POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Meccanica

TESI DI LAUREA MAGISTRALE

COMPARATIVE ANALYSIS OF INVERSE KINEMATICS STRATEGIES FOR EFFECTIVE CONTROL OF REDUNDANT MANIPULATORS IN COLLABORATIVE ROBOTICS

Relatori:	*Candidato:*
Prof. Stefano Paolo Pastorelli	Tommaso Zinno
Dott. Elisa Digo	Matricola:313358
Dott. Valerio Cornagliotto	

Anno Accademico 2024/2025
Sessione di Laurea Aprile 2025

Table of contents

ABSTRACT

Inverse kinematics (IK) is a crucial concept in robotics, enabling a robot to determine the joint configurations required to achieve a desired end-effector position. In the case of redundant robots, which have more degrees of freedom (DOF) than necessary for a specific task, the IK problem becomes more complex. However, the redundancy provides the opportunity to optimize the robot's motion, considering factors such as joint limits, obstacle avoidance, and energy efficiency.

In the first chapter, the introduction provides an overview of the state-of-the-art in collaborative robots and their increasing role in Industry 5.0, emphasizing their challenging integration into Human-Robot teams.

In the second chapter, the model of a linear manipulator is presented, along with basic approaches to solving the inverse kinematics problem for redundant robots. The focus is on the mathematical formulations and the algorithm implementation. Specifically, the pseudo-inverse, weighted pseudo-inverse, and Damped Least Squares (DLS) methods are proposed.

In the third chapter, two trajectories are analyzed to explore and understand the functioning of the said techniques.

In the last one, conclusions regarding the algorithms are formulated, such as their advantages and limits based on criteria like: task constraints, robot singularities and which of them can be the most appropriate solution for a Human-Robot-Collaboration (HRC).

INTRODUCTION

1.1 Collaborative robotics

Collaborative robots, referred to as cobots or co-robots, are one of the latest technologies in robotics. They are designed to interact with human operators in a shared work environment guaranteeing safety while shortening production times and hence improving the efficiency.

The first cobot was invented in 1996 by J. Edward Colgate and Michael Peshkin. The definition is that of "*a device and method for direct physical interaction between a person and a computer-controlled manipulator*."

Since then, numerous cobots have been marketed. Kuka Robotics, which also launched one of the first industrial robots on the market, launched its first cobot in 2004, the LBR 3. Universal Robots, one of the world's largest cobot suppliers, launched its first cobot in 2008, the UR5. Seven years later, it was followed by UR3, the one used in this thesis [1].

Cobots were being conceived principally because some tasks were too complex or expensive to be carried out by an industrial robot, or too weary and repetitive to be wholly made by operators. Accordingly, it is possible to combine human and robot complementary capabilities: the precision and fatigue-free nature typical of the robotic structure are exploited together with the operator's ability to make decisions, predict and solve inaccurate situations, and adapt to the flexibility and variability of tasks.

The main differences between industrial robots and collaborative robots can be summarized in three terms: *safety*, *flexibility* and *speed of installation*.

Safety is the main parameter that defines collaborative robots as they are equipped with integrated safety systems, such as emergency stop, limits on joint speeds and other passive measures that allow the sharing of the workspace with the operator without the aid of physical barriers. First of all, the size of cobot is smaller than that of a typical industrial robot. However, the safety feature derives not only from the size of the robot and from the limitations on movement speeds, but also from the possibility of equipping the structures with sensors, such as vision systems. These technologies allow to record the operator's position and consequently to control the movements of the robot and its reaction times [2].

As far as *flexibility* is concerned, an industrial robot performs well in terms of productivity and economic return when it works on large volumes, as it ensures great speed of production and repeatability. Indeed, it is a type of automation called 'rigid', as it does not allow large variations in the layout of the production system and the change of production takes time. On the contrary, since collaborative robots are typically smaller, more compact and lighter, they can be easily and quickly moved within the industrial layout; in addition, they are easily reprogrammable for quick job changes.

Finally, differently from bulky industrial robots, the *installation* of cobots does not require major changes in the production layout. This allows for a quicker integration inside work areas as there is no need to install physical barriers separating robots from operators. Therefore, the implementation of such robotic structures is not only simpler but also much faster than that of industrial robots.

Thanks to these characteristics, collaborative robotics has made it possible to introduce automation in sectors and industries where traditional robotics could not be applied due, for example, to limited budgets, insufficient space in the industrial layout, or lack of operators specialized in programming industrial robots [2].

Typical industries in which cobots are deployed are food, plastics, packaging, electronics, pharmaceutical, automotive, and metal.

1.1.1 Human-robot Collaboration (HRC)

The transition of robots from caged robots to collaborative ones has been a gradual process, driven by advancements in technology and changing attitudes towards robotics in the workplace, as seen in Fig. 1.1.0 Here there is a brief overview of the key stages of this transition [3]:

- **Caged Robots**: In the early stages of industrial automation, robots were confined within protective cages or enclosures. These physical barriers were designed to prevent any direct contact or interaction between robots and human workers. The primary objective of this approach was to ensure safety in the workplace, as the robots employed during this period were limited in their capabilities and lacked the advanced safety mechanisms necessary for safe coexistence with humans.
- **Collision Avoidance**: As technological advancements continued to unfold, robots began to be equipped with a range of sensors and cameras that enabled them to

detect the presence of humans in their vicinity. These sensors allowed robots to identify the positions of nearby humans and respond appropriately to prevent potential collisions or accidents. This marked a significant step forward in enhancing safety in shared workspaces, as robots could now slow down or stop their movements when a human was nearby, reducing the risk of injuries and accidents.

- **Human–Robot Interaction (HRI)**: Further progress in robotics brought about breakthroughs in natural language processing and speech recognition technologies. These developments enabled robots to understand and respond to verbal commands and cues from humans. As a result, the communication gap between humans and robots began to narrow significantly. This phase ushered in a new era of more interactive and responsive robot behavior, making it easier for humans to work alongside and instruct robots effectively [3].
- **Human–Robot Collaboration (HRC)**: Recent advancements have shifted the focus towards HRC. In this phase, robots and humans actively collaborate on tasks, often in close proximity. This collaborative approach necessitates robots' ability to interpret human intentions, cooperate effectively, and ensure safety throughout the collaborative process. HRC represents a profound shift from the earlier isolation of robot functions to a mode where humans and robots work together as complementary partners.
- **Physical HRC (pHRC)**: The next stage in the evolution of robotics is Physical Human–Robot Collaboration. At this level, robots are not only collaborating with humans but also physically interacting with them. This interaction may involve tasks such as sharing tools, passing objects, or jointly manipulating objects. Achieving successful pHRC requires the development of highly advanced sensing and control systems that ensure safe and efficient cooperation. This stage represents a deeper physical integration between humans and robots, where their actions are closely intertwined.
- **Human–Robot Teaming (HRT)**: The last stage of robotics evolution is Human–Robot Teaming. In this advanced stage, robots are integrated into human teams as equal partners. HRT demands sophisticated AI and machine learning algorithms that enable robots to adapt to human behaviors, preferences, and decision-making processes. These robots become active, adaptive team members that work alongside humans to achieve common goals HRT marks a paradigm shift in the relationship between humans and robots, where robots are not just passive instruments but active

contributors to collaborative endeavors. This final level of integration and teamwork represents the cutting edge of robotics technology, paving the way toa wide range of applications across various industries [3].

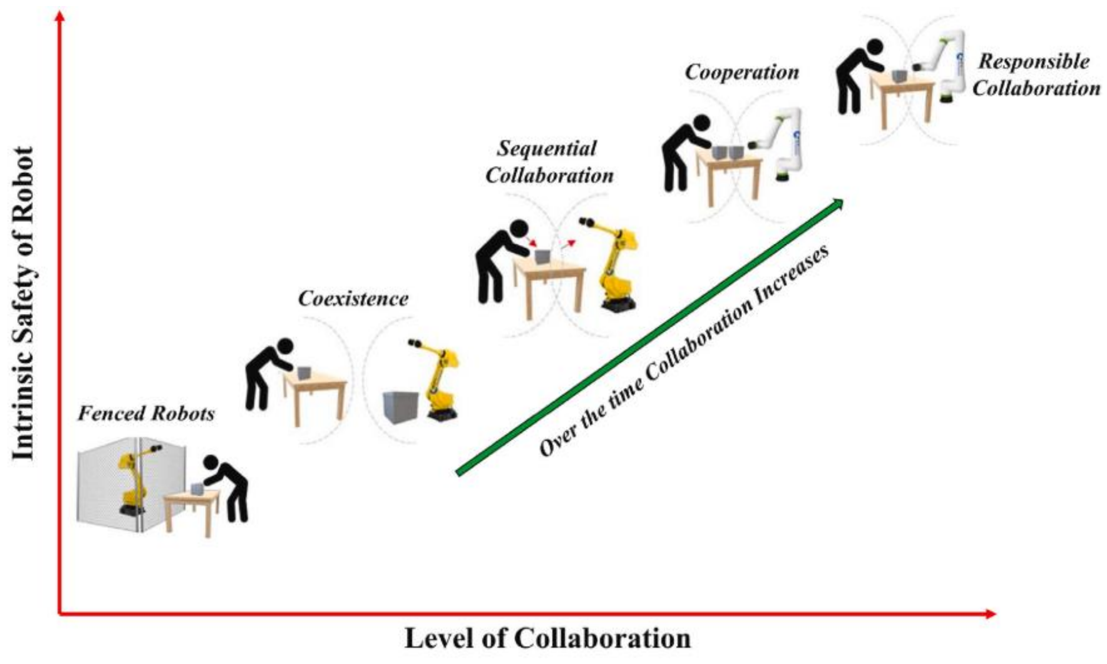
Intrinsic Safety of Robot
Level of Collaboration
Fenced Robots
Coexistence
Sequential Collaboration
Cooperation
Responsible Collaboration
Over the time Collaboration Increases

Figure 1.1.0 - Different types of shared workspace in HRC systems [3].

1.2 From Industry 4.0 towards industry 5.0

A new industrial paradigm, Industry 5.0, emerged shortly after Industry 4.0, sparking discussions about its application and relevance. Industry 4.0 is built around the concept of the smart factory, where smart products, machines, storage systems, and data converge in the form of cyber-physical production systems. From a technical perspective, Industry 4.0 has enhanced human-machine interaction, but in terms of social sustainability, the technological advancements of Industry 4.0 must carefully account for the central role of humans.

The significance of employees became especially evident during the COVID-19 pandemic, which also prompted a reevaluation of the Industry 4.0 paradigm. As a result, the concept of Industry 5.0 emerged as a natural extension of Industry 4.0, incorporating social and environmental dimensions. Industry 5.0 focuses on workers' skills, knowledge, and their ability to collaborate with machines and robots, while also emphasizing flexibility in production processes and minimizing environmental impact.

As a reference to past trends, Figure 1.2.0 illustrates the transformations through the paradigms, highlighting the key participants and industry segments involved [4].

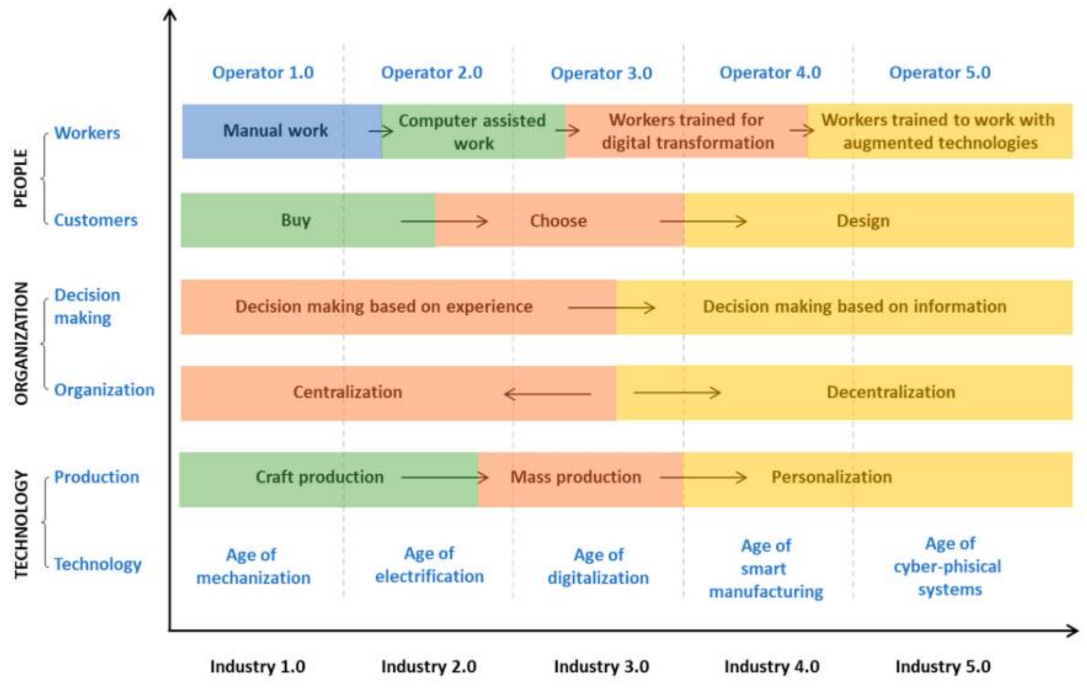

Figure 1.2.0 - Transformations through industrial paradigms [4].

1.2.1 Basic Driving Concepts of Industry 4.0 and Industry 5.0

Industry 4.0 is based on the concept of smart factories, whose initiative was founded by partners from industry and academy as an environment for test future technologies and to learn by doing. There are important key drivers [4]:

- **Internet of Things**, services and data that enable the communication between objects. By placing the intelligence into objects, they are turned into smart objects able not only to collect information from the environment and interact or control the physical world, but also to be interconnected to each other through Internet, to exchange data and information.
- **Cloud computing** is a driver which supports the Internet of Things, enabling the access to large datasets and its processing to generate new useful information through different types of reports.
- **Cyber-physical system (**CPS) is defined as a new generation system with integrated computational and physical capabilities that can interact with people through new modalities.
- **Artificial intelligence** supports the cyber-physical system for filtration of the multitude data incoming from different sensors in a production system and analyzes it through the reports. It offers the data-driven predictive analytics and capacity to assist decision- making in highly complex, nonlinear, and multistage production.
- **Augmented reality** (AR) represents the integration of the virtual and real environments where objects in the real world are enhanced by computer-generated information or objects with the help of different technologies.
- **Simulation** is a powerful tool used for decision making. The application of simulation methods is becoming increasingly relevant as developments in the field of digitalization lead to more comprehensive, efficient, embedded, and cost-effective simulation methods

  Autonomous robots can detect problems and independently adjust their tasks to ensure that processes run smoothly. However, there are levels of robot autonomy, ranging from teleoperation to fully autonomous systems, that influence human–robot interaction.

These elements enable the connectivity of the virtual and real world in order to achieve better results in production with maximum profit. A completely profit-driven approach is not sustainable for the long term. Instead of taking technology as a crucial element, the document of European Commission sees three key drivers as the center of new industrial paradigm Industry 5.0 (Figure 1.2.1):

- **Human-centric** approach, which places human needs at the heart of the production process, asking what technology can do for workers and how can it be useful.
- **Sustainability**, which focuses on reuse, repurpose, and recycle of natural resources and reduce of waste and environmental impact.
- **Resilience**, which implies an introduction of robustness in industrial production. This robustness provides support through flexible processes and adaptable production capacities, especially when a crisis occurs.

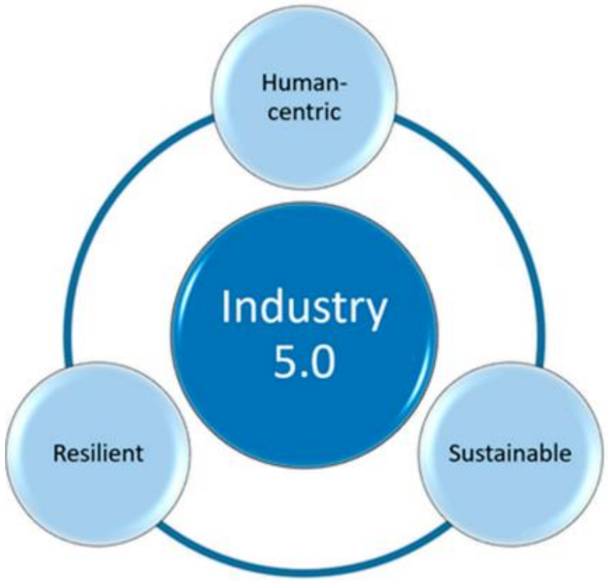

Figure 1.2.1 - Industry 5.0 with three key drivers [4]

According to the European Commission, Industry 5.0 is a necessary evolutionary step of Industry 4.0 because of three important issues: (1) **Industry 4.0 is not the right framework to achieve Europe's 2030 goals,** because the current digital economy is a winner-takes-all model that creates technological monopoly and giant inequality. (2) **Industry 5.0 is not a technological leap forward**, but a way to see the Industry 4.0 approach in a broader context, providing regenerative purpose and directionality to the technological transformation of industrial production for people–planet–prosperity. (3) **Industry 5.0 is a transformative model** that reflects the evolution of our thinking postCOVID-19 pandemic, by taking into consideration learnings from the pandemic and the need to design an industrial system that is inherently more resilient to future shocks and truly integrates social and environmental principles [4].

1.2.2 Opportunities and challenges in integrating cobots in Industry 5.0

The transition from Industry 4.0 to Industry 5.0 marks a significant shift towards human centric manufacturing processes, emphasizing the integration of collaborative robots with advanced sensory and cognitive abilities. Unlike previous industrial revolutions, Industry 5.0 prioritizes the integration of human workers alongside advanced technologies, emphasizing collaboration, and acknowledging the unique strengths of both humans and machines, with a focus on human wellbeing.

The transition, however, also poses several challenges [5]:

1) **Concerns about how safety considerations impact performance, leading to hesitancy in adopting cobots.**
   Despite their advantages, the deployment of cobots remains
   limited and falls short of expectations. Illustrating this reluctance are the International Federation of Robotics data, showing that the share of cobots over the past five years, although gradually growing, amounted to just 10 % of total robot installations.
   One key challenge is the complexity of ensuring safety with cobots, which involves integrating technical, human, and organizational factors. This creates a contradiction in cobot safety, leaving practitioners uncertain about deployment. In fact, safety in industrial environments is often seen as conflicting with productivity, but modern approaches suggest that safety should not hinder performance. Instead, performance should be optimized within safety constraints, focusing on a balanced approach that also considers worker well-being.
2) **Narrow safety focus and overlooked system-wide impacts.**
   Safety considerations in human-robot collaboration (HRC) have traditionally focused on physical risks, such as collision avoidance and compliance with ISO standards (e.g., ISO 15,066). However, this narrow focus often overlooks broader system-wide impacts, including psychosocial safety and operator well-being, which are equally critical in dynamic collaborative work environments. The proximity and

interaction between humans and cobots introduce additional complexities that cannot be fully addressed through technical safeguards alone.

3) **Limited guidance on the precise drivers of well-being.**

Industry 5.0 builds on the human-centric aspects of Science and Technology Studies (STS) by promoting technologies that enhance human well-being and facilitate meaningful human machine collaboration. Industry 5.0 explicitly incorporates well-being as a core principle, advocating for the development of technologies that support sustainable, resilient, and human-centered industrial systems.

Well-being in human-robot collaboration (HRC) is framed within ergonomics and human factors, which include physical and cognitive aspects. Safety is a core component of well-being, ensuring that robots operate safely alongside humans. This includes addressing both physical safety and cognitive and emotional impacts. While research has been conducted on ergonomics and human factors in HRC, there is limited practical guidance on implementing well-being measures and on the precise drivers of well-being in HRC [5].

1.3 Sliding manipulators

Sliding manipulators are usually articulated manipulators with six or more degrees of freedom characterized by a prismatic joint as their first joint or positioned on a track. In the first case, they are redundant structures whose mobility in terms of translation is allowed by the initial prismatic joint; in the second case, they are a structure made redundant by the addition of the degree of translation determined by the mobile base (as shown in Figure 1.3.0).

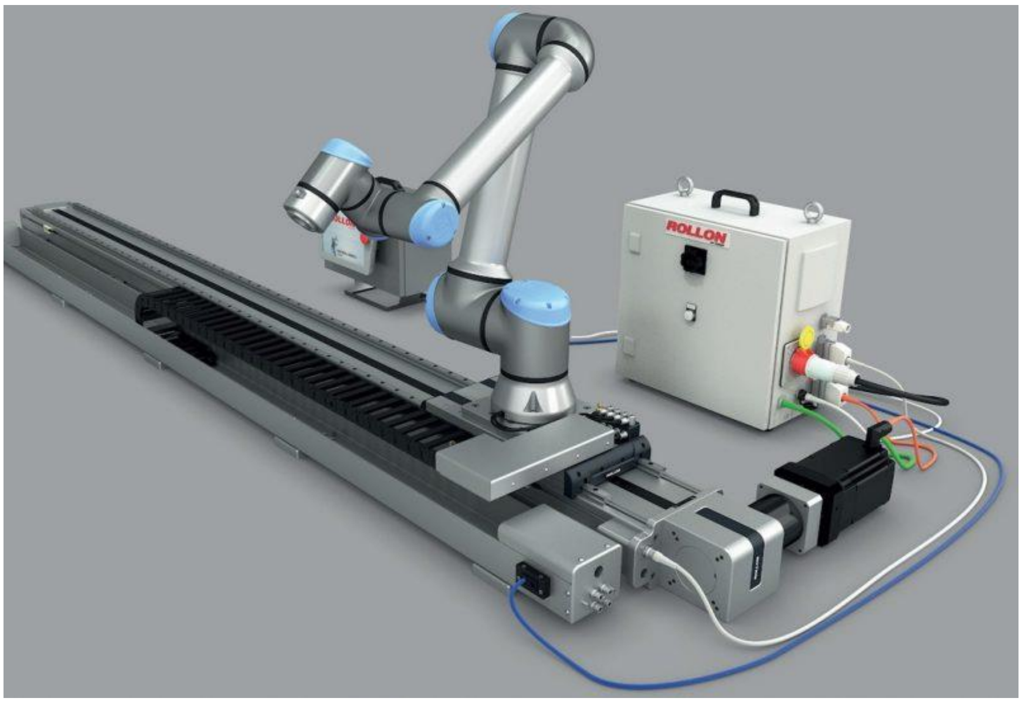

Figure 1.3.0 - A sliding manipulator [6]

In both cases, these structures have the typical advantages of both a redundant device and a mobile device: large working space, light weight, high fault tolerance, and high operating accuracy. In detail, redundancy results in: (i) greater dexterity or the ability to perform tasks increased manipulability, (ii) limitation of joint speeds, (iii) uniform distribution of motion on the joints, (iv) minimized consumption in the optimization of task execution times, (v) increased reliability with respect to failures, and (vi) avoidance of obstacles and/or singularity configurations. On the other hand, mobility allows for better maneuverability and control capacity as well as an increase in the overall workspace of the manipulator compared to a traditional manipulator with a fixed base [2].

