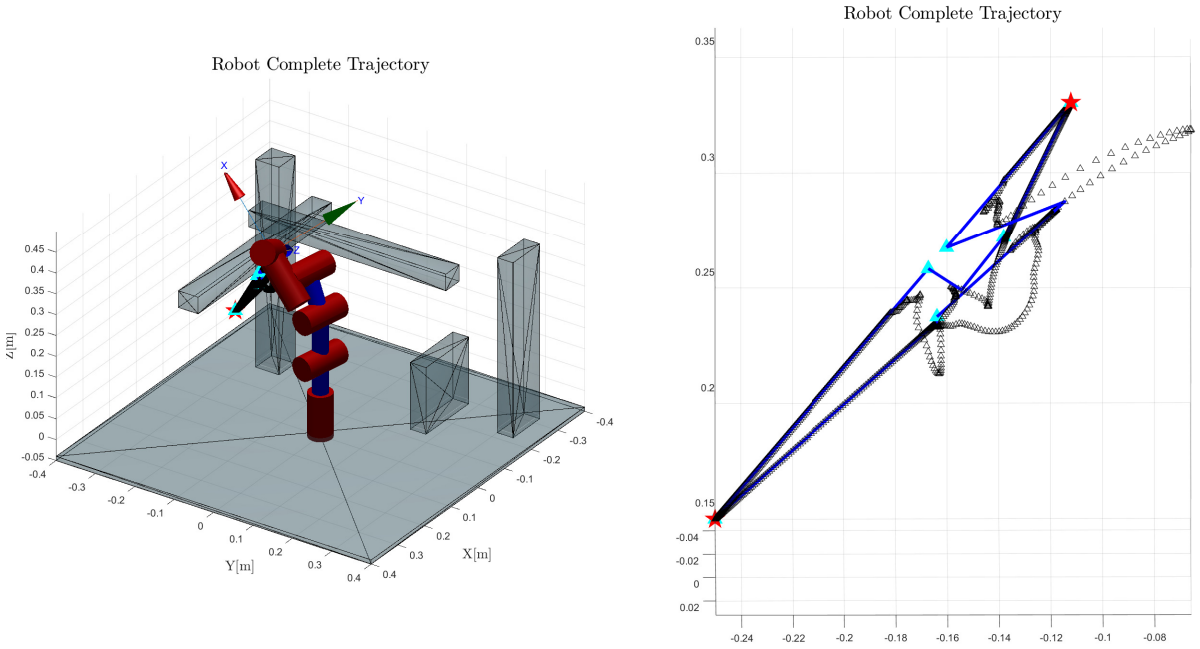


Figure A.5: Complete Performed Trajectory

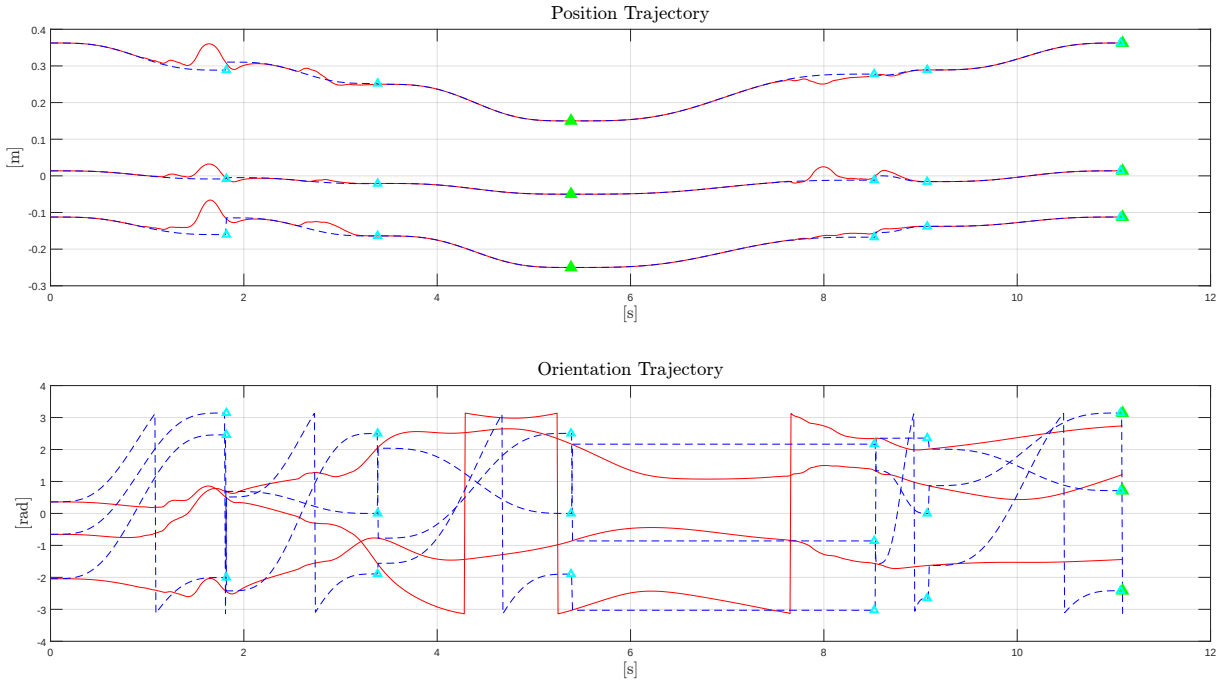


Figure A.6: Performed Trajectory(red) and Generated Trajectory(blue dotted)