Despite the several advantages brought by the use of a redundant sliding robotic system, its integration inside a production system needs some aspects to take into account:

1) **Static load capacity**: The base of the robot must be able to withstand the static load (consisting of the weight of the robot and the load, the bending, and the overall center of gravity under static conditions) so as to prevent the structure from tipping over.
2) **Dynamic load capacity**: in addition to static conditions, the base must be properly sized taking into account the loads that develop when the robot performs the planned trajectory and, in addition, it is necessary to appropriately size the stops to prevent the robot from leaving the guide.
3) **Speed and acceleration**: it is necessary to choose and appropriately size the robot actuator according to the speeds and loads required during the execution of the tasks.
4) **Precision and accuracy**: in the case of several robots positioned on the same guide, it may be necessary to integrate support beams to comply with the requirements of precision and repeatability during the execution of the operations as well as to provide reference systems for calibrating the position of the robot on the guide itself [2].

1.4 Aim of the thesis

Currently, literature research concerning sliding manipulators does not focus on HRI.

On the contrary, in Scopus database, it is possible to find scientific papers in which robots are used in industrial environments such as [7] and [8], where the sliding manipulator is employed for agricultural applications. Other studies such as [9] and [10] focus on the analytical aspects of the redundancy.

In the market too, sliding cobots are mainly built for applications where the pHRC is not contemplated. The "SLIDEKIT 2.0" of Universal Robots, for instance, is able to perform operations, e.g. welding, palletising, finishing, machine tending, and parts inspection [11].

The first full collaborative solution is the "Movotrak Cobot transfert unit" of Universal Robots, provided by the manufacturer "Thompson linear" (Figure 1.4.0) [12]. Its appearance on the market (January 2025) is very recent with respect to the writing of this thesis.

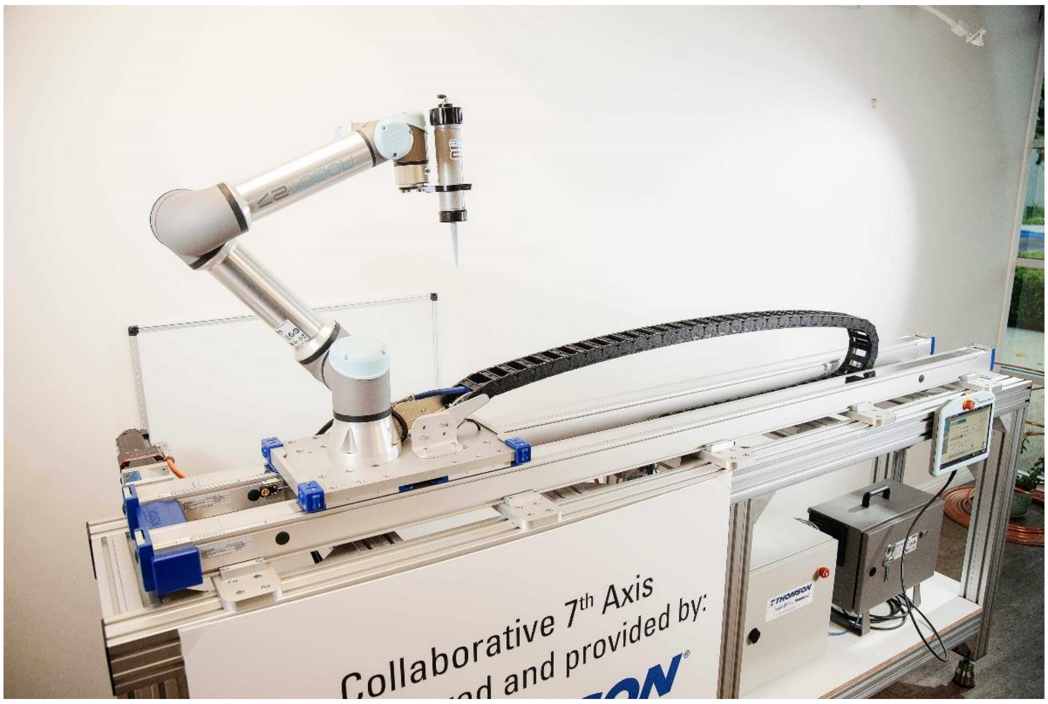

Figure 1.4.0 - Movotrak Cobot transfer unit [11]

This solution is truly collaborative according to ISO safety standards, for its effective collision detection inside the base, which embeds a load sensor that stops the whole robot when triggered. In other 7th axis manipulators, the cobot is able to translate only when the collaborative mode is turned off.

Accordingly, to fill the gap in the academic literature on this topic, the aim of this master thesis is to evaluate the integration of a linear guide under the base of a collaborative robot (UR3, Universal Robot). Specifically, the thesis focuses on the computational analysis through Matlab and Simulink of the complex kinematics of a redundant robot without modeling the robot dynamics. In the following chapters, the mathematical procedures for resolving the kinematics are explained, modeled and traduced for specific tasks. Subsequently, the obtained movements of the robot are observed though a simulation environment in Simscape. Finally, results are presented and discussed, highlighting the advantages of the linear guide insertion for a collaborative scenario.

The chronological workflow that has been following is shown in Figure 1.4.1.

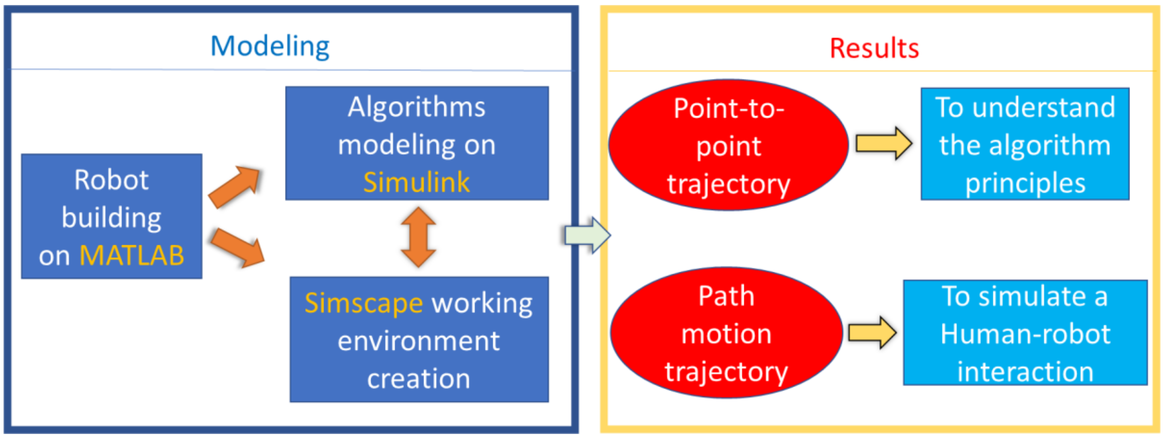
Modeling
Robot building on MATLAB
Algorithms modeling on Simulink
Simscape working environment creation
Results
Point-to-point trajectory
To understand the algorithm principles
Path motion trajectory
To simulate a Human-robot interaction

Figure 1.4.1 - Thesis workflow

MODEL

In this chapter, in the first section, the analytical problem of the inverse kinematics for a redundant manipulator is presented. In the second section, the Simulink model implementing the theoretical part previously described is proposed.

2.1 Inverse kinematics

A manipulator is defined kinematically redundant when it has a number of degrees of freedom (DOFs) n that is greater than the number of variables necessary to describe a given task m. Therefore, redundancy is a relative concept: a manipulator can be redundant with respect to a task and non-redundant with respect to another task. With reference to the above-defined spaces, a manipulator is intrinsically redundant when the dimension of the operational space is smaller than the dimension of the joint space ($m < n$) [13]. This is the case of the present thesis, because the linear and angular velocities of the End-Effector (EE) are always defined ($m = 6$), and there are 7 DOFs (6 revolute UR3 joints + prismatic guide joint).

The redundancy implies a Jacobian matrix ($\mathbf{J}(\mathbf{q})$) with more columns than rows (6x7), so there are infinite joint velocities $\dot{\boldsymbol{q}}$ (7x1 vector) that can generate a desired EE velocity $\boldsymbol{v}_d$ (6x1 vector). Hence, unlike non-redundant robots, the simple inversion of the following differential kinematics equation is no longer possible.

$$\boldsymbol{v}_d = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.00}$$

Closed-form solutions, which use direct kinematics for the inverse kinematics problem, cannot be used since infinite solutions exist. Therefore, numeric solutions techniques must be implemented, and this naturally leads to starting the problem from (2.00). This is one of the reasons why the trajectory must be also defined in velocity terms.

In order to deal with the redundancy, a viable solution method is to formulate the problem as a constrained linear optimization problem. The values of $\boldsymbol{v}_d$, an initial joint configuration $\boldsymbol{q}(0)$, and the discrete time $t = k \cdot T_s$, where k is a positive integer number, and T_s is the sampling time, are given.

By instantaneously optimizing an objective function $H(q, \dot{q})$ that generally depends on the joint configuration and velocities, it is possible to compute the discrete joint velocities $\dot{\boldsymbol{q}}(k \cdot T_s)$; by integrating them with a numeric method, like the simple Euler's one, the configurations of the next time step are obtained as shown [13]:

$$\boldsymbol{q}\big((k+1) \cdot T_s\big) = \boldsymbol{q}(k \cdot T_s) + T_s \cdot \dot{\boldsymbol{q}}(k \cdot T_s) \tag{2.01}$$

From this result, the joint velocities of the current time step are solved, and so on for each time step by iterating the method, until the last sampled time.

These are the so-called *local methods* used in this thesis. The *global methods*, in which the optimization of the objective function is done throughout the whole trajectory time, are more complex. For this reason, their numeric solutions are computationally intensive, therefore they cannot be solved online like the local methods [14].

One of the simplest objective functions is the quadratic cost function of joint velocities (2.02), where $\boldsymbol{W}$ is a suitable $(n \times n)$ symmetric positive definite weighting matrix.

$$H(\dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{W} \dot{\boldsymbol{q}} \tag{2.02}$$

Thus, once the EE velocity and Jacobian matrix are available for a given configuration, it is desired to find solutions $\dot{\boldsymbol{q}}$ that satisfy the linear equation in (2.00) and minimize (2.02). This minimization can be obtained with the *method of Lagrange multipliers,* and, under the assumption that the Jacobian has full (row) rank, the solution achieved is the following one:

$$\dot{\boldsymbol{q}} = \boldsymbol{W}^{-1}\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^T)^{-1}\boldsymbol{v}_d \tag{2.03}$$

Where the matrix $J_W^{\#}$ is the *right weighted pseudo-inverse* of $\boldsymbol{J}$:

$$\boldsymbol{J}_W^{\#} = \boldsymbol{W}^{-1}\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^T)^{-1} \tag{2.04}$$

A particular case occurs when the weighting matrix W is the identity matrix. In this case, the solution simplifies into

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\boldsymbol{v}_d \tag{2.05}$$

Where the matrix $\boldsymbol{J}^{\#}$ is the *right pseudo-inverse* of $\boldsymbol{J}$.

$$\boldsymbol{J}^{\#} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T)^{-1} \tag{2.06}$$

The obtained solution locally minimizes the norm of joint velocities, meanwhile (2.03) minimizes the weighted norm [13].

If $\boldsymbol{J}$ is not full rank, then the pseudo-inverse is numerically computed using the Singular Value decomposition (SVD) of $\boldsymbol{J}$[14].

The SVD is a factorization of a matrix into a rotation, followed by a rescaling followed by another rotation. It generalizes the eigendecomposition of a square normal matrix with an orthonormal eigenbasis to any $(m \times n)$ matrix. Specifically, the singular value decomposition of an $(m \times n)$ real matrix $\boldsymbol{M}$ is a factorization of the form $\boldsymbol{M} = \boldsymbol{U\Sigma V}^T$,where $\boldsymbol{U}$ is a $(m \times m)$ real orthogonal matrix, $\boldsymbol{\Sigma}$ is a $(m \times n)$ rectangular diagonal matrix with non-negative numbers on the diagonal and $\boldsymbol{V}$ is $(n \times n)$ real orthogonal matrix. The diagonal entries $\sigma_i = \Sigma_{\mathrm{ii}}$ of $\boldsymbol{\Sigma}$ are uniquely determined by $\boldsymbol{M}$ and they are known as the singular values of $\boldsymbol{M}$. The number of non-zero singular values is equal to the rank of $\boldsymbol{J}$ [15].

Therefore, the rank decreases when the last singular value approaches zero. Since the determinant of $\boldsymbol{J}$ does not exist, the monitoring of this parameter can be used instead as a singularity detection.

By applying this decomposition to the Jacobian matrix, it can be proved that its numerical pseudo-inverse $\boldsymbol{J}^{\#}$ is obtained as it follows:

$$\boldsymbol{J}^{\#} = \boldsymbol{V\Sigma}^{\#}\boldsymbol{U}^T \tag{2.07}$$

Where $\boldsymbol{\Sigma}^{\#}$ is a $(n \times m)$ rectangular diagonal matrix with the diagonal entries being the reciprocal of σ_i [14]. An alternative solution overcoming the problem of inverting differential kinematics in the neighborhood of a singularity is provided by the so-called *damped least-squares (DLS) inverse*, expressed in the following equation:

$$\boldsymbol{J}_{DLS} = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T + \mu^2\boldsymbol{I})^{-1} \Rightarrow \dot{\boldsymbol{q}} = \boldsymbol{J}_{DLS}(\boldsymbol{q})\boldsymbol{v_d} \qquad (2.08)$$

Where μ is a damping factor. It can be shown that such a solution can be obtained by reformulating the problem in terms of the minimization of the cost functional:

$$H(\dot{\boldsymbol{q}}) = \frac{\mu^2}{2}||\dot{\boldsymbol{q}}||^2 + \frac{1}{2}||\boldsymbol{v}_d - \boldsymbol{J}\dot{\boldsymbol{q}}||^2 \qquad (2.09)$$

This method induces a robust behavior when crossing singularities, but in its basic version always gives a task error proportional to the damping factor [12].

$$||\boldsymbol{v}_d - \boldsymbol{J}\dot{\boldsymbol{q}}|| = \mu^2(\boldsymbol{J}\boldsymbol{J}^T + \mu^2\boldsymbol{I})^{-1}\boldsymbol{v}_d \qquad (2.10)$$

Hence, it is a compromise between large joint velocity occurring in the proximity of a singularity and task accuracy. Ideally, μ should be variable depending on the value of the last singular value. Accordingly, the damping effect appears only when it is really needed, and the task error is confined to just a part of the trajectory [14].

Finally, the redundancy allows robot *self-motion* in which joint velocities do not contribute to the EE motion. These internal joint displacements can be chosen so as to improve/optimize the behavior of the robot.

Analytically, a *null* space of $\boldsymbol{J}$ exists. It is the subspace $\mathcal{N}(\boldsymbol{J})$ in $\mathbb{R}^n$ of joint velocities that do not produce any EE velocity in the given manipulator posture. In our case study of intrinsically redundant robot, and if the Jacobian has full rank, the dimension of $\mathcal{N}(\boldsymbol{J})$ is equal to one, but it can increase if a singularity happens.

Thus, self-motions can contribute along with the pseudo-inverse (or weighted-pseudo inverse) solution as demonstrated in the following equation:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\boldsymbol{v}_d + (\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})\dot{\boldsymbol{q}}_0 \tag{2.11}$$

The second term of the right member of (2.11) includes all the possible solutions of the associated homogeneous equation $\boldsymbol{J}\dot{\boldsymbol{q}}_0 = 0$ (self-motions), where: $\dot{\boldsymbol{q}}_0$ is a vector of arbitrary joint velocities, and the matrix that pre-multiplies it is a projector in the null space of $\boldsymbol{J}$. This equation can be deduced from the minimization of the norm vector $\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0$, as described in the following new cost function:

$$H(\dot{\boldsymbol{q}}) = \ \|\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0\|^2 \tag{2.12}$$

In other words, the goal is to find solutions that satisfy the constraint (2.00) while remaining as close as possible to $\dot{\boldsymbol{q}}_0$. Hence, the objective specified through $\dot{\boldsymbol{q}}_0$ becomes unavoidably a secondary objective to satisfy with respect to the primary objective specified by the constraint (2.00) [13].

Finally, it is important to discuss the way to specify the vector $\dot{\boldsymbol{q}}_0$ for a convenient utilization of redundant DOFs. A typical choice is the *Projected Gradient*:

$$\dot{\boldsymbol{q}}_0 = \boldsymbol{k}_0 \left(\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}}\right)^T \tag{2.13}$$

Where $\boldsymbol{k}_{0,i} > 0$ and $w(\boldsymbol{q})$ is a secondary function of the joint variables. Since the solution moves along the direction of the gradient of the objective function, it attempts to maximize it locally in compliance with the primary objective (kinematic constraint). The simplest objective function and the one used in the present thesis is known as *"distance from mechanical joint limits"*, which is defined as:

$$w(\boldsymbol{q}) = -\frac{1}{2n}\sum_{i=1}^{n}\left(\frac{q_i - \bar{q}_i}{q_{i,M} - q_{i,m}}\right)^2 \tag{2.14}$$

Where $q_{i,M}(q_{i,m})$ denotes the maximum (minimum) joint limit and $\bar{q}_i$ the middle value of the joint range; thus, by maximizing this distance, redundancy is exploited to keep the joint variables as close as possible to the center of their ranges [13].

Even if they are simple and effective, there are some limits to these Jacobian-based methods. There is no guarantee that singularities are *globally* avoided during task execution; despite joint velocities are kept to a minimum, this is only a *local* property and "avalanche" phenomena may occur. Also, cyclic motions in *task space* do not map to cyclic motions in *joint space*, so motion are *non-repeatable* in the joint space [14].

2.1.1 Inverse kinematics algorithm

Considering the numerical techniques to solve (2.00), by using a discrete-time implementation like shown in (2.01), the reconstruction of joint variables $\boldsymbol{q}$ is entrusted to a numerical integration which involves *drift* phenomena of the solution. Consequently, the end-effector pose corresponding to the computed joint variables differs from the desired one. This inconvenience can be overcome by adopting a solution scheme that takes into account the *operational space error* between the desired and the actual EE position and orientation. Let $\boldsymbol{x}$ be the 6x1 vector that contains the position and the three Euler angles, the error is expressed as [13]:

$$\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x}_e \tag{2.15}$$

Now, the time derivative of (2.15) is obtained as it follows:

$$\dot{\boldsymbol{e}} = \boldsymbol{v}_d - \boldsymbol{v}_e \tag{2.16}$$

However, instead of using the Euler angles velocity, the angular velocities are preferred, as it will be explained later in the paragraph; thus, according to differential kinematics (2.00), $\dot{\boldsymbol{e}}$ is obtained as:

$$\dot{\boldsymbol{e}} = \boldsymbol{v}_d - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.17}$$

For this equation to serve as the basis for an inverse kinematics algorithm, it is useful to relate the computed joint velocity vector $\dot{\boldsymbol{q}}$ to the error $\boldsymbol{e}$, so that (2.17) defines a differential equation describing the error evolution over time. Nonetheless, it is necessary to choose a relationship between $\dot{\boldsymbol{q}}$ and $\boldsymbol{e}$ that ensures convergence of the error to zero. Based on the assumption that the Jacobian matrix is square and nonsingular, the chosen relationship is:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{v}_d + \boldsymbol{K}\boldsymbol{e}) \tag{2.18}$$

By substituting (2.18) into (2.17), the equivalent linear system is obtained as:

$$\dot{\boldsymbol{e}} + \boldsymbol{K}\boldsymbol{e} = 0 \tag{2.19}$$

If $\boldsymbol{K}$ is a positive definite (usually diagonal) matrix, the differential system (2.19) is *asymptotically stable.* The error tends to zero along the trajectory with a convergence rate that depends on the eigenvalues of matrix $\boldsymbol{K}$. The larger the eigenvalues, the faster the convergence. The block scheme corresponding to the inverse kinematics algorithm in (2.18) is illustrated in Fig.2.10, where $\boldsymbol{k}(\cdot)$ indicates the direct kinematics function that starting from $\boldsymbol{q}$ provides $\boldsymbol{x_E}$ [13].

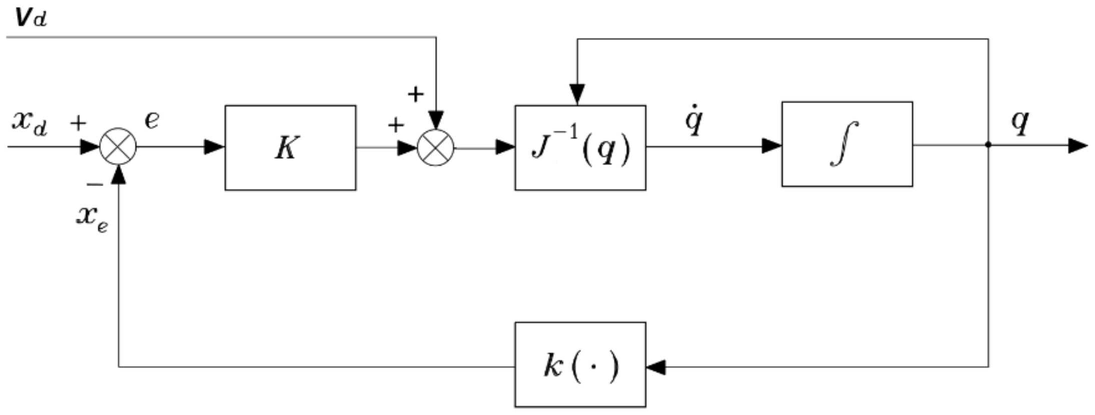

Figure 2.10 - Inverse kinematics algorithm with Jacobian inverse [8]

In the case of a *redundant manipulator,* solution (2.18) can be generalized into

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{v}_d + \boldsymbol{K}\boldsymbol{e}) + (\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})\dot{\boldsymbol{q}}_0 \tag{2.20}$$

which represents the algorithmic version of solution (2.11).

So far the structure of the inverse kinematics algorithm has been seen from a *planning* perspective but it can also be conceptually adopted for a simple robot *control* technique, known as *kinematic control*. Indeed, a manipulator is an electro-mechanical system actuated by motor torques, so *dynamic control* techniques are usually implemented to properly account for the nonlinear and coupling effects of the dynamic model. However, at first approximation, it is possible to consider a kinematic command as the system input, typically a velocity. This approximation is possible considering the presence of a low-level control loop, which 'ideally' imposes any specified reference velocity. On the other hand, such a loop already exists in a 'closed' control unit, where the user can also intervene with kinematic commands. In other words, the scheme in Fig. 2.10 can implement a kinematic control, considering that the integrator represents a simplified model of the robot, thanks to the presence of single joint local servos ensuring a more or less accurate reproduction of the velocity commands. Nevertheless, it is important to emphasize that this kinematic control technique provides satisfactory performance only when excessively fast motions or rapid accelerations are not required [13] and hence by considering the system as a *first-order model* [14].

Finally, concerning the operational space error, in regards to the *position error*, its expression is given by:

$$\boldsymbol{e}_P = \boldsymbol{p}_d - \boldsymbol{p}_e \tag{2.21}$$

Where $\boldsymbol{p}_d$ and $\boldsymbol{p}_e$ denote the desired and computed EE positions, respectively. Furthermore, its time derivative is:

$$\dot{\boldsymbol{e}}_P = \dot{\boldsymbol{p}}_d - \dot{\boldsymbol{p}}_e \tag{2.22}$$

On the other hand, for what concerns the *orientation error*, its expression depends on the particular representation of EE orientation, in terms of *Euler angles*, *angle and axis*, or *unit quaternion* [12]. The formers are of significant interest for kinematic structures having a spherical wrist; however, since this is not the case of the UR3 robot, they cannot be employed in this thesis, and this is the reason why they were not implemented in (2.17). The complexity of unit quaternions is not justified for this work, so the choice fell on the angle and axis.

If $\boldsymbol{n}, \boldsymbol{s}, \boldsymbol{a},$ are the unit vectors of the EE frame, $\boldsymbol{R}_d = [\boldsymbol{n}_d\ \boldsymbol{s}_d\ \boldsymbol{a}_d]$ denotes the desired rotation matrix of the EE frame and $\boldsymbol{R}_e = [\boldsymbol{n}_e\ \boldsymbol{s}_e\ \boldsymbol{a}_e]$ the rotation matrix that can be computed from the joint variables. The orientation error between the two frames can be expressed as:

$$\boldsymbol{e}_O = \boldsymbol{r} \sin\theta \tag{2.23}$$

Where θ and $\boldsymbol{r}$ identify the *angle and axis* of the equivalent rotation that can be deduced from the orientation matrix describing the rotation needed to align $\boldsymbol{R}_e$ with $\boldsymbol{R}_d$.

$$\boldsymbol{R}(\theta, \boldsymbol{r}) = \boldsymbol{R}_d \boldsymbol{R}_e^T(\boldsymbol{q}) \tag{2.24}$$

Notice that (2.24) gives a unique relationship for $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$. The angle θ represents the magnitude of an orientation error, and thus the above limitation is not restrictive since the tracking error is typically small for an inverse kinematics algorithm. It can be proved that (2.23) can be written as it follows [13]:

$$\boldsymbol{e}_O = \frac{1}{2}(\boldsymbol{n}_e(\boldsymbol{q}) \times \boldsymbol{n}_d + \boldsymbol{s}_e(\boldsymbol{q}) \times \boldsymbol{s}_d + \boldsymbol{a}_e(\boldsymbol{q}) + \boldsymbol{a}_d) \tag{2.25}$$

Estimating the derivative of (2.25) over time leads to the following equation:

$$\dot{\boldsymbol{e}}_O = \boldsymbol{L}^T \boldsymbol{\omega}_d - \boldsymbol{L}\boldsymbol{\omega}_e \tag{2.26}$$

Where

$$\boldsymbol{L} = -\frac{1}{2}\big(\boldsymbol{S}(\boldsymbol{n}_d)\boldsymbol{S}(\boldsymbol{n}_e) + \boldsymbol{S}(\boldsymbol{s}_d)\boldsymbol{S}(\boldsymbol{s}_e) + \boldsymbol{S}(\boldsymbol{a}_d)\boldsymbol{S}(\boldsymbol{a}_e)\big) \tag{2.27}$$

And $\boldsymbol{S}$ is the operator that transforms a vector into a skew-symmetric matrix.

At this stage, the global expression of the operational space error is:

$$\dot{\boldsymbol{e}} = \begin{bmatrix} \dot{\boldsymbol{e}}_P \\ \dot{\boldsymbol{e}}_O \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{p}}_d \\ \boldsymbol{L}^T \boldsymbol{\omega}_d \end{bmatrix} - \begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{L} \end{bmatrix} \boldsymbol{J}\dot{\boldsymbol{q}} \tag{2.28}$$

A subdivision of $\boldsymbol{K}$ into two 3x3 matrices is performed: one is related to the positional error $\boldsymbol{K}_P$, and one to the orientation error $\boldsymbol{K}_O$. Hence, the equation (2.20) becomes (2.29) [13]:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q}) \begin{bmatrix} \dot{\boldsymbol{p}}_d + \boldsymbol{K}_P \boldsymbol{e}_P \\ \boldsymbol{L}^{-1}(\boldsymbol{L}^T \boldsymbol{\omega}_d + \boldsymbol{K}_O \boldsymbol{e}_O) \end{bmatrix} + (\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})\dot{\boldsymbol{q}}_0 = \boldsymbol{J}^{\#}(\boldsymbol{q})\dot{\boldsymbol{r}} + (\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})\dot{\boldsymbol{q}}_0 \tag{2.29}$$

Where $\dot{\boldsymbol{r}} = \begin{bmatrix} \dot{\boldsymbol{r}}_v \\ \dot{\boldsymbol{r}}_\omega \end{bmatrix}$ is the velocity command, composed of two 3x1 vectors: its linear component $(\dot{\boldsymbol{r}}_v) = \left(\dot{\boldsymbol{p}}_d + \boldsymbol{K}_P \boldsymbol{e}_P\right)$, and its angular one $\dot{\boldsymbol{r}}_\omega = \boldsymbol{L}^{-1}(\boldsymbol{L}^T \boldsymbol{\omega}_d + \boldsymbol{K}_O \boldsymbol{e}_O)$.

2.2 Model implementation

The algorithm shown in Figure 2.10 is implemented in the Simulink model as it follows (Figure 2.20).

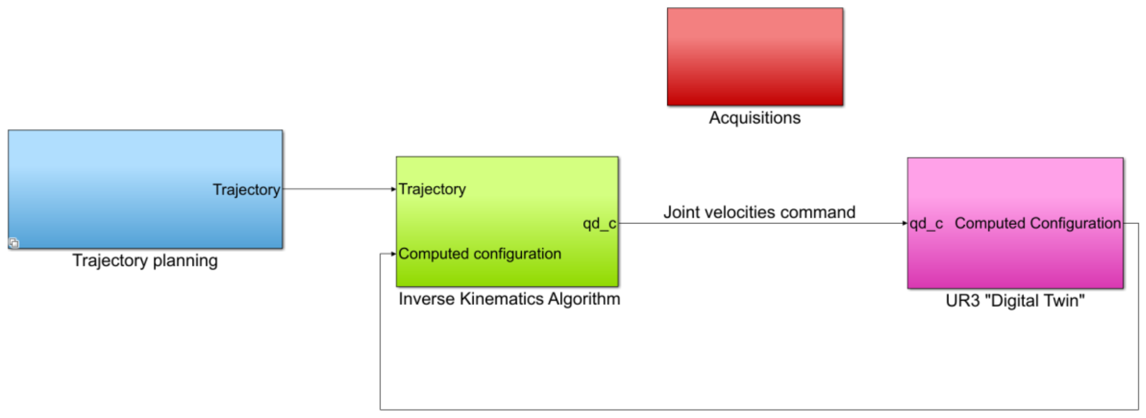

Figure 2.20 – Simulink model

The model has three main subsystems:

- The **Trajectory planning** subsystem allows the definition of the trajectory of the robot TCP in velocity and position/orientation terms. It is a variant subsystem with two options: point-to-point or path motion planning.
- The **Inverse Kinematics Algorithm** subsystem is the core of the model, in which the inverse kinematics is implemented, and the joint velocities are computed as the input to the following subsystem.
- The **UR3 "Digital twin"** subsystem contains the ideal and simplified robot behavior, from which the robot configuration is computed, that is needed for the inverse kinematics algorithm. It contains also post-processing results that do not interfere with the algorithm running.

The **Acquisitions** subsystem is solely created to recall all the *From* blocks related to the *Go to* blocks. The latter collects the internal outputs of interest, which, once recalled**,** are sent into the workspace so they can be used and analyzed. The background colors of all blocks are chosen according to the following color code: **green/yellow** for inputs/outputs of a subsystem, **orange** for constants or inputs defined before the model run, **cyan** for "Go to" blocks, **light blue** for blocks used in correspondence of a switch, so they operate as selectors, and their value is defined before the model run.

2.2.1 Trajectory planning

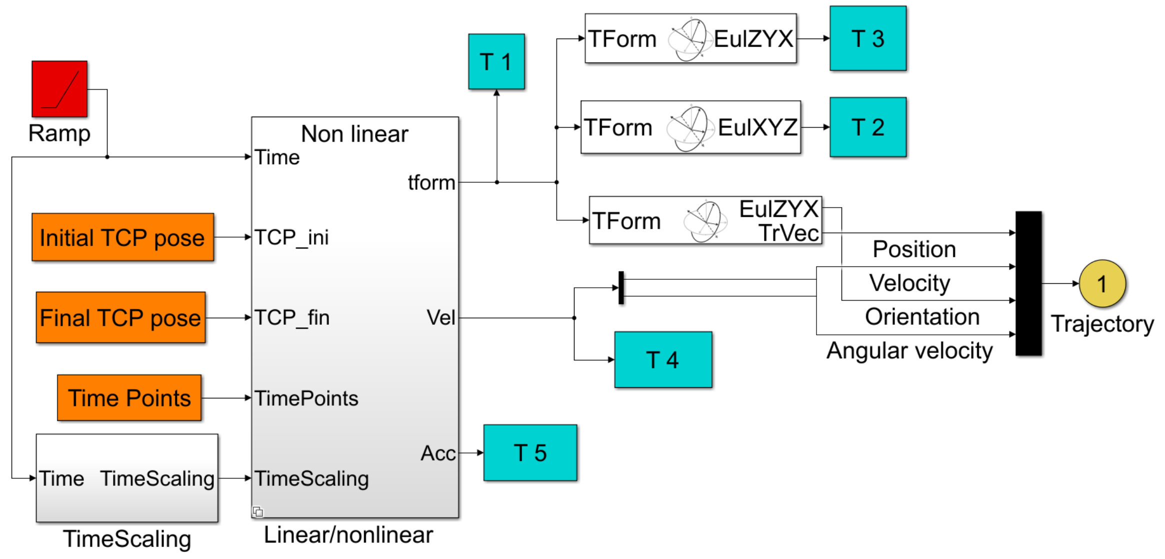

Figure 2.21 p-t-p planning subsystem

Figure 2.21 displays the point-to-point trajectory planning subsystem.

The provided inputs are the “Time Points” array, in which the trajectory starting and ending times are defined, and the initial and final TCP homogenous transformation matrix. Subsequently, the TCP velocity, acceleration, and matrix are computed through the “Transformation Trajectory” block from the “Robotics System toolbox”. This block is hidden in the “Linear-nonlinear” variant subsystem, which is activated depending on which time scaling is chosen. If the latter is requested, the time scaling is computed in the bottom left subsystem as demonstrated in Figure 2.22.

A trapezoidal velocity profile or a fifth polynomial trajectory can be chosen through a selector. For the latter, the boundary conditions are given as input. To assure that a possible numeric error does not exceed the limits (0 and 1), the local coordinate is saturated.

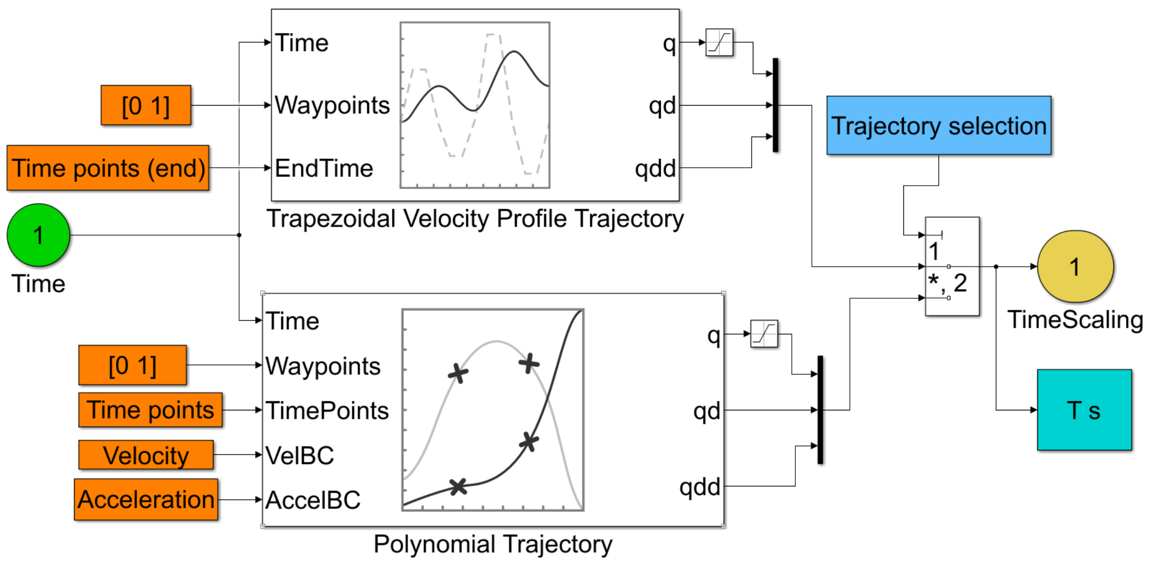

Figure 2.22 - Time scaling subsystem

Always considering the p-t-p subsystem, the position and body Euler angles “zyx”, which are collected along with the velocity and angular velocity in the “Trajectory” output, are extracted from the TCP matrix.

For the path motion type of planning, the layout was taken from an open MathWorks source [16], but it is similar to what it is shown for the point-to-point, so it is not presented. Indeed, the operating principle is not exactly a path motion but a repeated point-to-point. For this reason, a fixed time step is needed for the Simulink solver, while in the other case a variable time step is preferable. In both scenarios, an automatic Simulink solver was used. In this kind of planning, the inputs are the waypoints and their corresponding orientations, and the time instant in which each waypoint is reached.

2.2.2 Inverse kinematics algorithm subsystem

Except for the integrator, the scheme introduced in Figure 2.10 is in the main subsystem "Inverse Kinematics Algorithm". When opening this subsystem, the content of Figure 2.24 is obtained:

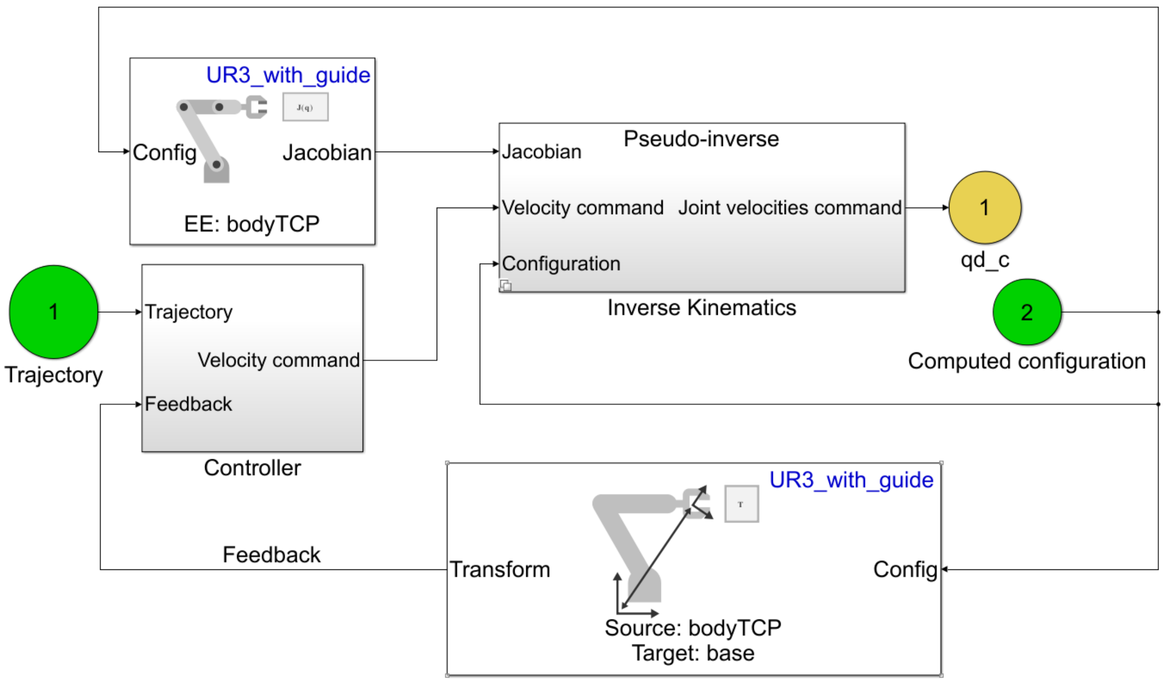

Figure 2.24 - "Inverse Kinematics Algorithm" subsystem

The Jacobian matrix is computed through the "Get Jacobian" Simulink block, and the direct kinematics function $\boldsymbol{k}(\cdot)$ is employed by the "Get Transform" block, which gives as output the TCP homogeneous transformation matrix that is used as feedback into the "Controller" subsystem shown in Figure 2.25.

In this final subsystem, equations (2.21), (2.22), (2.25), and (2.26) are solved, but they produce the velocity command only if closed-loop inverse kinematics is desired. Otherwise, in open-loop mode, the desired trajectory is instantly treated as the reference command. The choice is made a priori with the selector, and the two options cannot be switched during the simulation.

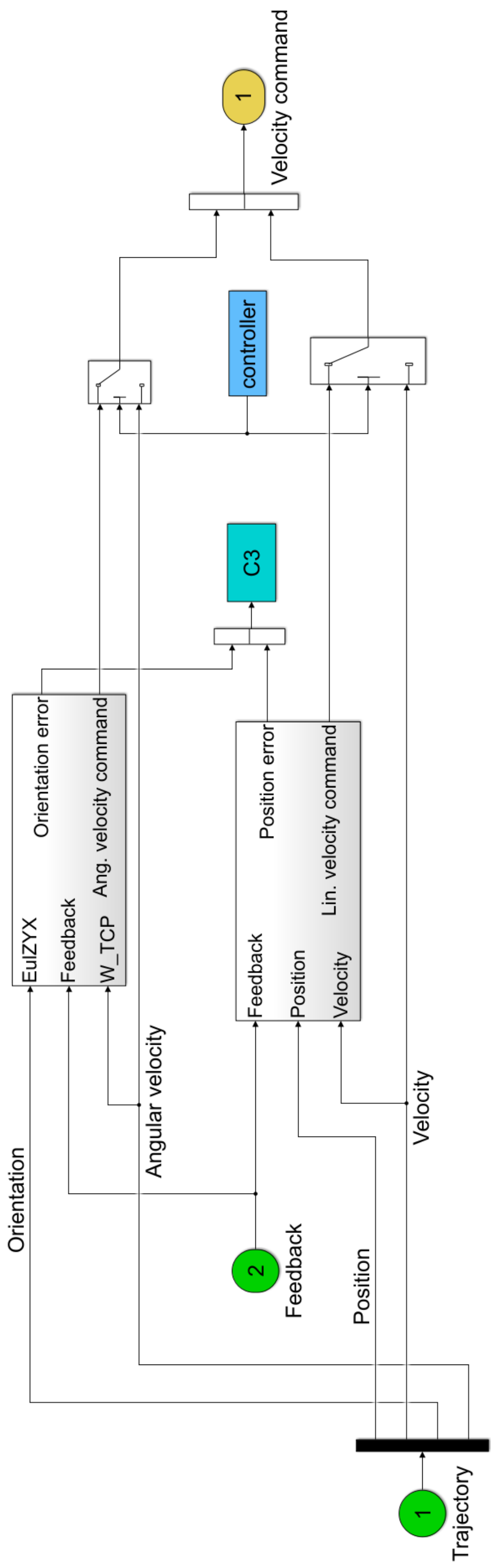

Figure 2.25 - "Controller" subsystem

The “Inverse Kinematics” block is a variant subsystem containing the three possible inverse analyzed to solve the problem: Pseudo-inverse, Weighted Pseudo-inverse and DLS inverse.

In Figure 2.24 the Pseudo-inverse is active, so its name appears on the top of the block; its contents are presented on Figure 2.26.

In this section, the equation (2.11) is re-written as (2.30) to have an efficient evaluation of the solution.

$$\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}_0 + \boldsymbol{J}^{\#}(\dot{\boldsymbol{r}} - \boldsymbol{J}\dot{\boldsymbol{q}}_0) \tag{2.30}$$

Three “Saturation” blocks are needed for the joint velocities to recreate a real robot behavior. The first is for the linear guide. Considering that it has a 1000 mm stroke, its maximum has been chosen to be equal to ±0.4 m/s for a proper human-robot collaboration. The second *Saturation* block groups the first three UR3 joints, which have a speed limit of ±180º deg/s. The third block is for the wrist joints, which have ±360 º deg/s as limits.

In the “Null space” subsystem, equations (2.13) and (2.14) are formulated. If the null solution term of (2.11) is not desired, it can be deactivated through the selector.