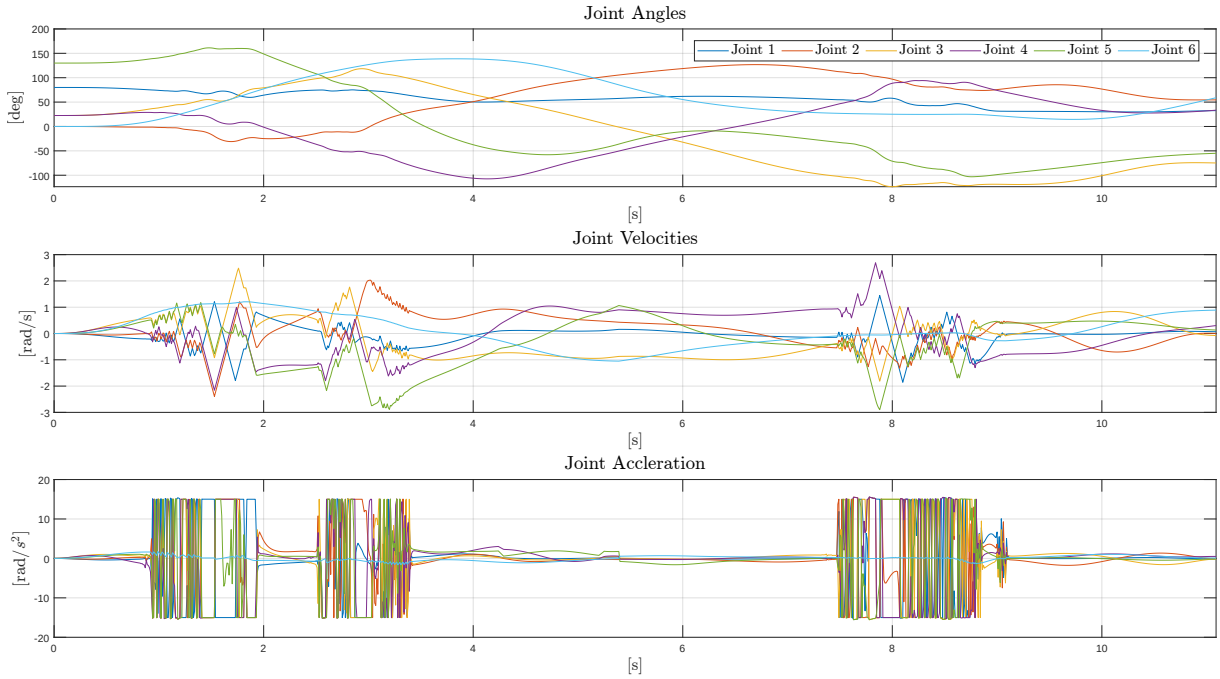


Figure A.7: Joints Motion

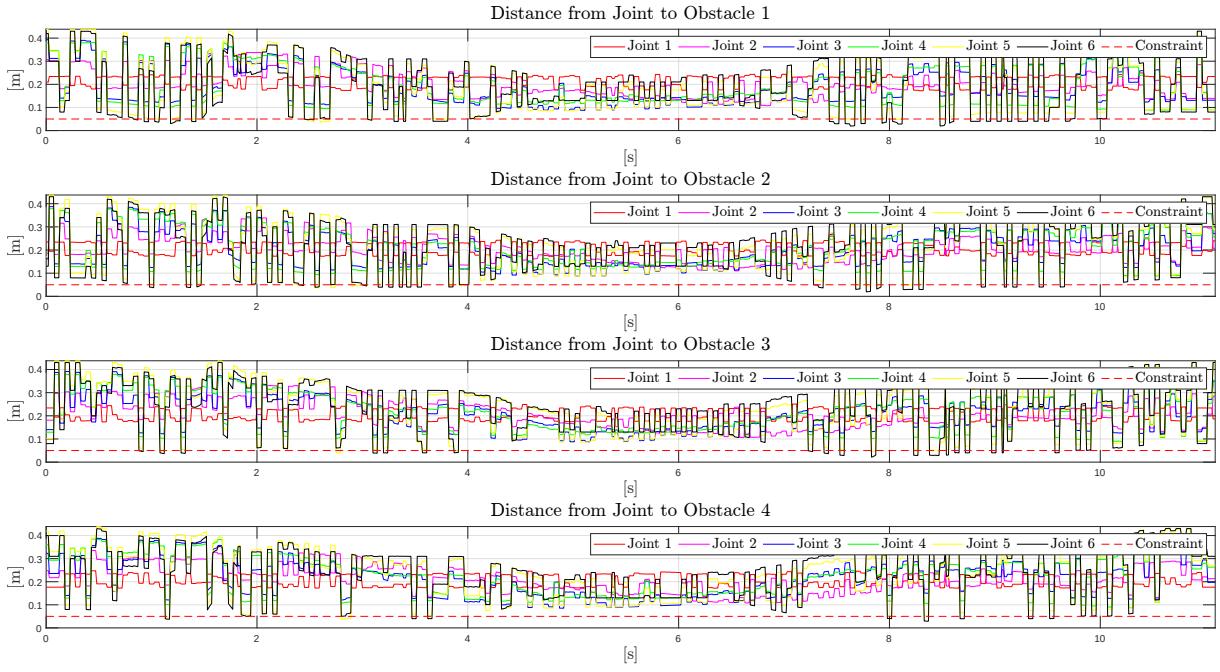


Figure A.8: Distance from each link to each obstacle

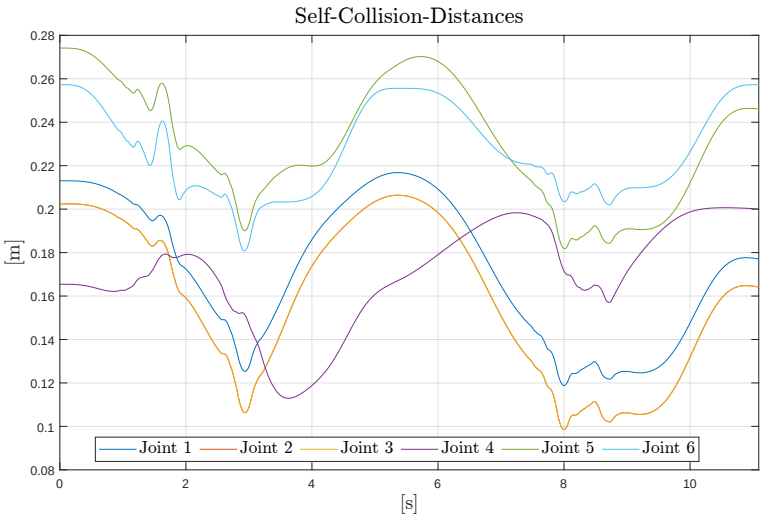


Figure A.9: Self-Collision Distances

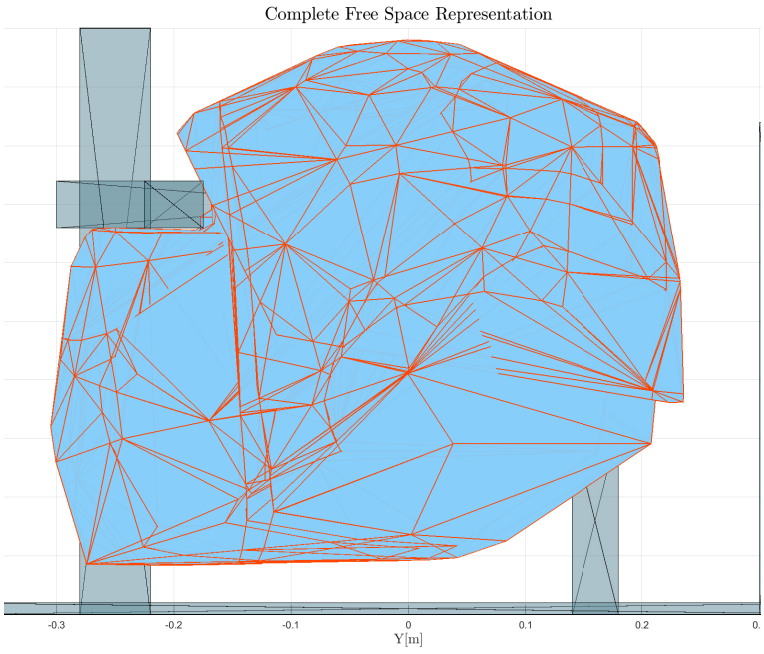


Figure A.10: Complete Obstacle Free Region.

A.1.2. Test 2

Test in a cluttered environment with flying obstacles but visibility of the target.

Description	Value
Initial Configuration [rad]	$[1.7453, -\pi/8, \pi/8, \pi/8, 2.4435, 0]$
Initial Position [m]	(0.0407,-0.0097,0.3844)
Target Position [m]	(0.0000,-0.2000,0.2800)

Table A.2: Test 2: Initial Configuration.