In the gradient function, joint limits are defined as follows:

$$\boldsymbol{q}_{i,M} = [0.500\ mm, 360°, 15°, 140°, 360°, 125°, 360°] \tag{2.31}$$

$$\boldsymbol{q}_{i,m} = -[0.500\ mm, 360°, 195°, 140°, 360°, 125°, 360°] \tag{2.32}$$

Especially for joints 3 (elbow) and 5 (second wrist joint) of the robot, the limits are stringent to avoid potential *self-collisions.*

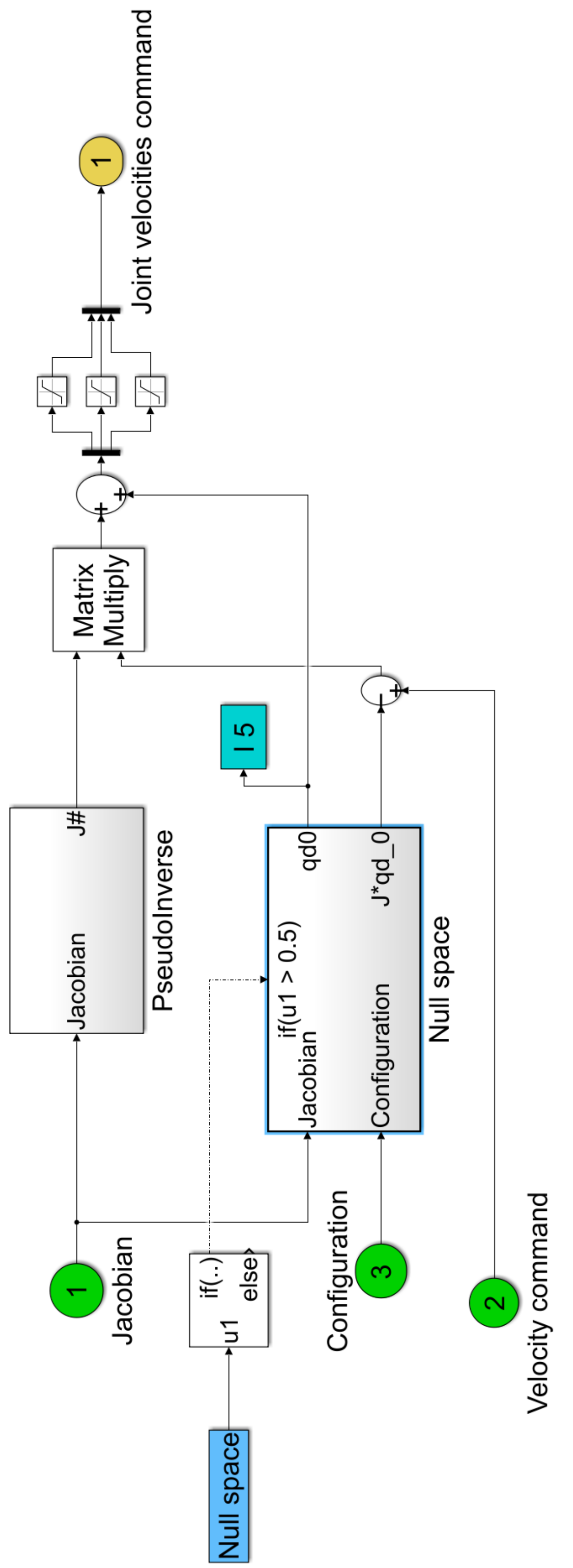

Figure 2.26 - Pseudo-inverse variant

Finally, in Figure 2.27, the Jacobian pseudo-inverse matrix is obtained, either analytically or numerically by using the “Pseudoinverse” Simulink block. The choice is made automatically depending on the size of the last singular value σ, which is extracted from the SVD algorithm executed through the “SVD” block. The size has been arbitrarily chosen to be equal to 0.05. Meanwhile, to monitor singularity proximity, the “Singularity counter” has an artificial method to count how many times σ goes below 0.1. When the singular value exceeds the threshold again, the counter is reset.

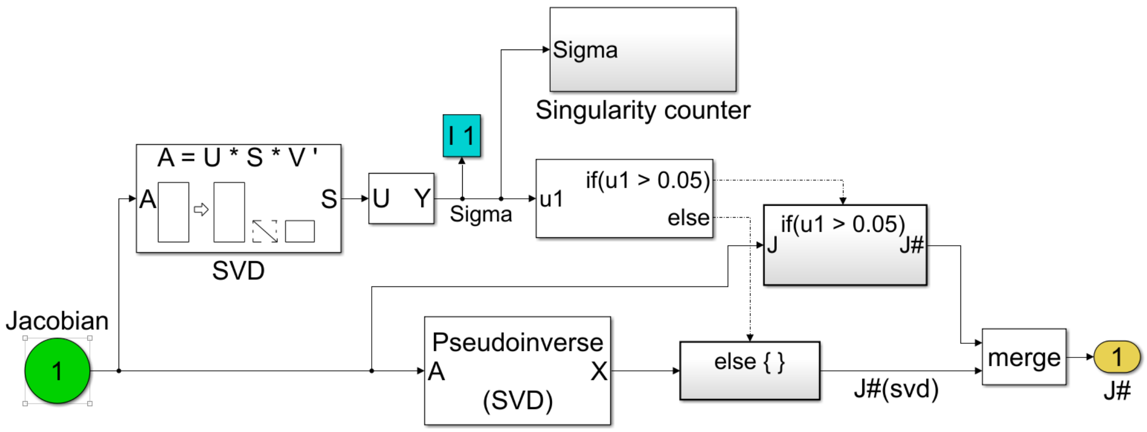

Figure 2.27 - “Pseudo-inverse “subsystem

The weighted pseudo-inverse and the DLS (the other variant subsystems) are not displayed because they are very similar to what has been demonstrated for the pseudo-inverse. However, just for clarification, it must be said that when the projected gradient method is applied to the DLS inverse, it creates a projection matrix $(\boldsymbol{I} - \boldsymbol{J}_{DLS}\boldsymbol{J})$ that has a component orthogonal to the null space of the Jacobian $\mathcal{N}(\boldsymbol{J})$. Hence, it would have some effects on the motion of the EE and *self-motions* in the proper term would be no longer available [17].

2.2.3 UR3 “Digital Twin”

The last main subsystem has two sub-blocks called *dynamics* and *post-processing dynamics*. The first is shown in Figure 2.28. It contains the integrator, where the robot is represented in its ideal (and approximative) first-order behavior.

The joint velocities command is given as input, and to reproduce joint inaccuracies or electromagnetic noises a disturbance signal can be added. Accordingly, the computed joint velocities differ from what is requested, like in a real trajectory. In this scenario, the kinematic control actively takes place in correcting the wrong EE pose resulted from the disturbance action.

Indeed, although the kinematic control was introduced to compensate the *drift* error derived from the numerical integration, its effect is relevant only when multiple cyclic motions are performed, and this is not of significant interest for this work. On the contrary, the control intervention it is much more evident and observable with an artificial disturbance.

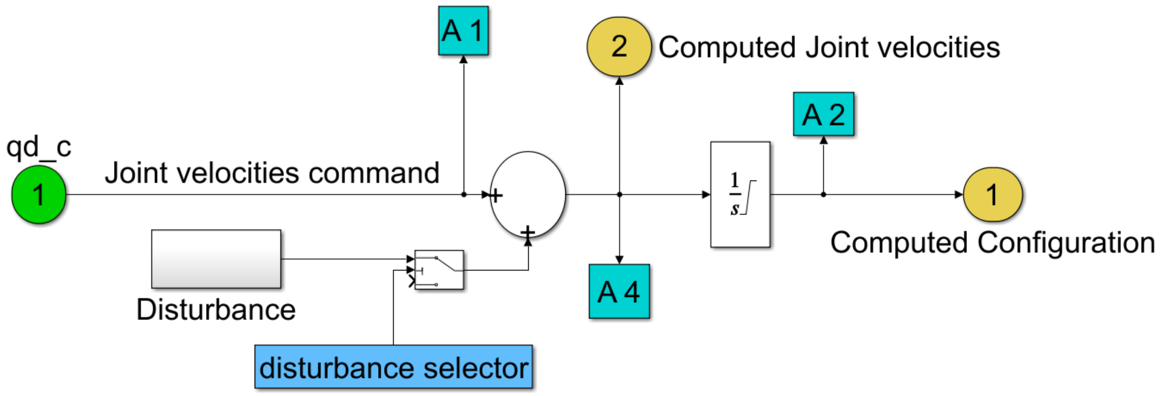

Figure 2.28 - "Dynamic" subsystem

In the integrator Simulink block, the initial configuration $\boldsymbol{q}(0)$ must be defined. By knowing and imposing specific initial EE pose and base position, it is possible to solve the inverse kinematics through an analytical method, just for the initial time step. In fact, by fixing a specific guide configuration, the redundancy ceases to exist.

Another useful way of exploiting the control is to intentionally introduce an incorrect initial configuration $\boldsymbol{q}(0)$ that produce an initial position/orientation EE error, as it will be demonstrated in the next chapter. In the integrator, outputs are also limited so they are not able to surpass the joint limits, the same ones defined in (2.31) and (2.32).

In the post-processing (Figure 2.29), joint torques and the guide force are computed based on joint configuration, velocity, and acceleration. For the static balance, only the first is needed. On the contrary, all of them are necessary for the inverse dynamics. In this case, the external force has been set to zero for all the robot bodies. Moreover, there is the obtained velocity of the TCP, solved from the known differential kinematics equation (2.00). This internal output can be compared to the desired velocity; hence, a task error can be analyzed. By deriving the velocity over time, the EE acceleration is obtained too.

All this procedure is possible thanks to the Simulink blocks: *Get Jacobian*, *Gravity Torque* and *Inverse Dynamics*.

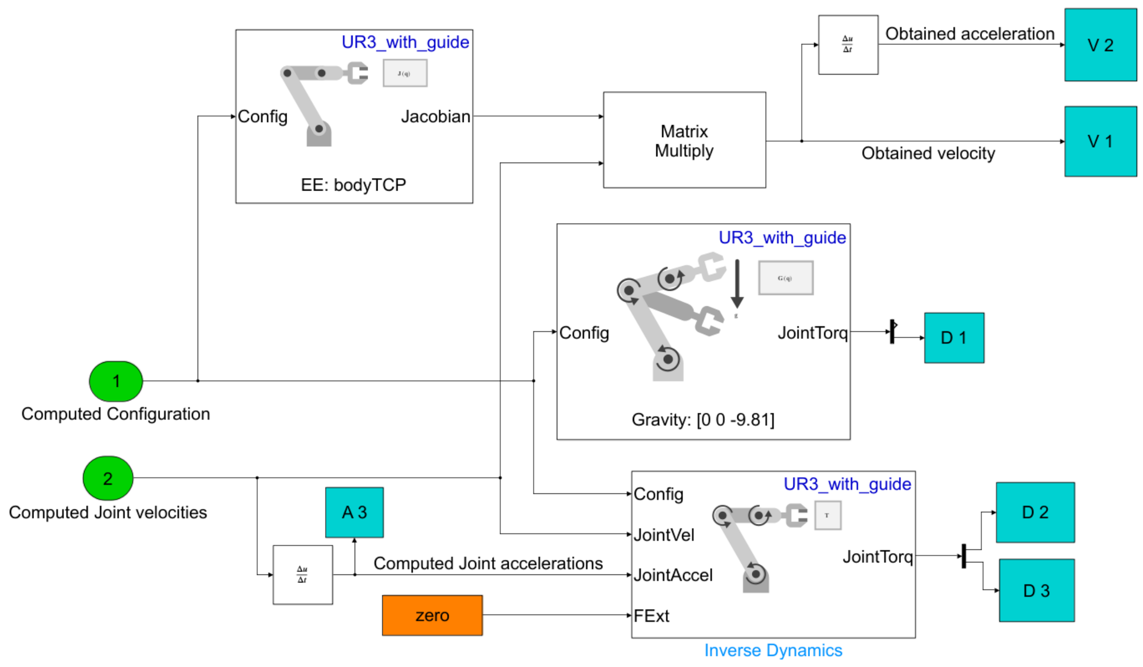

Figure 2.29 - "Post-processing Dynamics"

In conclusion, three additional considerations are needed.

First, the linear guide is placed horizontally, hence it does not contribute to the static balance. Second, the robot base is linked to the guide through a steel plate whose weight has been estimated to be roughly 4 Kg. Instead, its inertia is not of interest because the base can only translate. Therefore, in addition to the 11 Kg of the robot, the total moving weight is 15 Kg.

Thirdly, joint accelerations are only computed from a time derivative of joint velocities. However, they can be solved through the time differentiation of the differential kinematics equation (2.00), that leads to:

$$\boldsymbol{a}_d = \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \tag{2.33}$$

The equation (2.33) is the *second-order* differential kinematics equation in which also the EE acceleration $\boldsymbol{a}_d$ must be defined and given as input.

Under the assumption of a squared and non-singular Jacobian matrix, (2.33) can be inverted in terms of joint accelerations [13]:

$$\ddot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\big(\boldsymbol{a}_d - \dot{\boldsymbol{J}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}\big) \tag{2.34}$$

The numerical integration of (2.35) to reconstruct joint velocities and positions would unavoidably lead to a drift of the solution; hence a second-order algorithm is needed.

This kind of algorithm can be useful for *control* purposes when it may be necessary to invert a motion trajectory specified in terms of position, velocity, and acceleration. However, a first order differential kinematics is sufficient for a motion *planning* perspective, thus it was not implemented in this work.

RESULTS AND DISCUSSIONS

In this paragraph, a point-to-point and a path motion trajectory are presented. The former was ideated to understand the principles of the inverse kinematics algorithm, and the latter was intended to simulate a real Human-Robot Collaboration.

Since the robot does not have a mounted tool, the TCP frame is equal to the EE frame and hence the two terms will be used as synonyms.

The ground frame is in the middle of the table surface, while the robot base frame is 50 mm above it considering the sum of the guide and plate thicknesses. All the quantities are measured with reference to the ground.

During a Simscape simulation, to visually grasp when a position or orientation error is generated, a *physical* reference rgb frame is built to follow the ideal EE pose. When there is no error, this object appears as if it was mounted on the EE. Instead, the presence of an error is highlighted by a white *virtual* frame. on the TCP

Before introducing the trajectories, a brief explanation of the projected gradient method is presented.

3.1 Projected gradient method

In order to study the method, only self-motions are created by imposing a constant reference ($\boldsymbol{v}_e = 0$). Consequentially, by employing the pseudo-inverse, (2.11) becomes:

$$\dot{\boldsymbol{q}} = (\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})\dot{\boldsymbol{q}}_0 \tag{3.00}$$

Hence, the joint velocities match with the null space projection of $\dot{\boldsymbol{q}}_0$, a vector originated from the application of the gradient to the “distance from mechanical joint limits” objective function, as stated in (2.13) and (2.14).

A first robot pose is imposed, with an initial configuration (Figure 3.1.00) which permits self-motions originated from the application of the method to the guide and elbow.

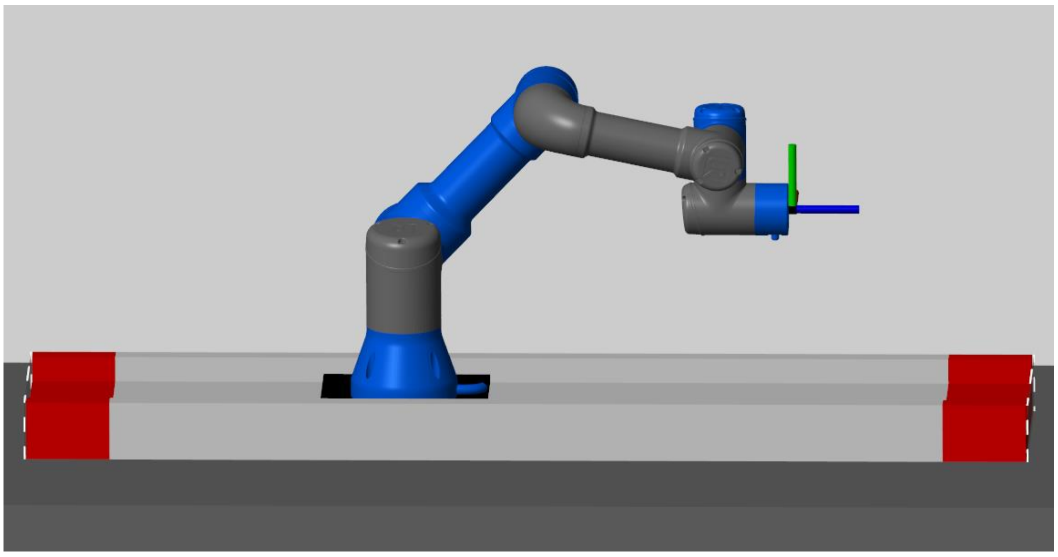

Figure 3.1.00 - Initial robot configuration

Indeed, the initial configuration provides a position of the base of 0.15 m, and the elbow starts with a value of 58.07 deg; in this way, the base and the elbow can move to reach their reference value $\bar{q}_i$, which is zero for both. By just activating the guide, its entry of the gain k_0 is the only one different from 0, with 100 as a value. The obtained $\dot{\boldsymbol{q}}_0$ is shown in Figure 3.1.01.

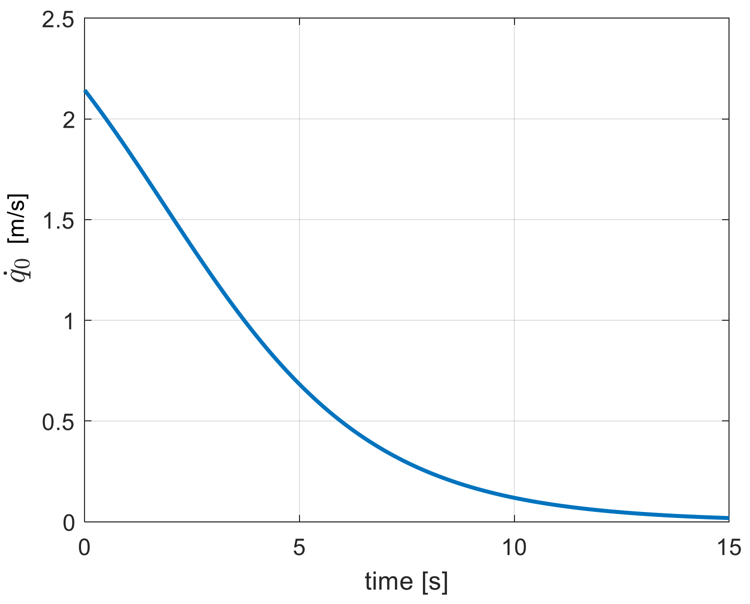

Figure 3.1.01 - Base velocity derived from its gradient. Guide gradient only.

Even if it seems that it exceeds the speed limit of 500 mm/s, the result must still be projected in the null space, as shown in Figures 3.1.02 and 3.1.03.

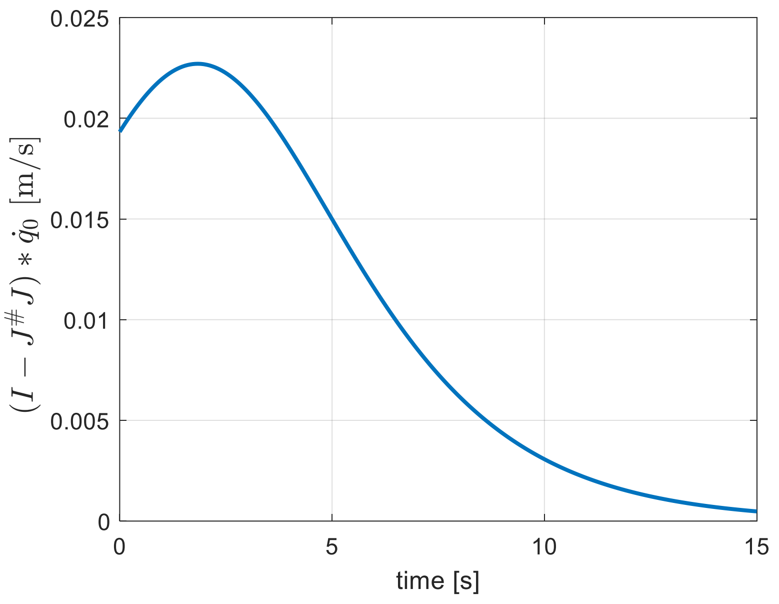

Figure 3.1.02 - Null space base velocity. Guide gradient only

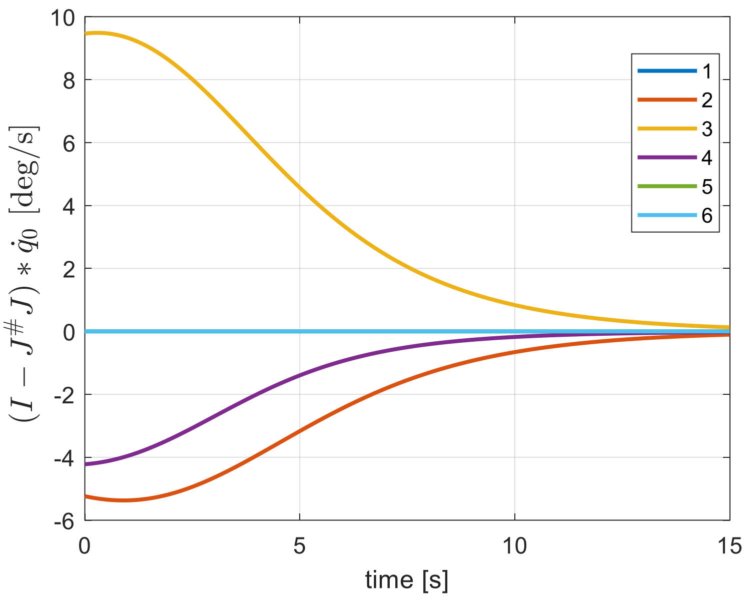

Figure 3.1.03 - Null space joint velocities. Guide gradient only

In this configuration, the null space projection activates not only the guide, but also the shoulder, the elbow, and the first joint of the wrist (q_4).

Of course, this is not unexpected because at least some other joints are needed to achieve self-motion. In this case, the elbow gets up (Figure 3.1.04) and the base reaches its centered value (Figure 3.1.05).

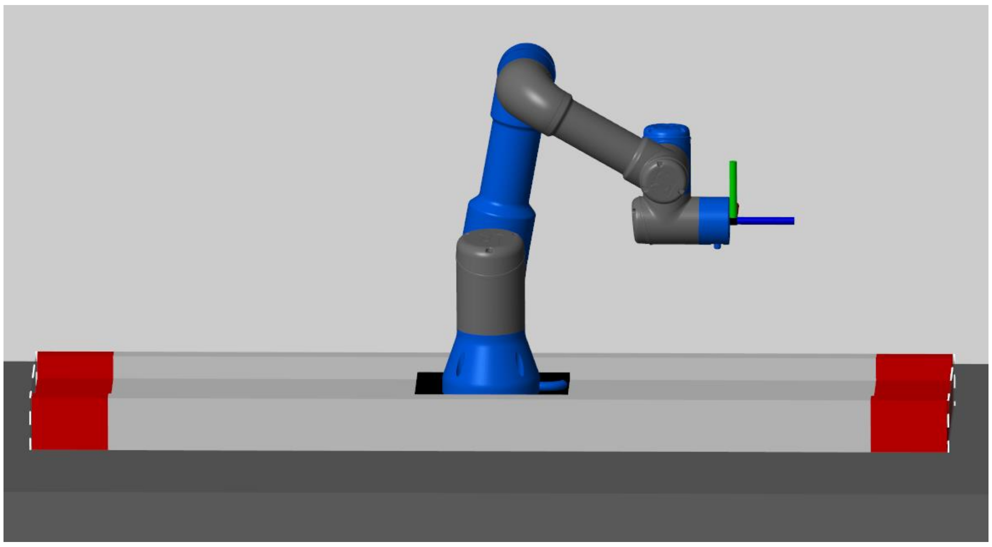

Figure 3.1.04 - Final robot configuration. q_{guide} = 0. Guide gradient only.

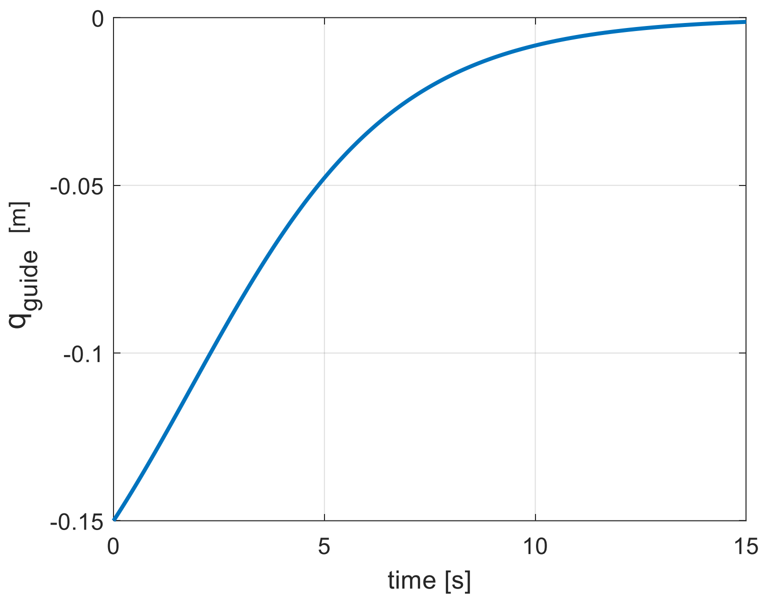

Figure 3.1.05 - Base position. Guide gradient only

Considering again Figures 3.1.01 and 3.1.05, it is clear that $\dot{\boldsymbol{q}}_0$ gets lower as the base approaches its reference value.

In this work, the reference value for k_0 will be equal to 100 even for revolute joints, because it ensures that the null space velocity term has a sufficient order of magnitude for its effect to be rapid and effective. For example, in Figure 3.1.02 the initial velocity is about 0.02 m/s; instead, through the calculation of the gradient alone ($k_0 = 1$), the velocity would become equal to 0.0002 m/s, and its effect would be irrelevant.

In this motion, the elbow gets closer to its limit. So, when it is the only one active, the resulting self-motion is the opposite of what was discussed before (Figure 3.1.06).

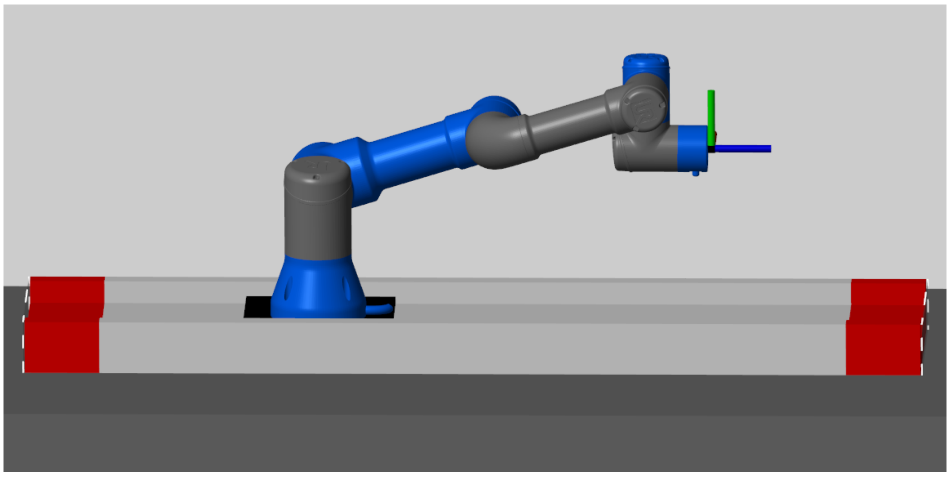

Figure 3.1.06 - Final robot configuration. q_3 = 0. Elbow gradient only

The base now moves to the left in the direction of one of its limits, while the elbow goes down, reaching the arm elongation that is associated with its null value (q_3 = 0).

If both are active, the spontaneous question is related to which joint prevails on the other. If both the gains are equal to 100, the elbow "wins" and the guide gradient is virtually non-existent (Figure 3.1.07). This could be explained by comparing the gradients (equation 3.01): the elbow begins with a value of 58.07 deg (1.013 rad) and it has a range of 280 deg (4.89 rad), while the guide starts with a value of 0.15 m and it has a range equal to 1.0.

$$\frac{0.15\, m}{(1\, m)^2} = 0.15 > \frac{1{,}013\, rad}{(4{,}89\, rad)^2} = 0.042 \tag{3.01}$$

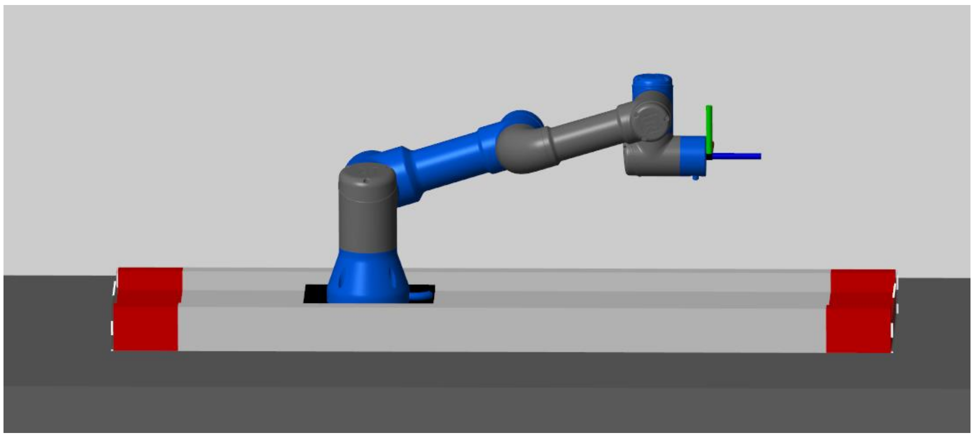

Figure 3.1.07 - Final robot configuration. q_3 = 0. Guide + elbow gradient

Surprisingly, even if the guide gradient is the greatest suggesting that its repulsive effect should be the predominating one, the opposite occurs. In fact, the projected base velocity gets low, as already happened before in Figures 3.1.01 and 3.1.02 where only the base gradient was computed. There, starting from approximately 2 m/s, it dropped sharply to 0.02 m/s. This does not happen for the elbow since it starts with (-)60 deg/s and, once projected, it starts with (-)23 deg/s (Figures 3.1.08 and 3.1.09).

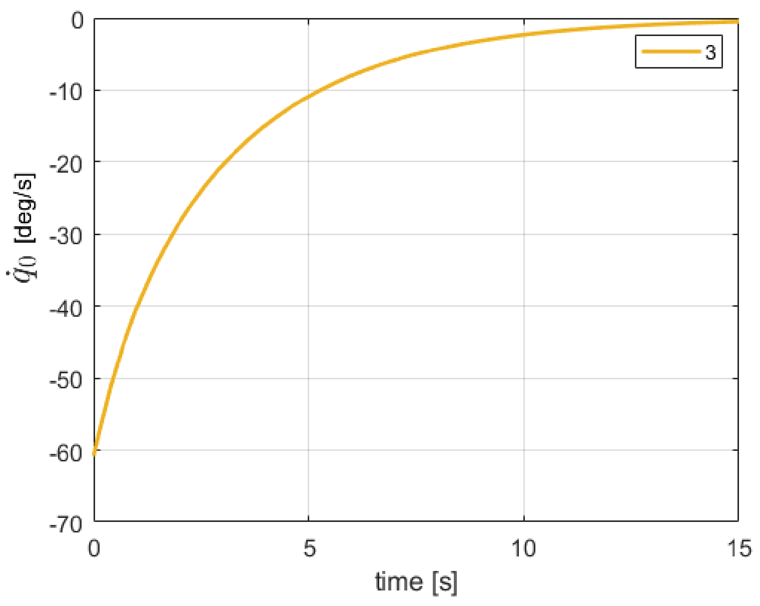

Figure 3.1.08 - Elbow joint velocity derived from its gradient. Guide + elbow gradient

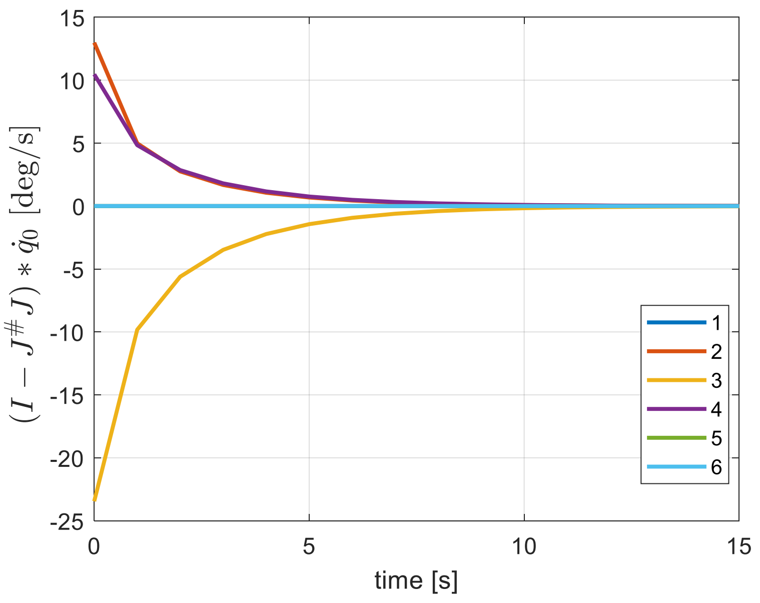

Figure 3.1.09 - Null space joint velocities. Guide + elbow gradient

The result cannot be easily predicted, because the null space projection matrix $(\boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J})$ is analytically hard to grasp. It ponders the easiness of obtaining all the desired $\dot{\boldsymbol{q}}_0$ in a given posture. Accordingly, for this particular configuration, it appears that the repulsive effect of the elbow limit is much stronger than the guide one.

In fact, if the projected gradient method is active for the joints q_1, q_5 and q_6, once their $\dot{\boldsymbol{q}}_0$ is projected in the null space, the resulting velocity is zero, because their motion would generate an orientation error. Consequently, they would violate the desired task and the definition of self-motion. This is demonstrated in Figures 3.1.10 and 3.1.11. Nevertheless, q_6 already equal to zero, its reference value, does not contribute.

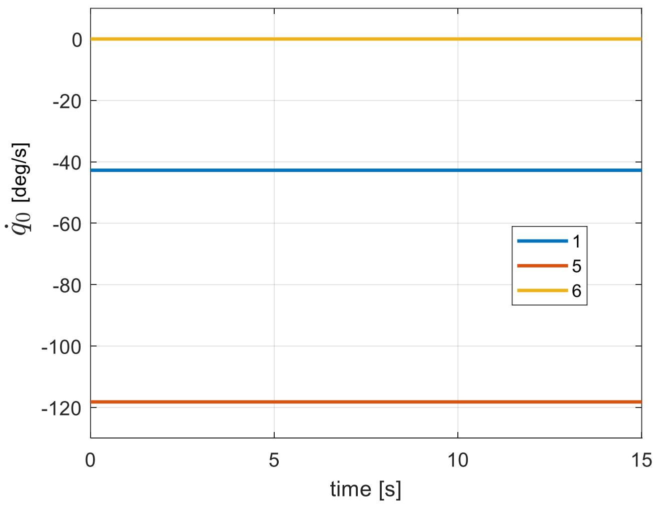

Figure 3.1.10 - Joint velocities derived from their gradient (q_1, q_5 and q_6)

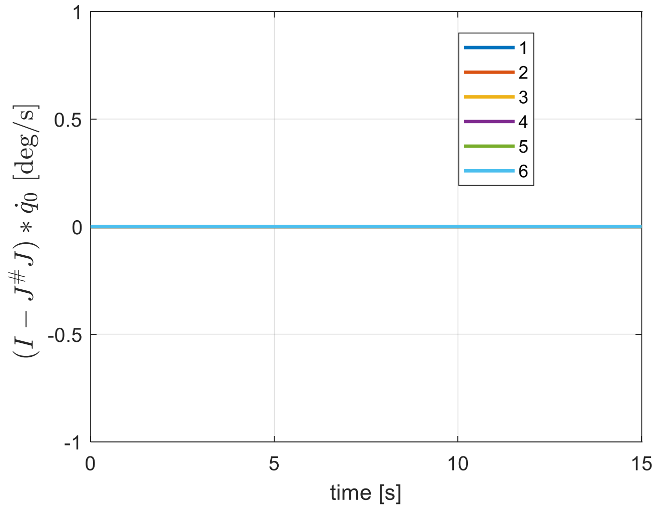

Figure 3.1.11 - Null space joint velocities. q_1 +q_5 +q_6 gradient

This example was proposed just to illustrate the method. In a real trajectory, which is the attained equilibrium or which reference value $\bar{q}_i$ is achieved does not really matter. What is requested from the projected gradient method is its repulsive effect from the limits, and overall the standard rule to understand which joint has the priority is the following one: the lower the joint range, the greater the gradient effect.

Hence, for q_1, q_4 and q_6, which have a wide range of motion of 720 deg, it is possible to neglect their proximity to the limits, since it will rarely occur. Indeed, it is undesirable for their activation to interfere with the self-motions generated from other joints with lower ranges. Accordingly, their gain has been put to zero, as if they would have an infinite range. An alternative solution could be to activate them anyway but with a minor order of magnitude like 10 or 1. For the sake of simplicity, the former solution has been chosen.

Reminding that 100 is a proper value to have an effective contribution of the null space velocities, the reference value for the gain $\boldsymbol{k}_0$ is defined as follows in equation (3.02):

$$\boldsymbol{k}_0 = 100 * [1\ 0\ 1\ 1\ 0\ 1\ 0] \tag{3.02}$$

3.2 Point to point trajectory

The trajectory is a fifth-degree polynomial with a linear path along the guide axis, constant x and z EE coordinates, and a fixed orientation. Thus, only a positive y linear TCP velocity is generated, as demonstrated in Figures 3.2.00 and 3.2.01. The initial and final boundaries are null both for the velocity and acceleration. The trajectory lasts 20 seconds.

The robot starts with a *left*-shoulder and wrist-*down* configuration, and the base is in its centered position, as demonstrated in Figure 3.2.02.

The question concerning this trajectory is related to how the robot will reach the final pose: only by the guide translation or also by an extension of the elbow.

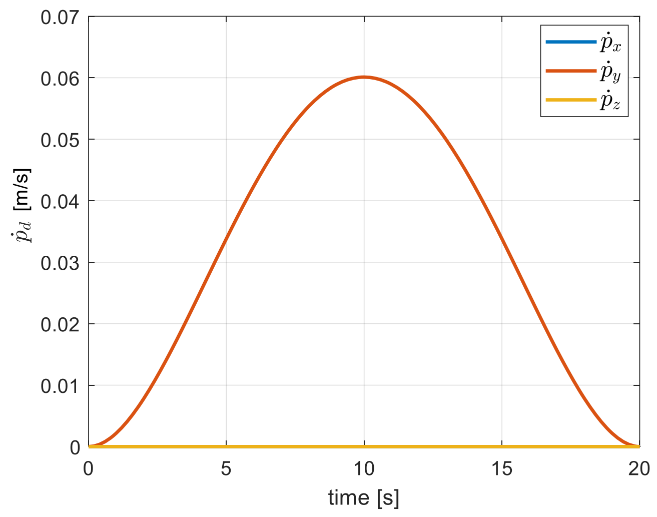

Figure 3.2.00 - Desired EE linear velocity

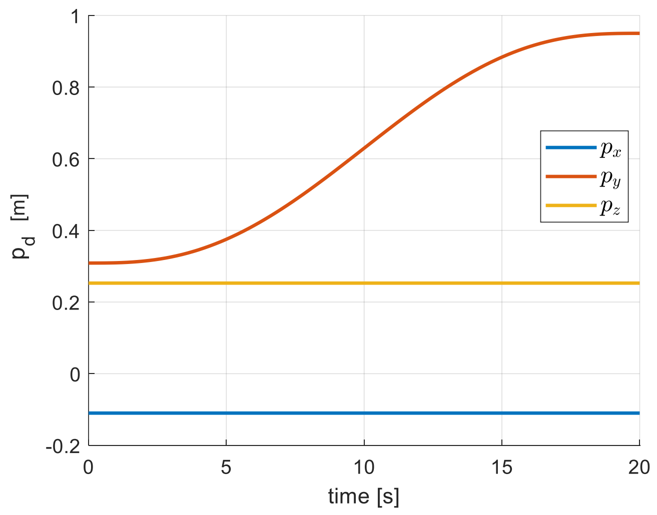

Figure 3.2.01 - Desired EE position

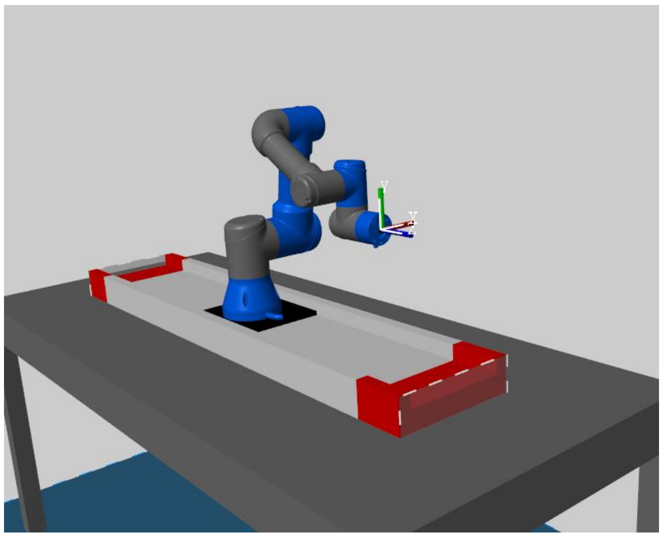

Figure 3.2.02 - Initial robot configuration

When employing the pseudo-inverse, since its cost function is to minimize the norm of the velocities, the final desired pose can be reached by only moving the base. However, it reaches its joint limit of 500 mm (Figure 3.2.03), and the EE is not yet in the right final pose. Hence, a position error is generated as shown in Figure 3.2.04.

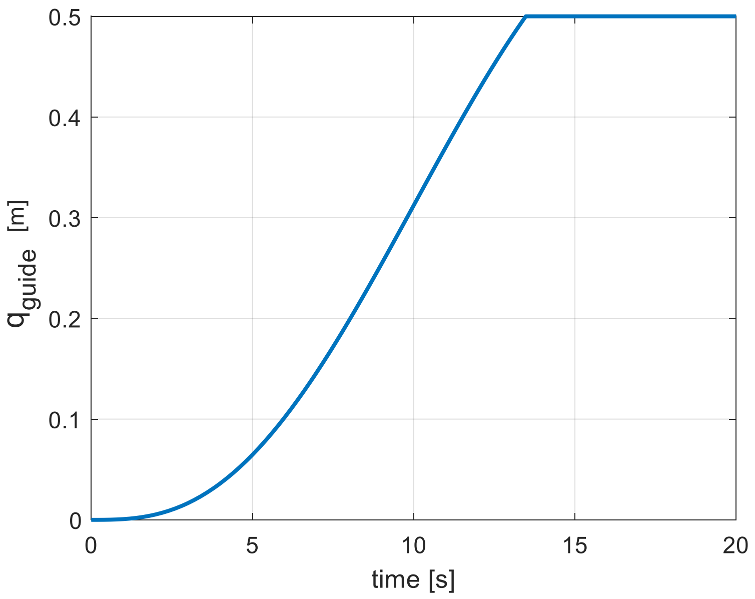

Figure 3.2.03 - Pseudo-inverse base position

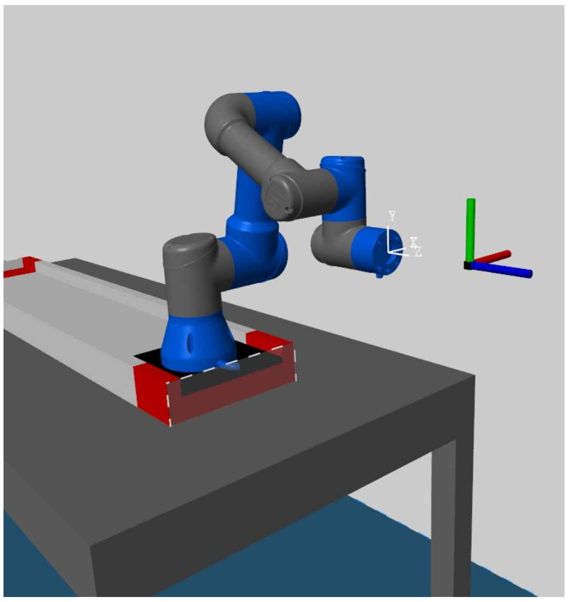

Figure 3.2.04 - Pseudo-inverse final robot configuration

This simple method is not enough for this trajectory. Even the controller activation does not help, because it would only add an extra velocity for the guide, but the saturation of the mechanical joint limit would not still be perceived. This leads to the implementation of a null space solution, and the saturation can be avoided thanks to self-motions that generate the elbow elongation.

Since in this scenario there is only one limit of interest, only the base gain is active to obtain simpler and clearer results.

The null space projection affects the other joints, in particular the shoulder, the elbow and the first joint of the wrist (Figure 3.2.07). In this way, while the guide slows down (Figure 3.2.06), the EE keeps progressing until the final pose is reached, as Figure 3.2.08 shows. The desired final pose is within the robot workspace; thus, the trajectory is feasible and no task error is generated.