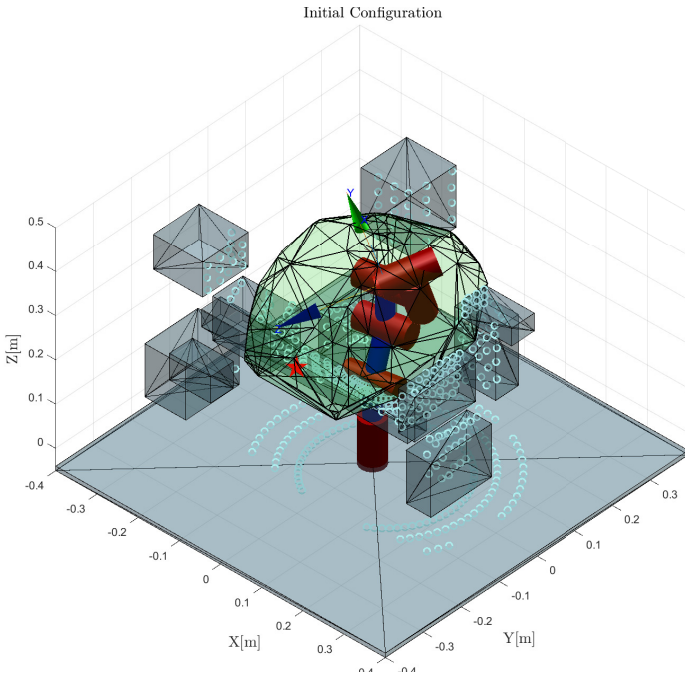


Figure A.11: Test 1: Initial Configuration

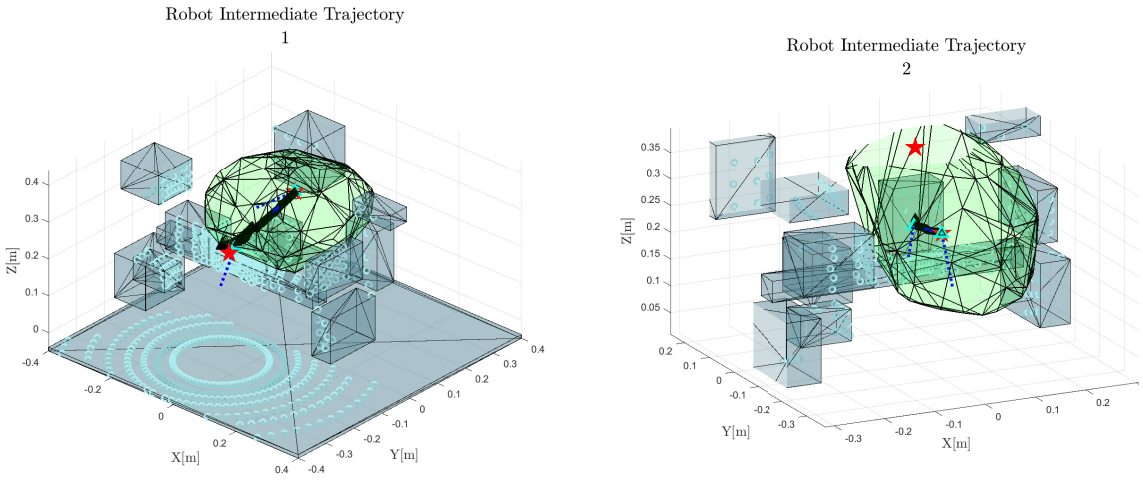


Figure A.12: Intermediate Trajectories 1 and 2