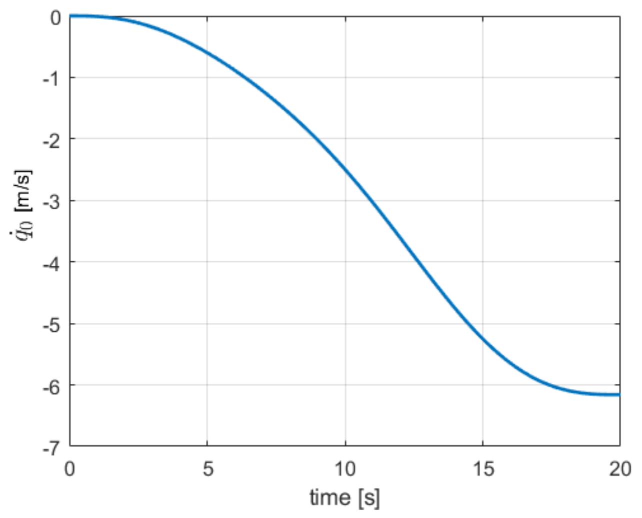

Figure 3.2.05 - Guide velocity derived from its gradient

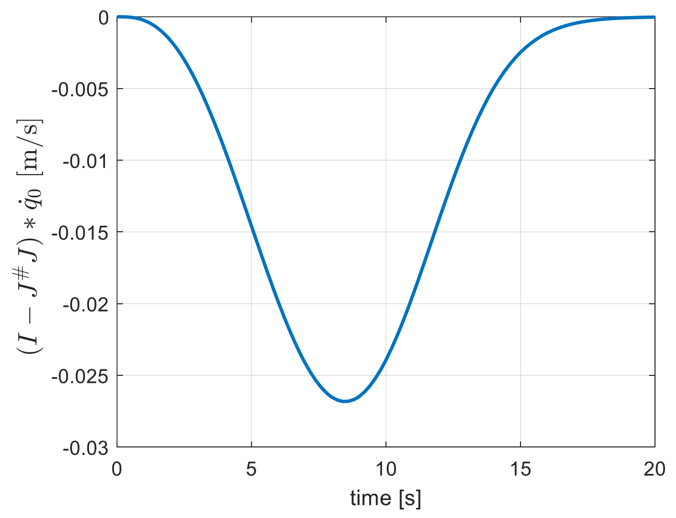

Figure 3.2.06 - Null space base velocity

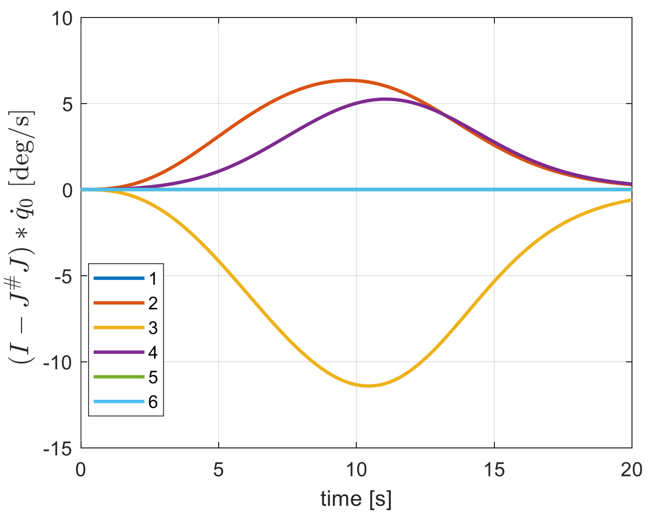

Figure 3.2.07 - Null space joint velocities

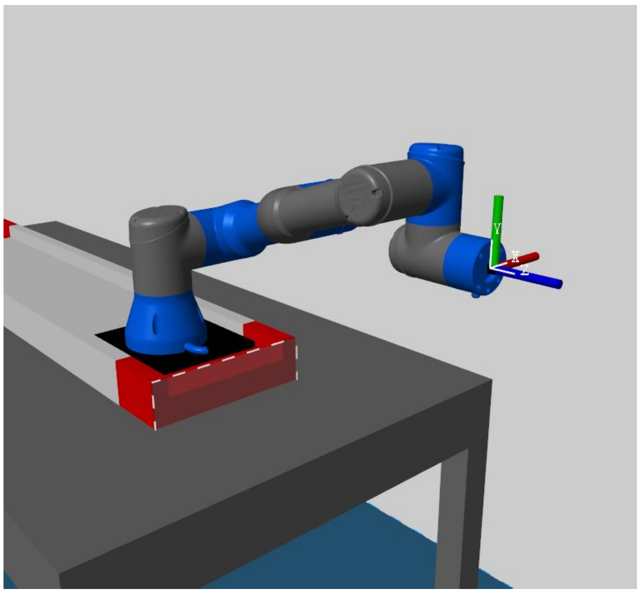

Figure 3.2.08 - Final robot configuration. Pseudo-inverse + null space

Another possible solution for avoiding saturation is to employ the weighted pseudo-inverse. By increasing the guide weight only up to 50, for example, its velocity gets lower, and the goal is obtained (Figure 3.2.09). Nevertheless, this does not ensure that the limit is not reached, because it depends on the arbitrary value of the weight. Indeed, if the weight is only 10, the limit is reached (Figure 3.2.10) and a position error occurs.

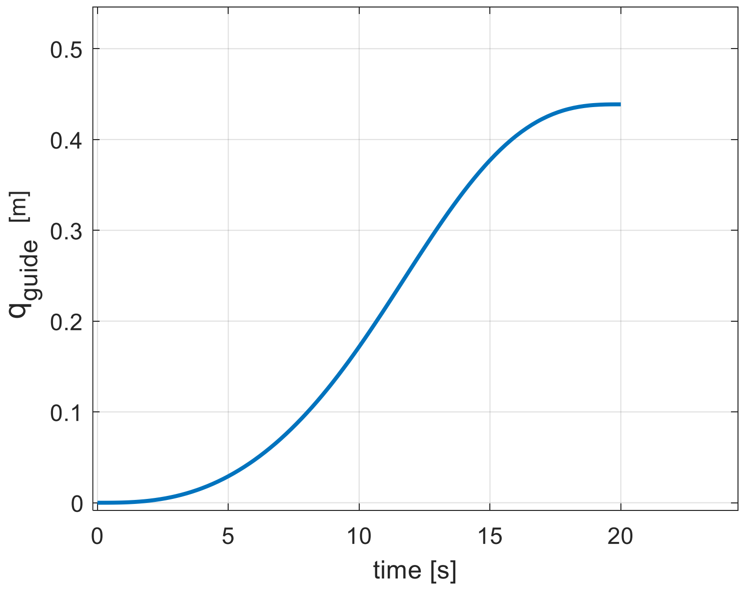

Figure 3.2.09 - Base position, guide weight = 50

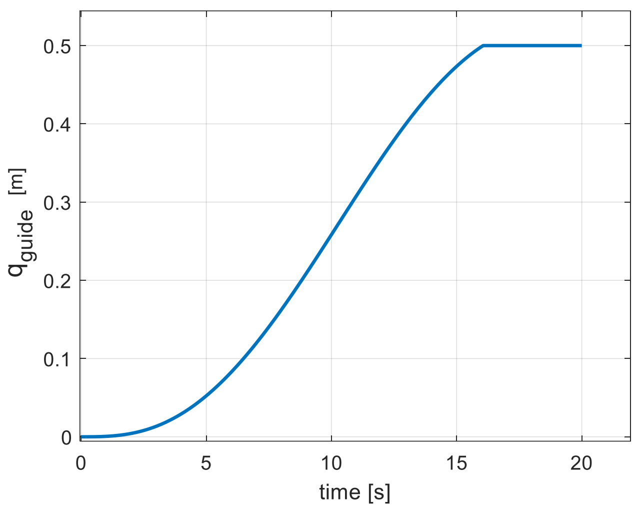

Figure 3.2.10 - Base position, guide weight = 10

For this reason, the null space contribution is still needed. Thanks to its “repulsive” effect, a weight equal to 10 can be adopted. However, in this case, the robot immediately extends the elbow (Figure 3.2.11), then it “drags” its base as soon as it recognizes that it has to move it anyway to reach the final pose.

Analytically, at the time instant of 10.8 s, which is approximately half of the trajectory duration, the elbow is close to its full extension ($q_3 = 0$, in Figure 3.2.12), but the base has moved only 16 cm (Figure 3.2.13). In the other half, while the other joints are constants, the base moves until the final pose is attained, covering a total distance of 27 cm. Since for the pseudo-inverse the elbow stretching lasts longer, it is more natural, therefore, that overall the robot movement appears more harmonious.

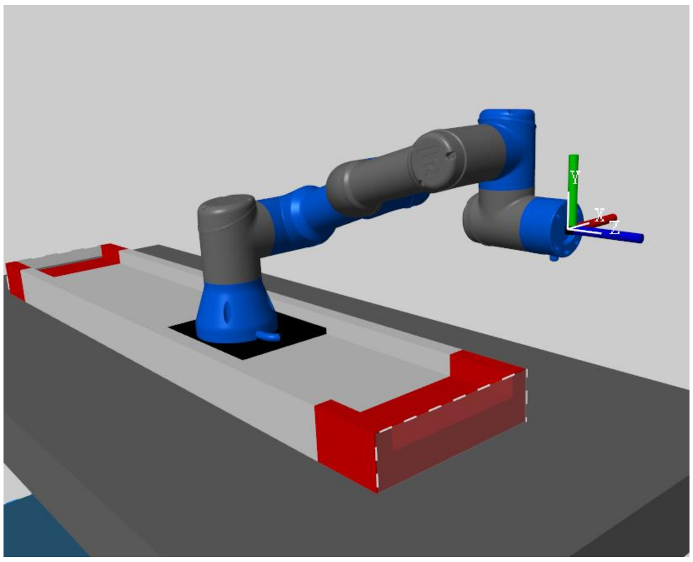

Figure 3.2.11 - Elbow stretching. t = 10,8 s

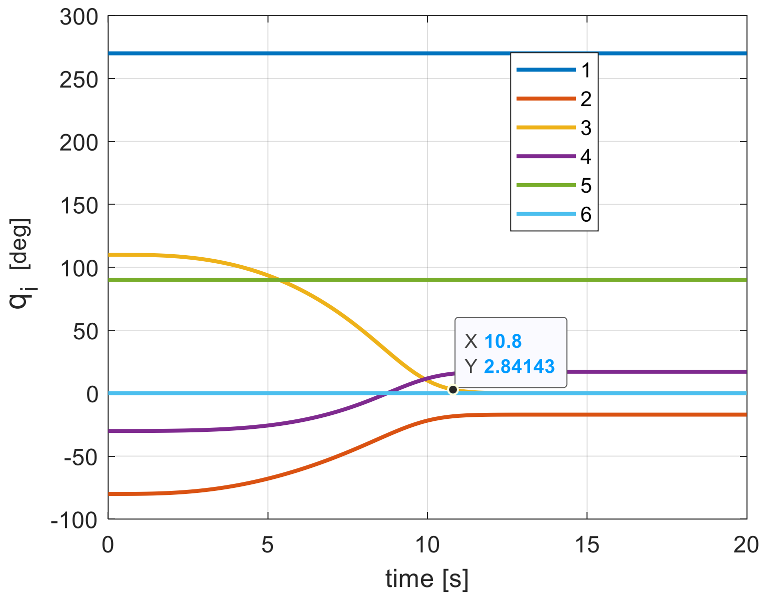

Figure 3.2.12 - Joint configurations, guide weight = 10+ null space

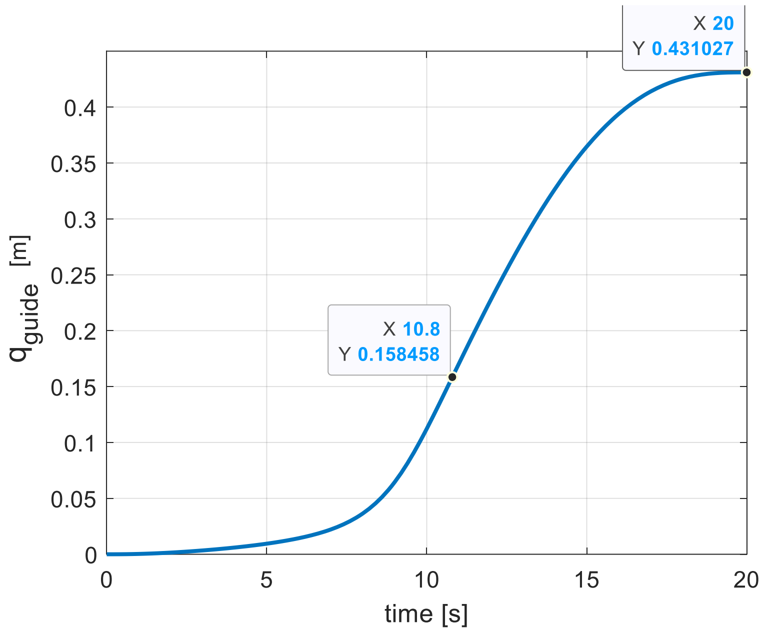

Figure 3.2.13 - Base position, guide weight = 10+ null space

3.2.1 Disturbance

To view the controller in action, and its effect on a closed-loop solution, a disturbance on the joint velocities is inserted. A pseudo-inverse solution is considered with the essential null space contribution for the guide. The control gains are equal to $\boldsymbol{K}_{P,i} = \boldsymbol{K}_{O,i} = \frac{1}{3}$. The reason for this choice will be explained in the next pages. In this occasion, only a disturbance affecting the shoulder of the robot is examined, and it is analytically defined as a time-series signal on Matlab, as displayed in Figure 3.2.14.

The signal is a step of 3 deg/s starting from the instant t = 5 s and finishing at t = 7 s. In this way, discontinuities of joint velocities are generated (Figure 3.2.15), leading to an increase of joint accelerations, which are anyway low (Figure 3.2.16). Accordingly, the kinematic control should be effectively working.

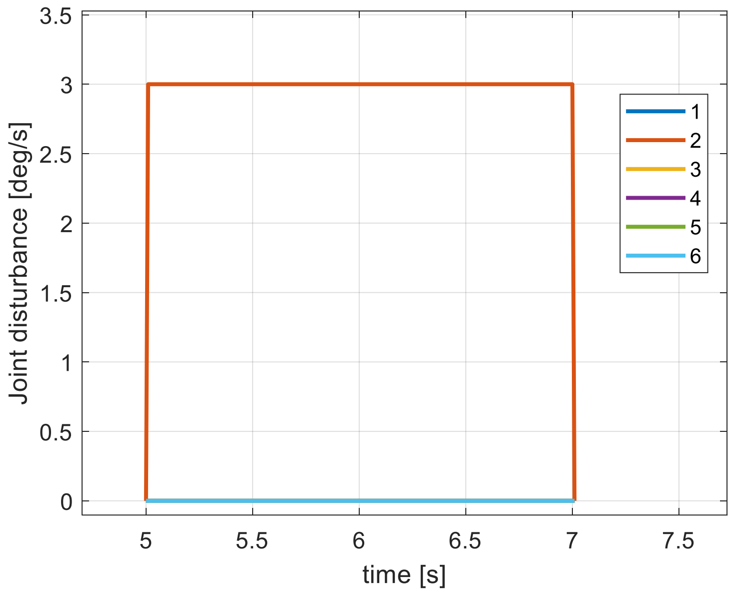

Figure 3.2.14 - Joint disturbance

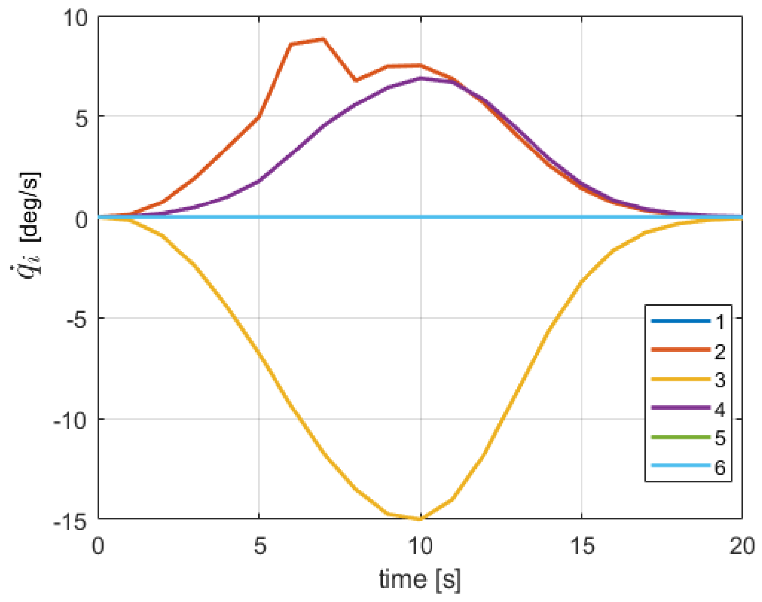

Figure 3.2.15 - Computed joint velocities due to the disturbance

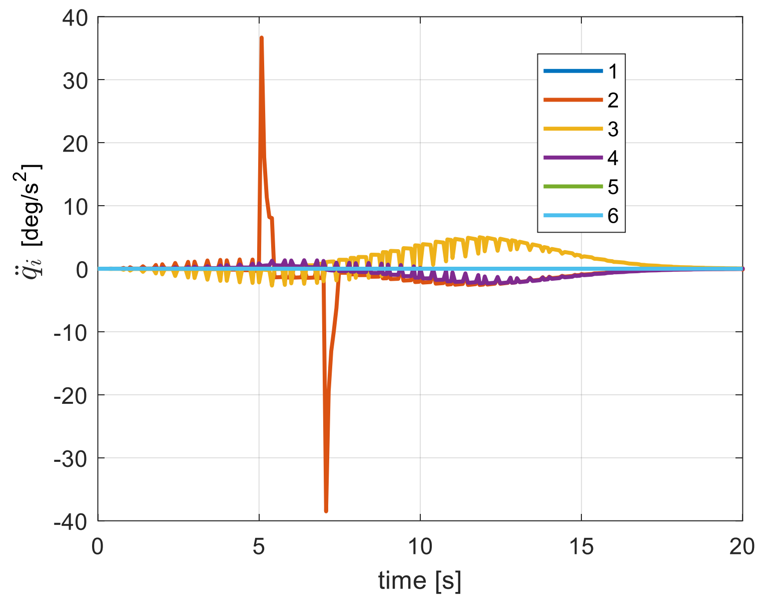

Figure 3.2.16 - Joint accelerations due to the disturbance

Once the signal appears, it modifies the configuration of the second joint (Figure 3.2.16) lowering the robot (Figure 3.2.17). In an operational space view, it creates a negative linear velocity along the *z*-axis, an undesired variation on the *y*-velocity (Figure 3.2.18), and a negative angular velocity along the *x*-axis (Figure 3.2.19). Hence, a pose error occurs (Figure 3.2.20).

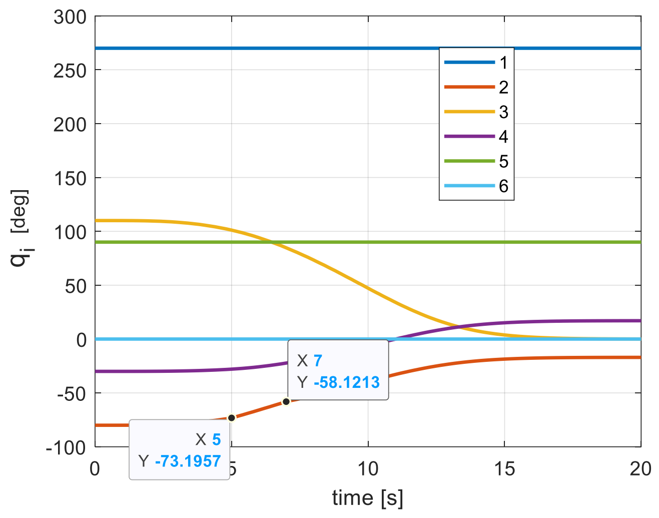

Figure 3.2.16 - Computed joint configurations due to the disturbance

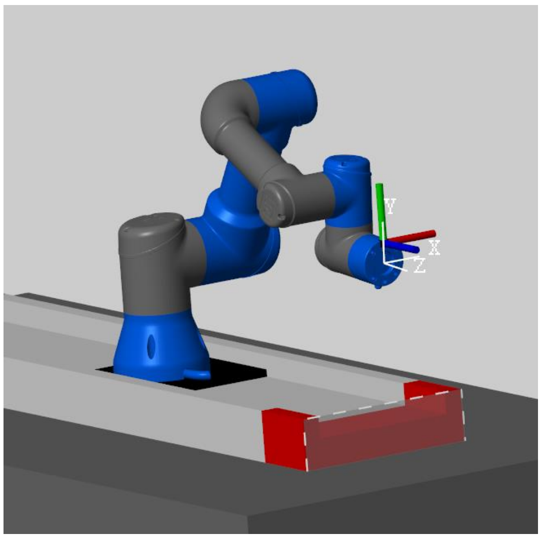

Figure 3.2.17 - Lowered robot configuration. t = 6,94 s

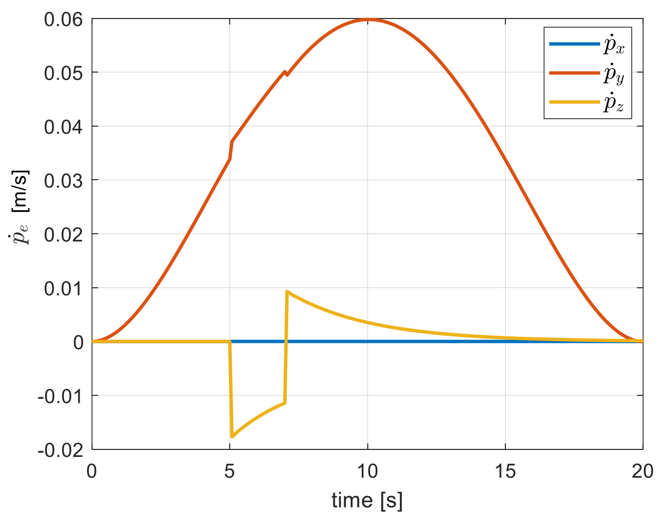

Figure 3.2.18 - Obtained linear velocity of the EE due to the disturbance

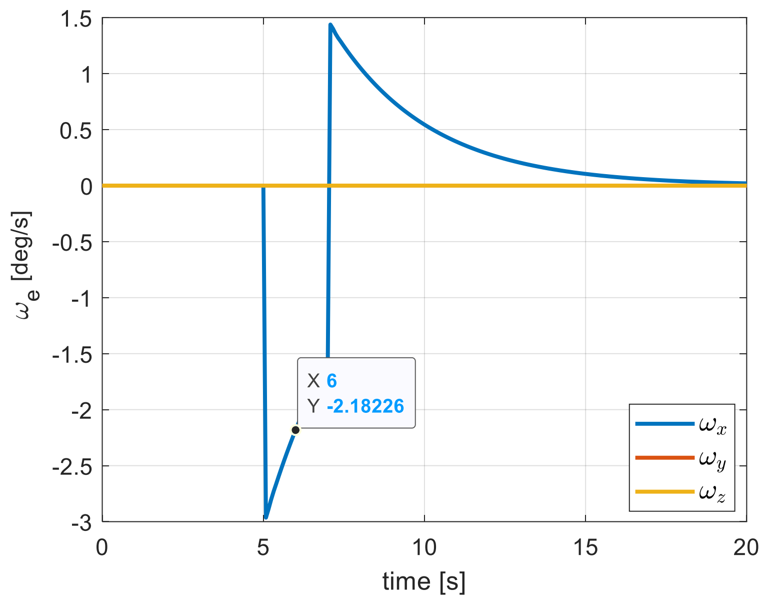

Figure 3.2.19 - Obtained angular velocity of the EE due to the disturbance

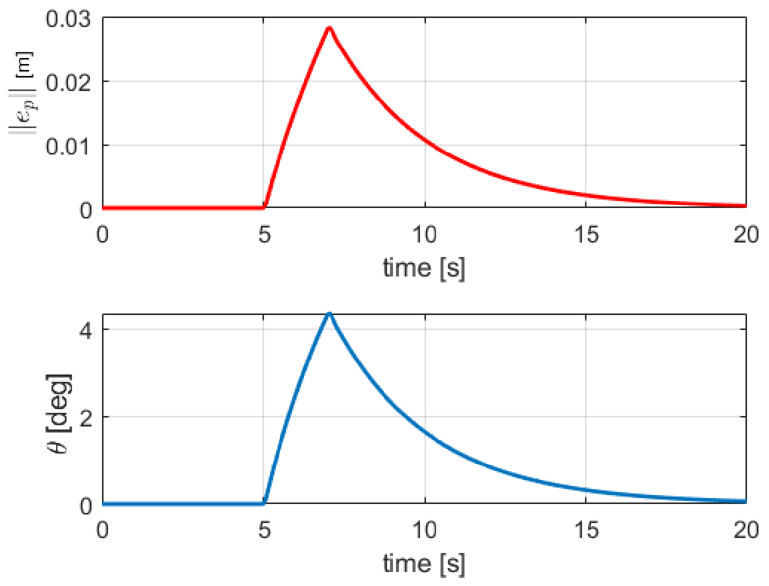

Figure 3.2.20 - Pose error due to the disturbance

Since the controller can only lower the slope of the error, the error increases as long as the disturbance is present; nevertheless, after the signal disappearance, it can effectively adjust the trajectory. It is worth highlighting the exponential trend of the error, obtained resolving the linear ODE of (2.19). Due to its action, the velocity command (Figures 3.2.21 and 3.2.22) changes from the original EE linear velocity (Figures 3.2.00) and the null angular velocity, due to the presence of the controller.