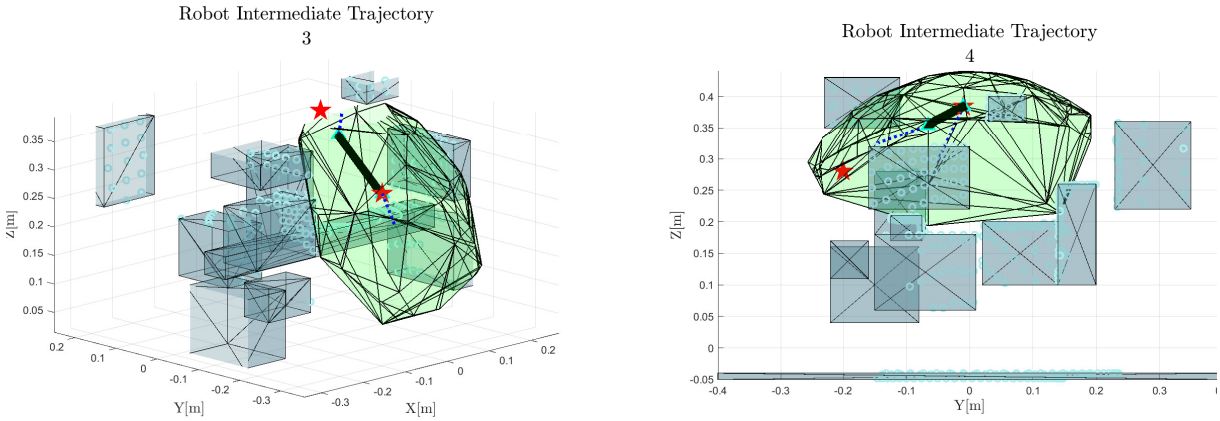


Figure A.13: Intermediate Trajectories 3 and 4

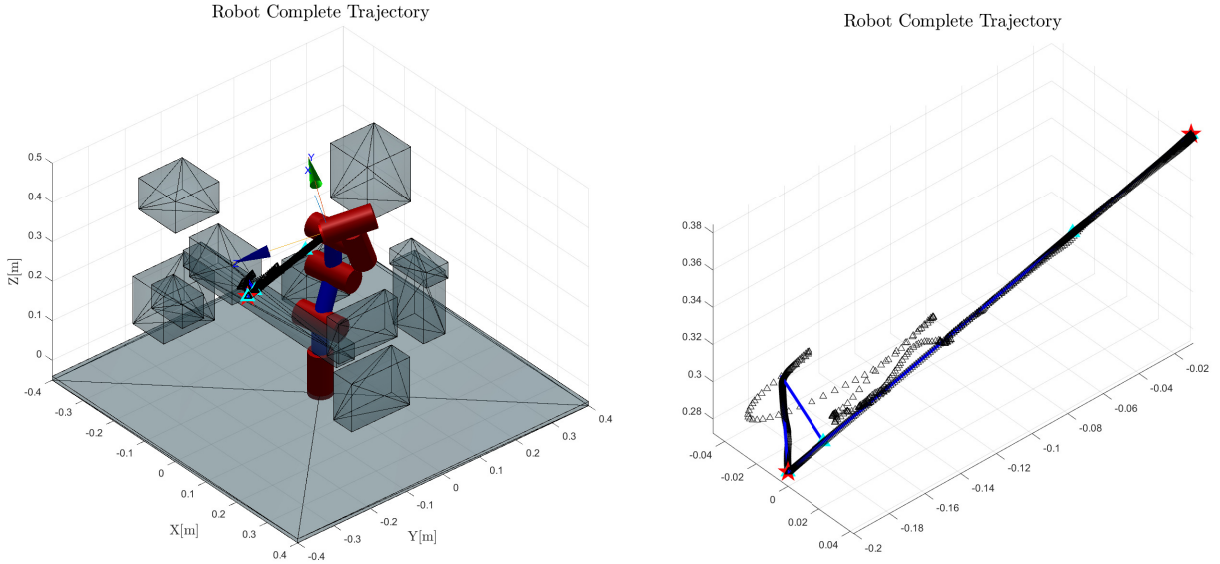