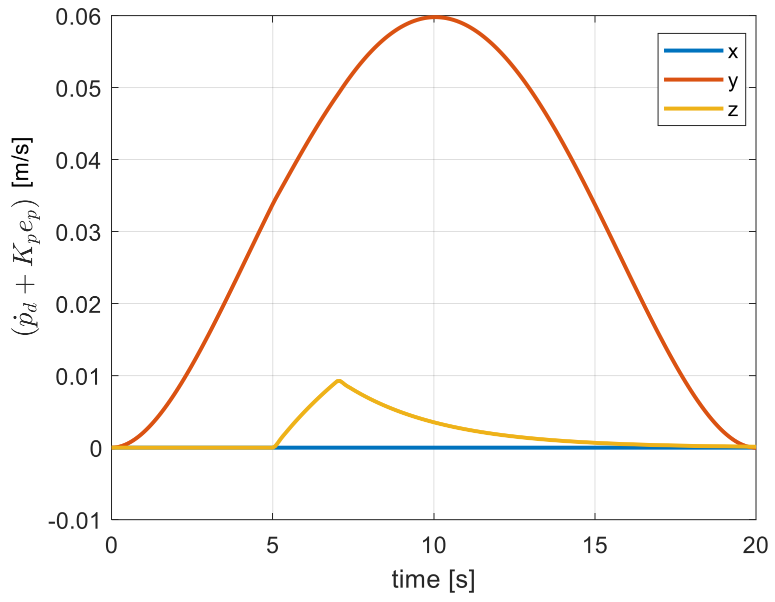

Figure 3.1.21 - Linear velocity command of the EE due to the disturbance

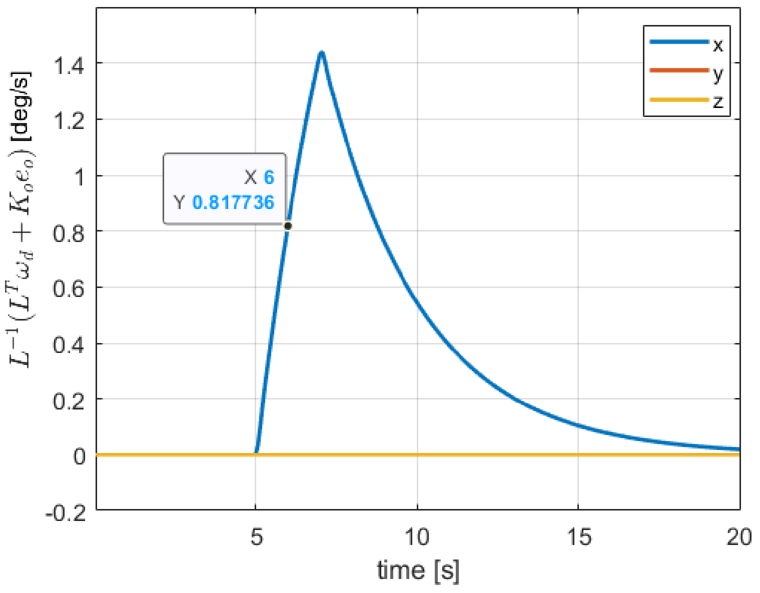

Figure 3.2.22 - Angular velocity command of the EE due to the disturbance

The EE velocity command is transformed into a joint velocity command through the inverse kinematics, obtaining Figure 3.2.23. Obviously, they differ from the computed ones (Figure 3.2.15), because the obtained joint velocities are affected by the signal. However, once the disturbance ceases to exist, they become the same. This is also valid for the EE velocity; indeed, the task error (Figure 3.2.24) instantaneously returns to zero once the signal disappears.

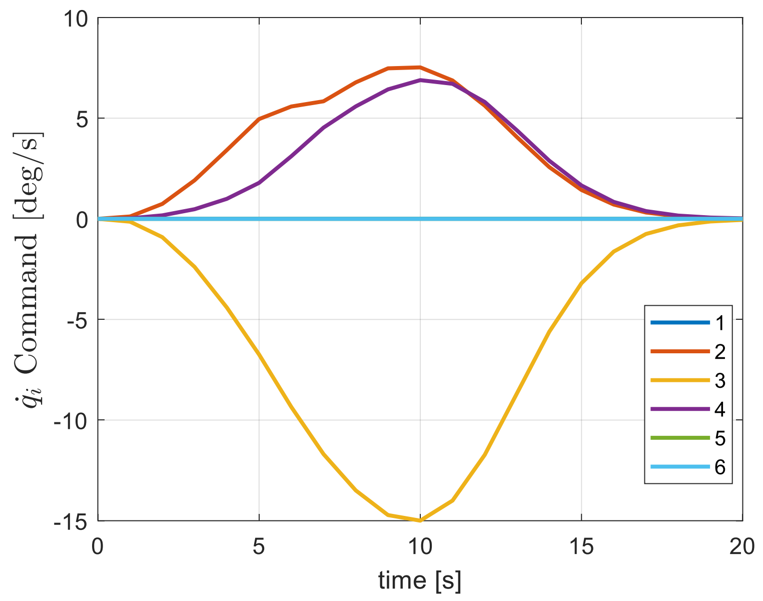

Figure 3.2.23 - Joint velocities command due to the disturbance

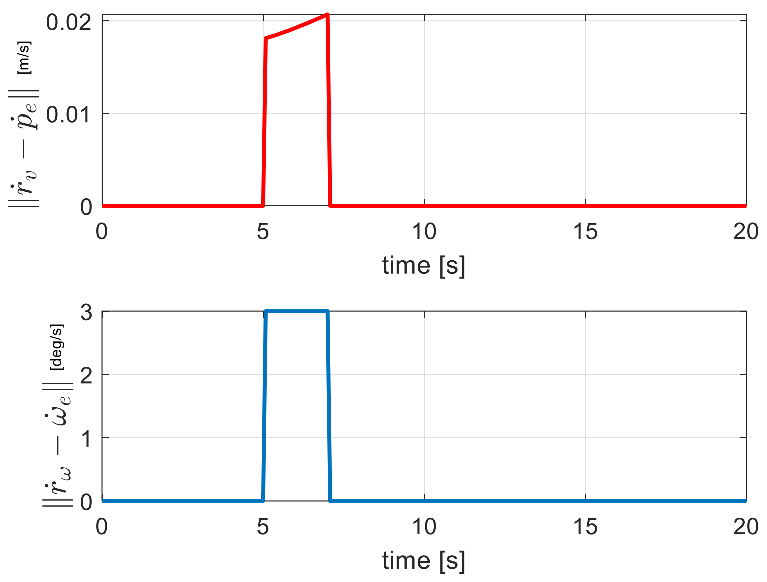

Figure 3.2.24 - Task error due to the disturbance

It is worth specifying that the task error compares the obtained EE velocity with the EE reference velocity (Figures 3.2.21 and 3.2.22), not the ideal one (Figure 3.2.00). In this sense, the angular velocity task error is constant and equal in value to the disturbance. The reason for this constancy is that the command gives an angular velocity with a value that has an opposite sign with respect to the disturbance to compensate it. Hence, the obtained EE angular velocity decreases with the same value. Consequently, the difference does not vary, as highlighted in Figures 3.2.19 and 3.2.22: at t = 6 s the angular velocity is equal to

0.82 deg/s for the reference, the actual velocity is equal to (-)2.18 deg/s, and their difference is equal to 3 deg/s.

Subsequently, at t = 7s, the EE angular velocity changes instantaneously its sign (Figure 3.2.19), as it happens for the linear *z* velocity (Figure 3.1.18), meaning that now the robot is going back to its original path by raising its posture to achieve its true height.

However, the linear task error is not exactly constant, but it varies by a small measure with a mean value of 0.019 m/s, that cannot be easily deduced from something that is known. In fact, the disturbance influences the shoulder, which is a revolute joint, so considerations for the angular velocity are immediate, while for the linear velocity are not simple.

Finally, once the signal disappears, it seems that under a velocity perspective, there are no issues anymore; however, this can be misleading. In fact, the adjustment of the robot pose actually takes place after that time instant, as said before and shown in Figure 3.2.20.

3.2.2 Initial error

In the previous section where the robot starts in its correct pose, a *trajectory following* kind of control was tested under the action of a disturbance.

Nevertheless, another way to test the controller is by starting with an incorrect initial robot configuration (Figure 3.2.25), that causes a pose error from the beginning of the trajectory. This can be done by autonomously selecting a wrong configuration, which differs from all the possible eight closed-form solutions, solved from the initial analytical inverse kinematics, which is in fact ignored.

Now two possible kinds of control are possible: the *trajectory tracking*, in which the reference is time-varying, and the *regulation of pose*, in which the reference is constant.

Starting with the former, for the first approach, only a pseudo-inverse solution is implemented.

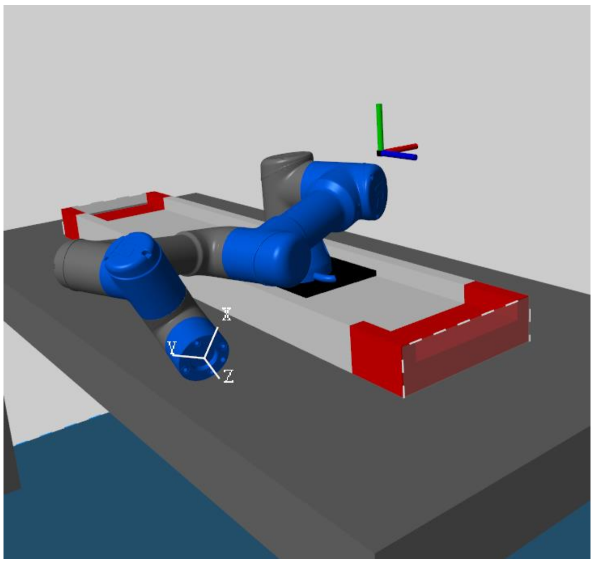

Figure 3.2.25 - Incorrect initial robot configuration

With a value of $\boldsymbol{K}_{P,i} = \boldsymbol{K}_{O,i} = \frac{1}{3}$ the error is compensated fast enough to ensure convergence by the end of the trajectory (Figure 3.2.26). This is also possible because the resulting error is great (0.59 m for the position), so a great recovering velocity is generated. As for the disturbance instance, this quantity was proven to be the best for getting a fast adjustment without obtaining unachievable joint accelerations, particularly for this case. Thus, this value will be used as reference for these kinds of control.

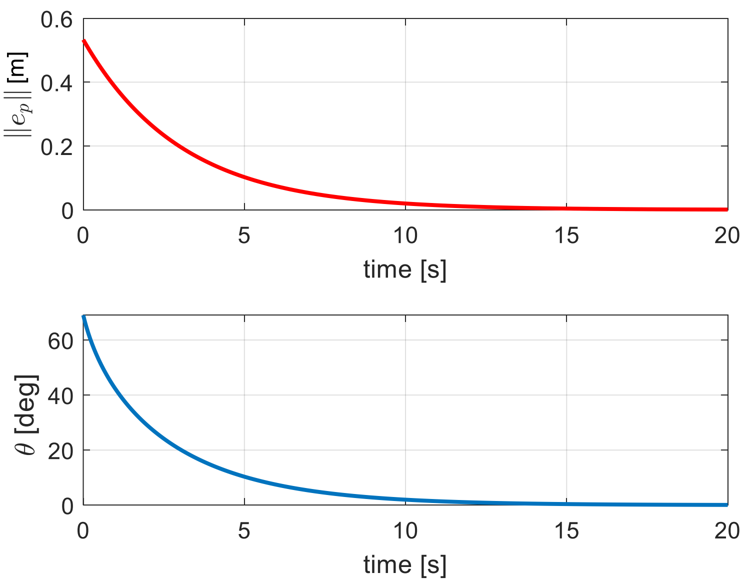

Figure 3.2.26 - Pose error due to the incorrect initial configuration

However, to quickly achieve the convergence, still great joint and base accelerations are generated at the beginning (Figures 3.2.27 and 3.2.28). Differently from the previous case, the dynamic effects are considerable, and a kinematic control could not be enough.

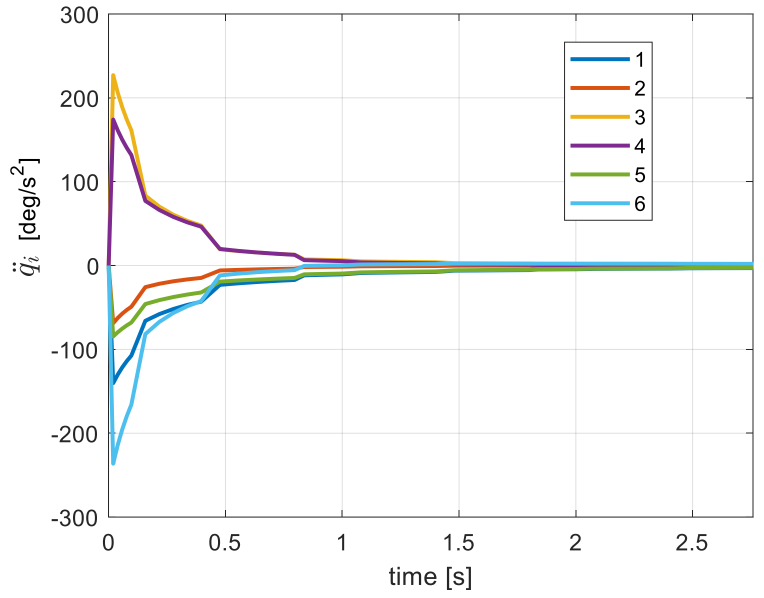

Figure 3.2.27 - Joint accelerations originated from the controller action. First 3 seconds

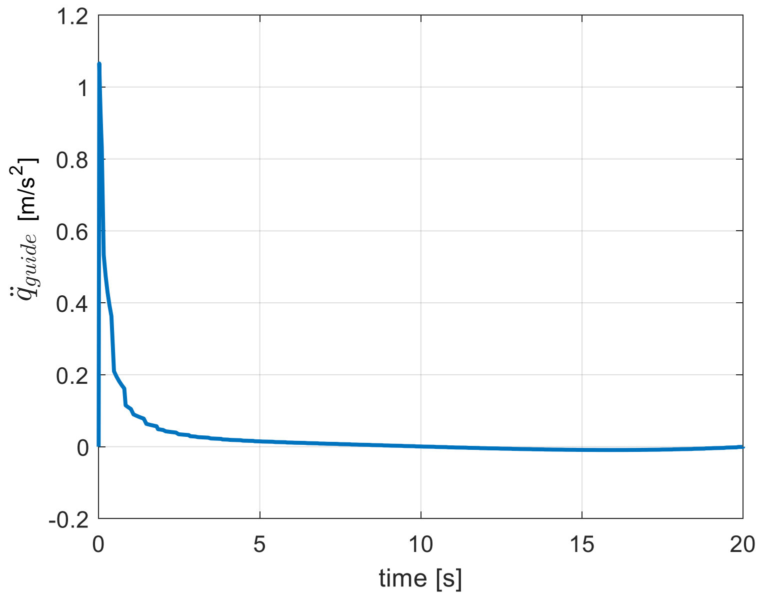

Figure 3.2.28 - Base acceleration originated from the controller action

Even if there is no null space contribution, the limit of the linear guide is not reached (Figure 3.2.29), because the robot rearrangement is accomplished through a backward movement of the base and an immediate elbow stretch (Figure 3.2.30). The robot arm is kept low throughout the rest of the duration of the trajectory allowing the avoidance of the limit (Figure 3.2.31).

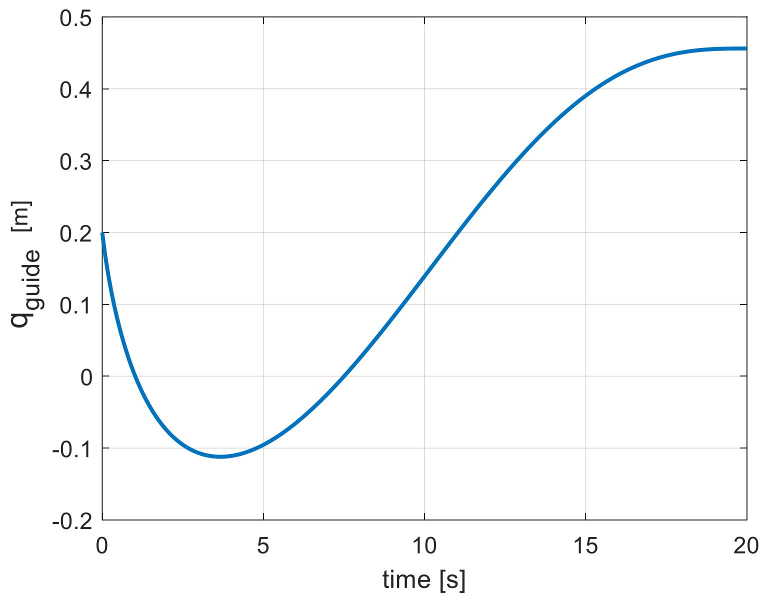

Figure 3.2.29 - Base position, pseudo-inverse solution for the incorrect initial configuration

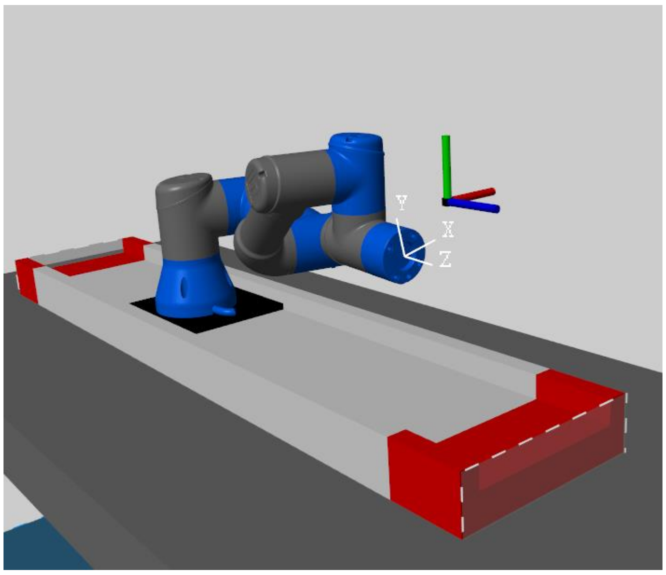

Figure 3.2.30 - Robot configuration. t = 4,2 s. (Pseudo-inverse)

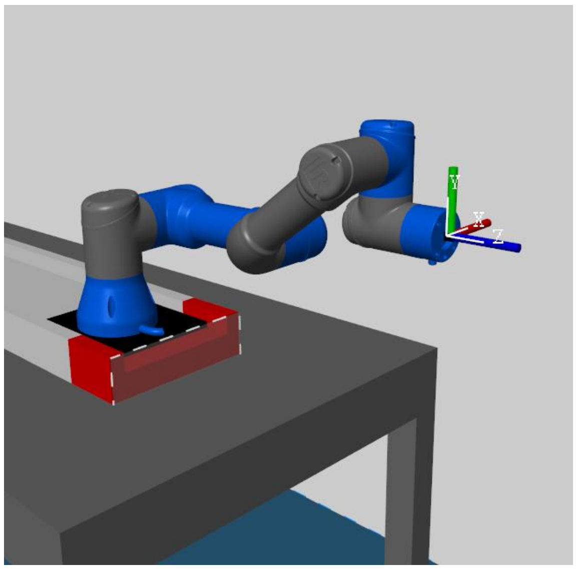

Figure 3.2.31 - Final robot configuration. (Pseudo-inverse)

Even if the actual torques are unknown, the post-processing dynamics can be analyzed every time, and this is the occasion to do it for the first time.

For the static balance (Figure 3.2.32), as expected, the shoulder contributes the most since it holds almost all the robot weight in the elongated robot posture.

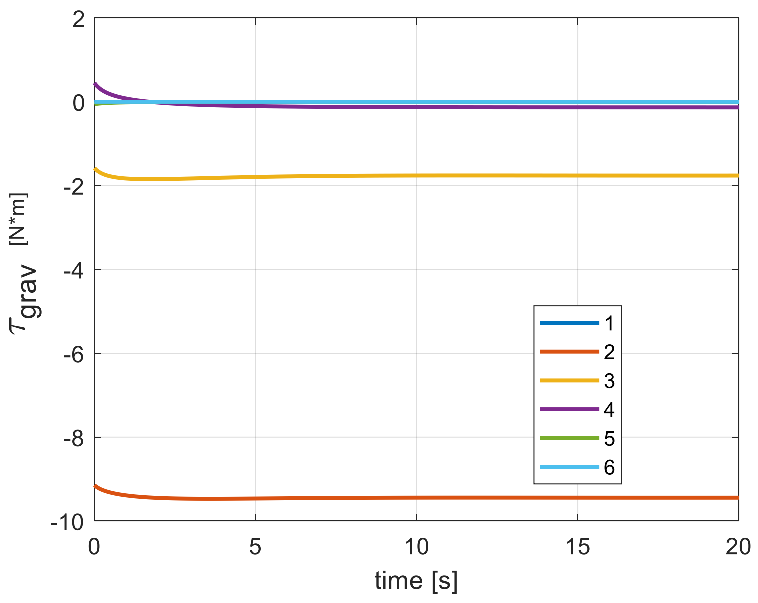

Figure 3.2.32 - Gravity torques

For the dynamic balance (Figure 3.2.33), only the effect of the joint velocities and acceleration are considered.