Figure A.14: Complete Performed Trajectory

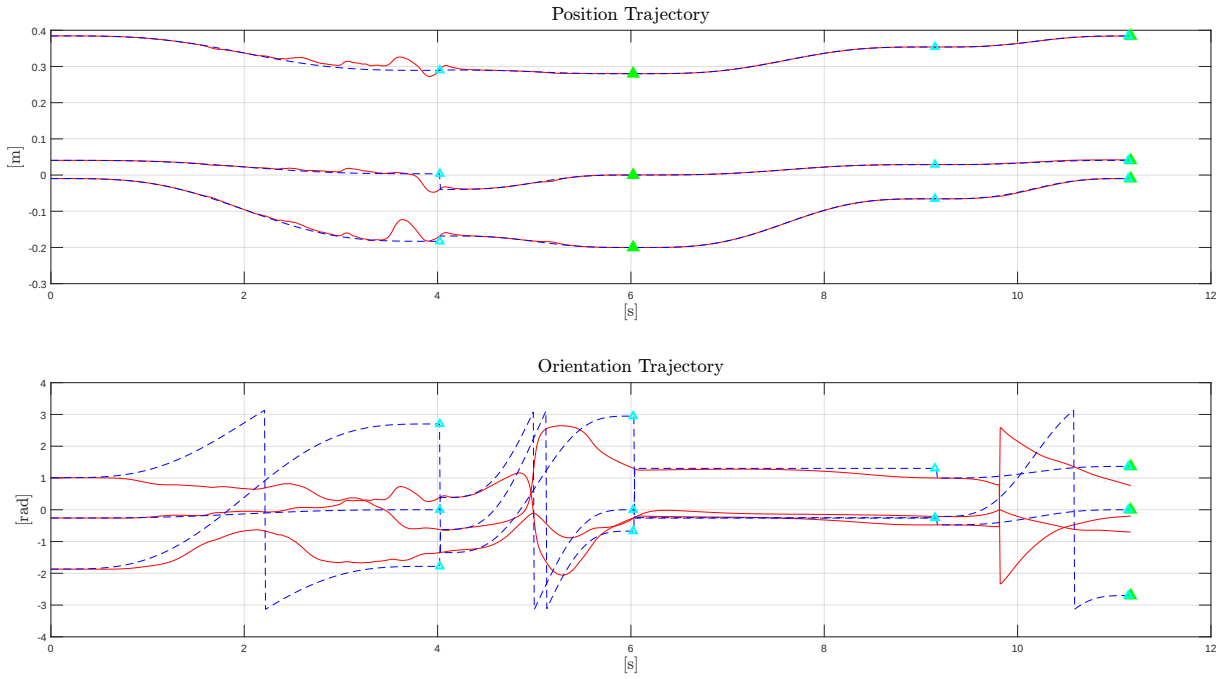


Figure A.15: Performed Trajectory(red) and Generated Trajectory(blue dotted)

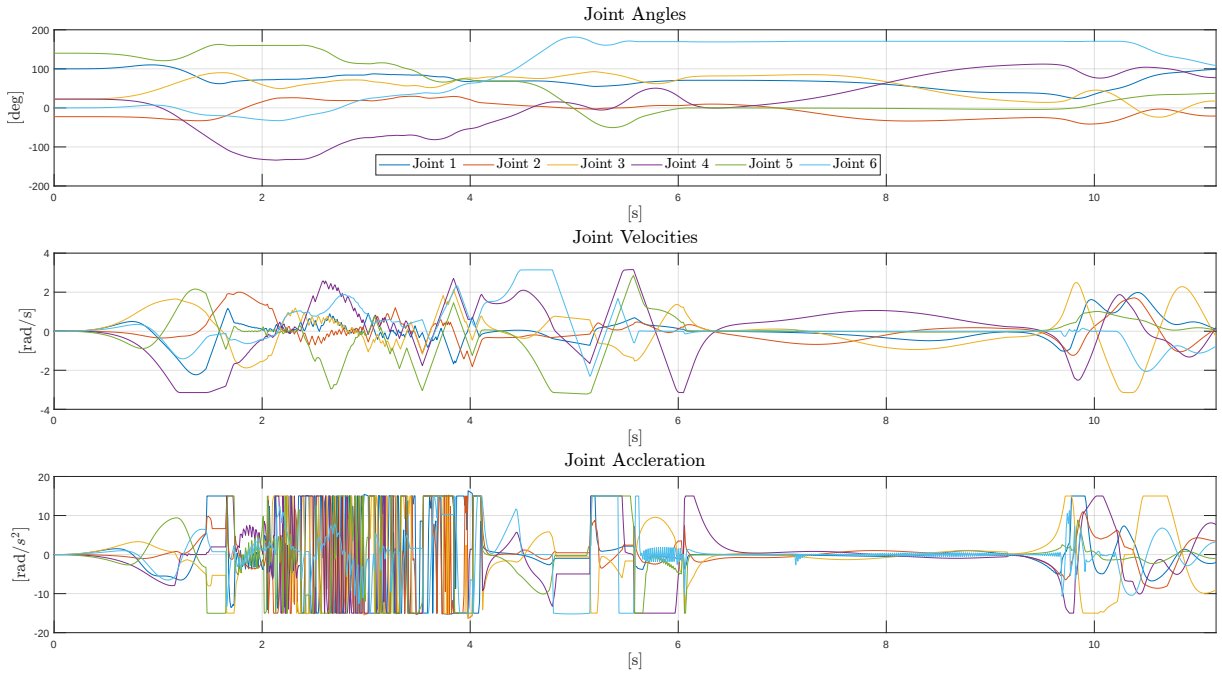


Figure A.16: Joints Motion

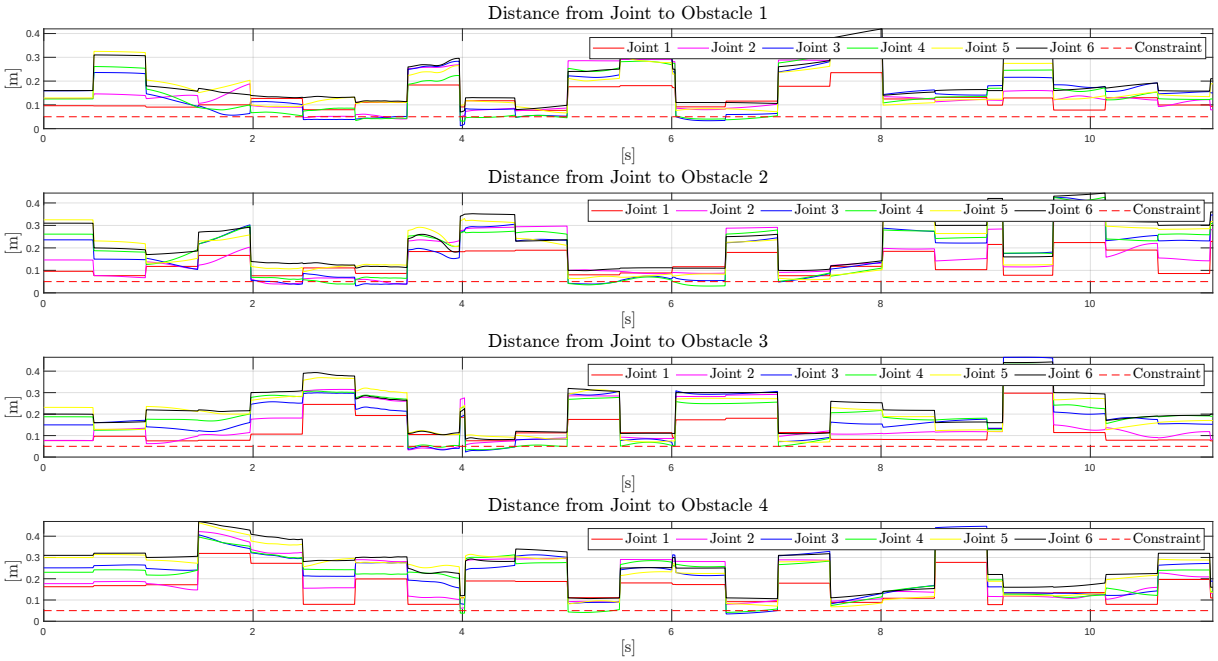


Figure A.17: Distance from each link to each obstacle

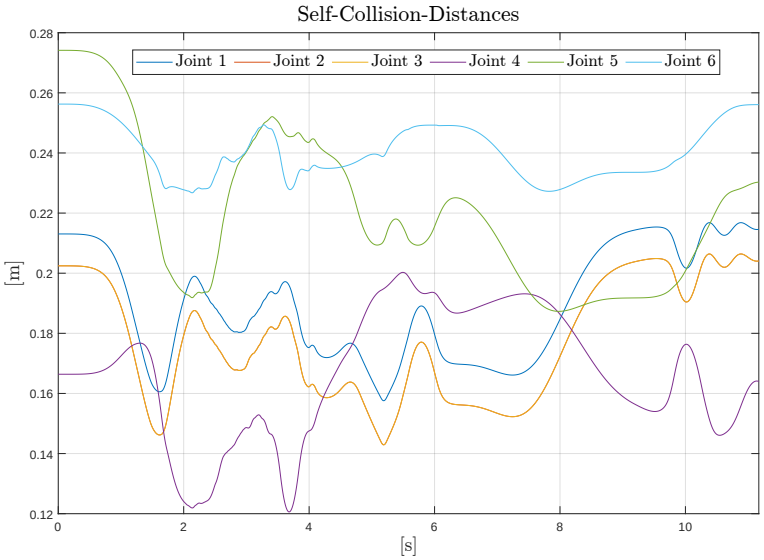
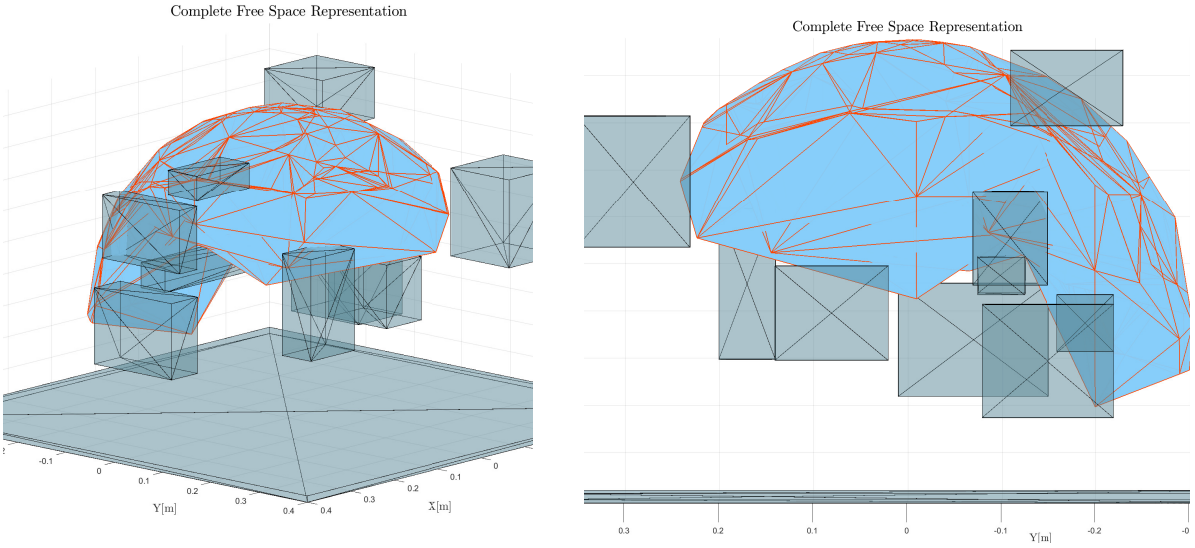


Figure A.18: Self-Collision Distances



(a) Isometric view.

(b) Side view.

Figure A.19: Complete Obstacle Free Region.