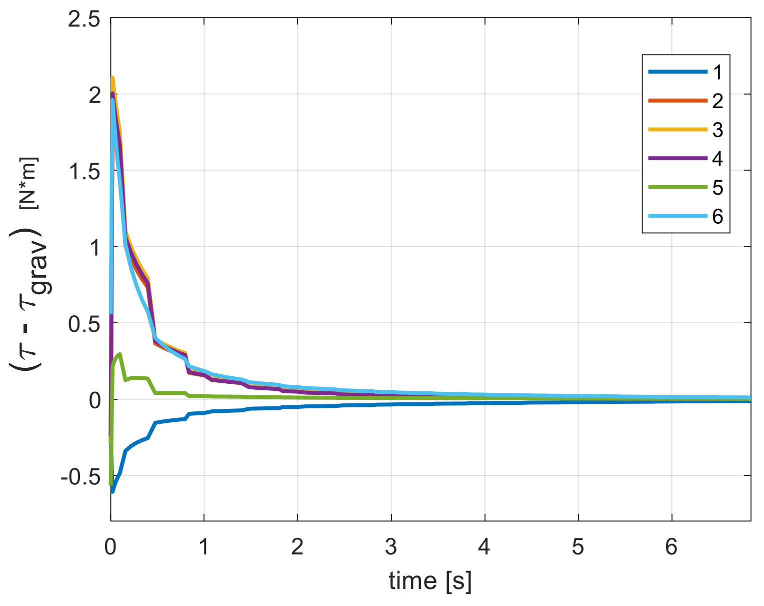

Figure 3.2.33 - Total torques needed minus the gravity torques, first 7 seconds

Although the joint accelerations are great, the moments of inertia of the robot are small, so the torques needed to produce them are eventually low. This cannot be said for the guide, since it holds all the robot weight of 15 kg, and for achieving an acceleration of 1 m/s (Figure 3.2.28) a force of 14 N is requested (Figure 3.2.34). The force is not 15 N as expected, presumably because of internal robot movements in the direction opposite to the base trajectory.

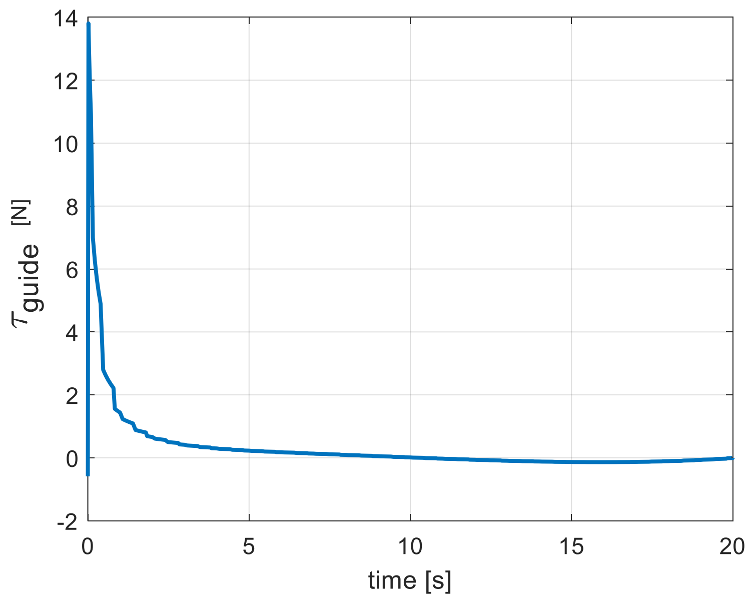

Figure 3.2.34 - Prismatic torque

In this case, it is the robot arm that is close to its mechanical limit of 15 degrees (Figure 3.2.35). For this reason, to avoid potential collisions between the shoulder and the guide (even if they did not happen in the precedent motion), in the following solution the projected gradient method is active for q_2, but also so for q_3 and q_4 (3.02). In this way, the elbow gets up, which is also preferable for a safer perception of the robot from a human point of view.

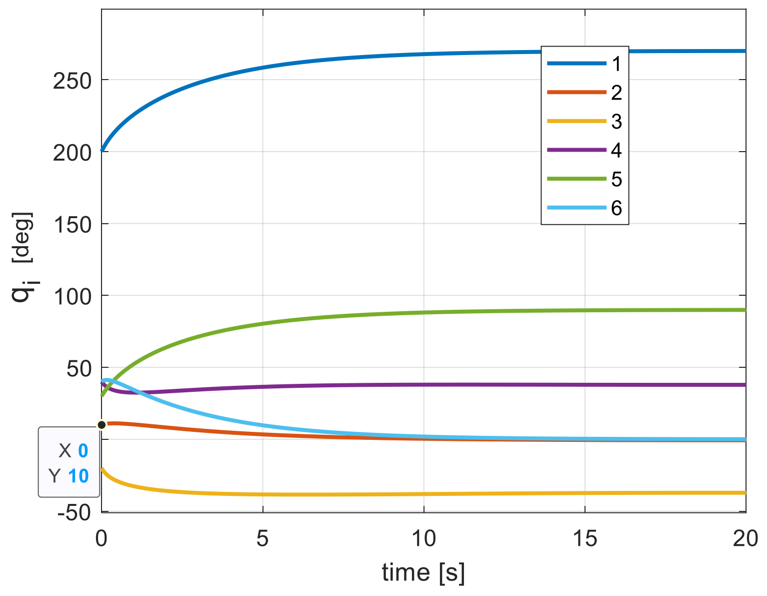

Figure 3.2.35 - Joint configurations (Pseudo-inverse)

The obtained gradient velocities are shown in Figures 3.2.36 and 3.2.37

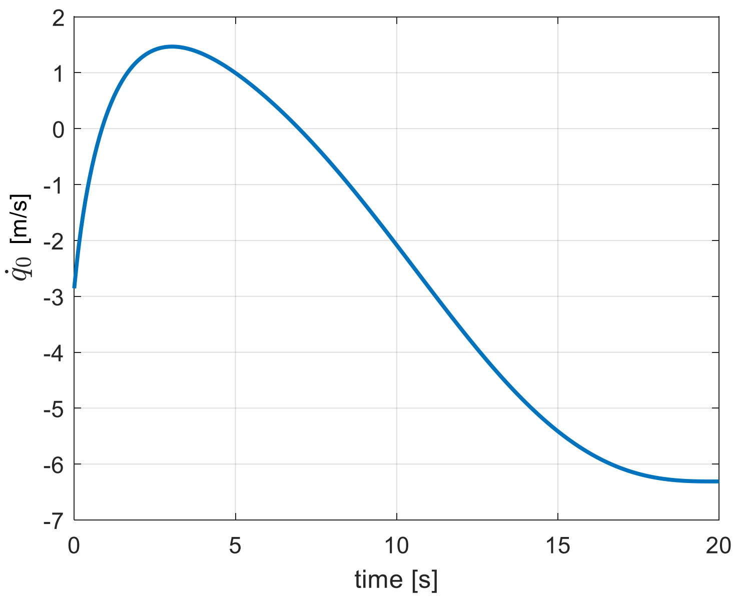

Figure 3.2.36 - Guide velocity derived from its gradient

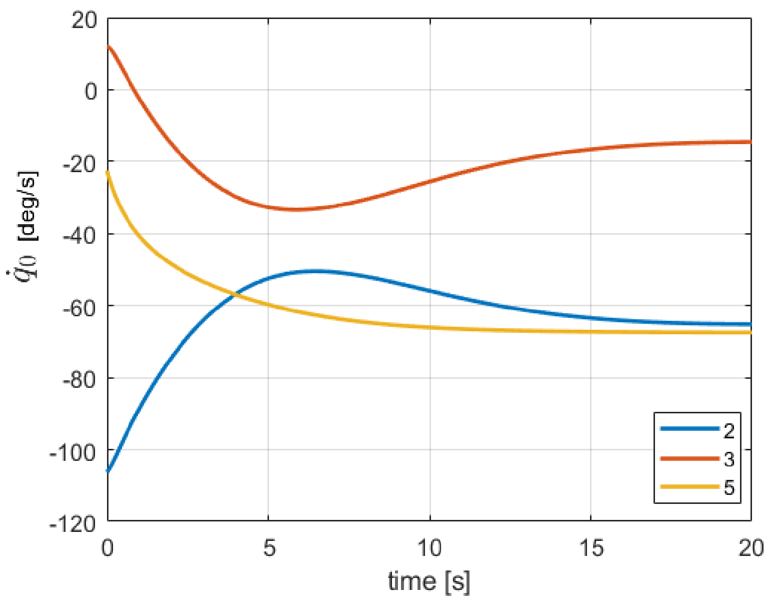

Figure 3.2.37 - Joint velocities derived from their gradient (q_2, q_3 and q_5)

As expected, the second joint starts with a great value, due to its closeness to the limit.

However, when its gradient velocity is projected in the null space, its effect is reduced (Figure 3.2.38). The null space projection has an influence also on q_4 and q_6, even if they are not included in the projected gradient method.

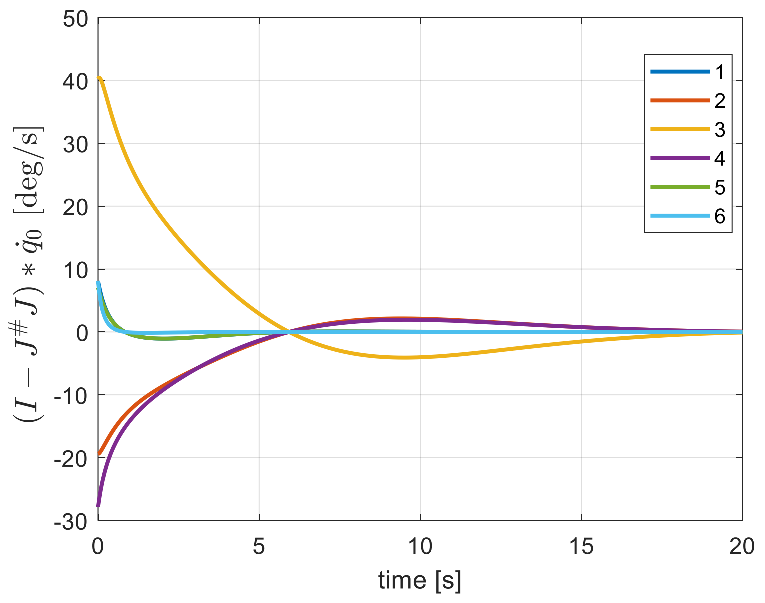

Figure 3.2.38 - Null space joint velocities

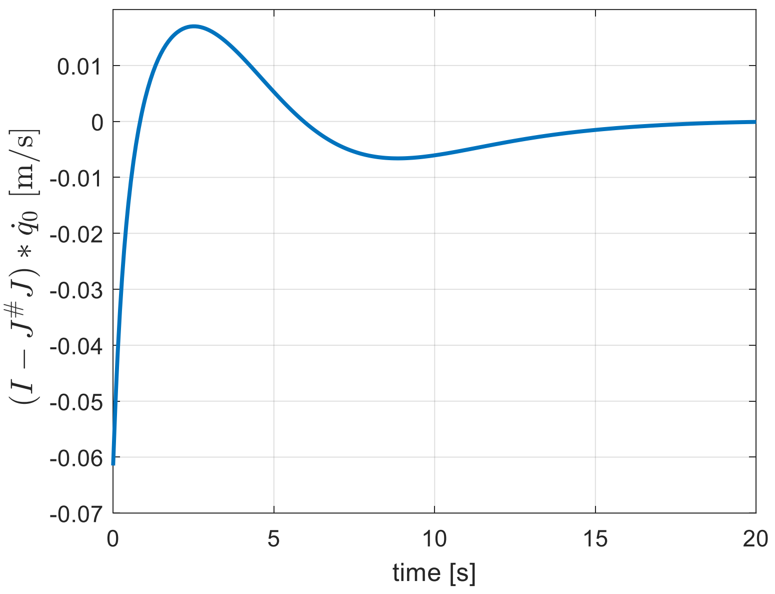

Figure 3.2.39 - Null space base velocity

From Figure 3.2.39 it is possible to see for a second time how the base velocity deriving from its gradient gets drastically reduced once projected.

As wanted, in the first seconds of the trajectory, the robot rises itself (Figure 3.2.40), and it keeps its elbow up until the end of the path (Figure 3.2.41).

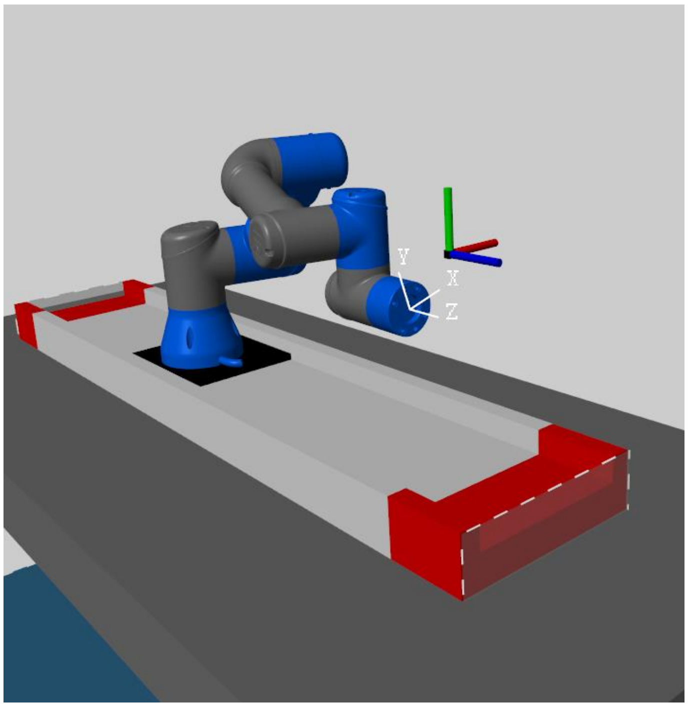

Figure 3.2.40 - Robot configuration. t = 4,2 s. (Pseudo-inverse +null space)

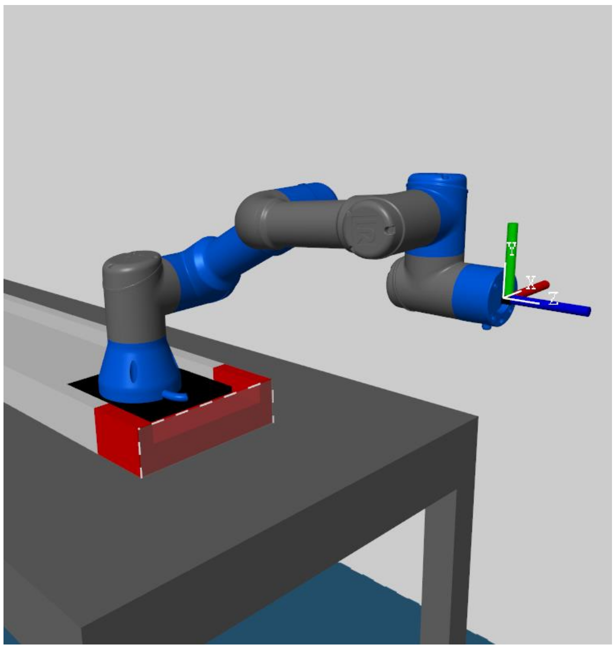

Figure 3.2.41 - Final robot configuration. (Pseudo-inverse + null space)

Finally, since the trajectory tracking control is implemented due to reference motion, the TCP returns to its desired path in a bent way (Figure 3.2.42), despite using a rectilinear p-t-p motion planning.

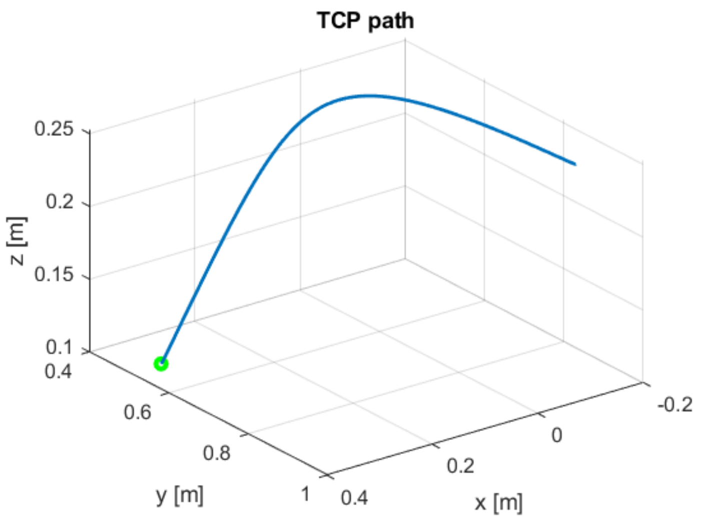

Figure 3.2.42 - TCP path. Pseudo-inverse + null space solution

By adopting the regulation pose control, instead, only the final pose is requested, which remains still. Starting from all the same parameters of the previous motion, now a straight path is achieved (Figure 3.2.43).

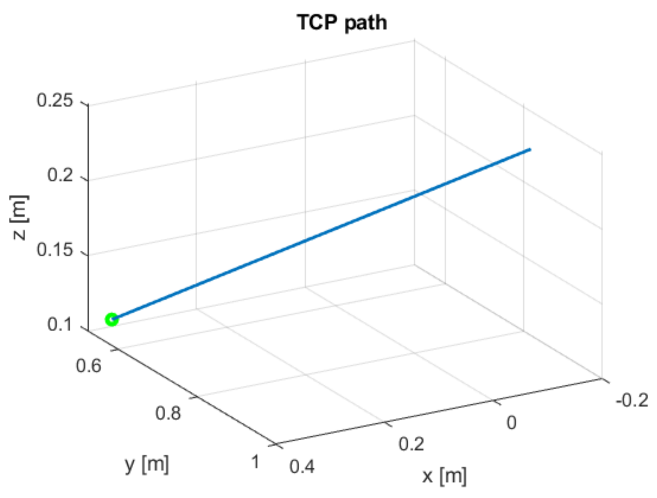

Figure 3.2.43 - TCP path. Pseudo-inverse + null space solution. Regulation pose control

Now that the desired velocity is null, its *feedforward* action is absent, so the velocity command is only generated from the *feedback* action. Indeed, its trend (Figure 3.2.44) is similar to the error one (Figure 3.2.45). For the position error/linear velocity the only difference is a scaling factor given by the gain $\boldsymbol{K}_P$. For the orientation error/angular velocity instead, remembering that $\dot{\boldsymbol{\omega}}_d$ is null, the norm of the command becomes:

$$\|\boldsymbol{L}^{-1}(\boldsymbol{K}_O\boldsymbol{e}_O)\| = \|\boldsymbol{L}^{-1}(\boldsymbol{K}_O\boldsymbol{r}\sin\theta)\| \tag{3.03}$$

Indeed, the orientation error $\boldsymbol{e}_O$ is a vector, as defined in (2.23), and not just equal to its magnitude θ. Furthermore, it is the sine of θ that is considered, which subsequentially is pre-multiplied not only by the gain $\boldsymbol{K}_O$, but also by the matrix $\boldsymbol{L}^{-1}$. Consequenly, the ratio between (3.03) and θ, is not clear. Moreover, the matrix is time-varying, and the sine function is not linear, so the ratio is not constant during the trajectory.

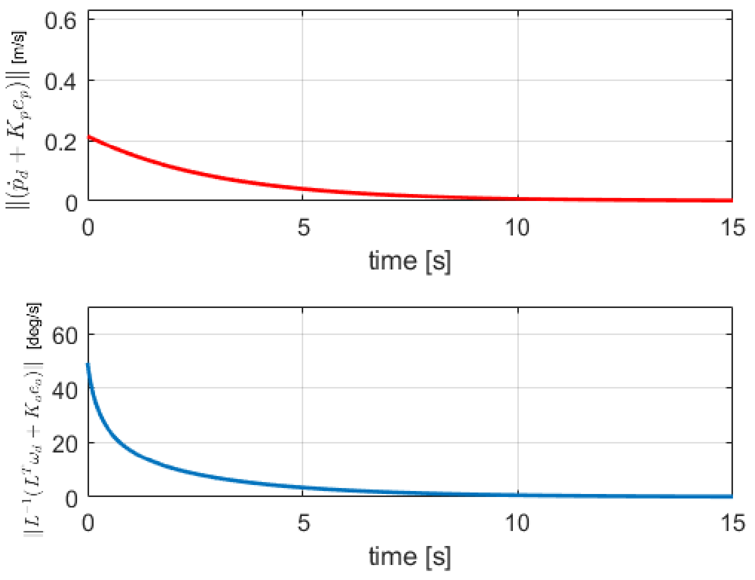

Figure 3.2.44 - Velocity command. Pseudo-inverse + null space solution. Regulation pose

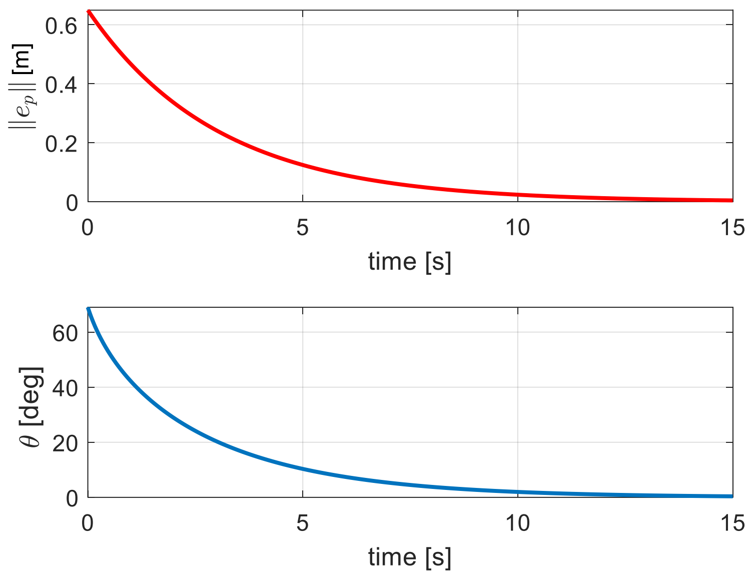

Figure 3.2.45 - Pose error. Pseudo-inverse + null space solution. Regulation pose control.

As the error gets lower and lower, the velocity diminishes proportionally to it, so the convergence is slowly assured. Again, higher gains would cause too great accelerations, even more in this kind of motion, in which the starting error is greater than before, due to the bigger distance of the desired reference from the initial pose.

Indeed, the joint accelerations (Figure 3.2.46) are already higher than in the tracking trajectory control (Figure 3.2.27). As a result, at the beginning a fast motion is achieved and the robot quickly arrives close to the final pose (Figure 3.2.47).

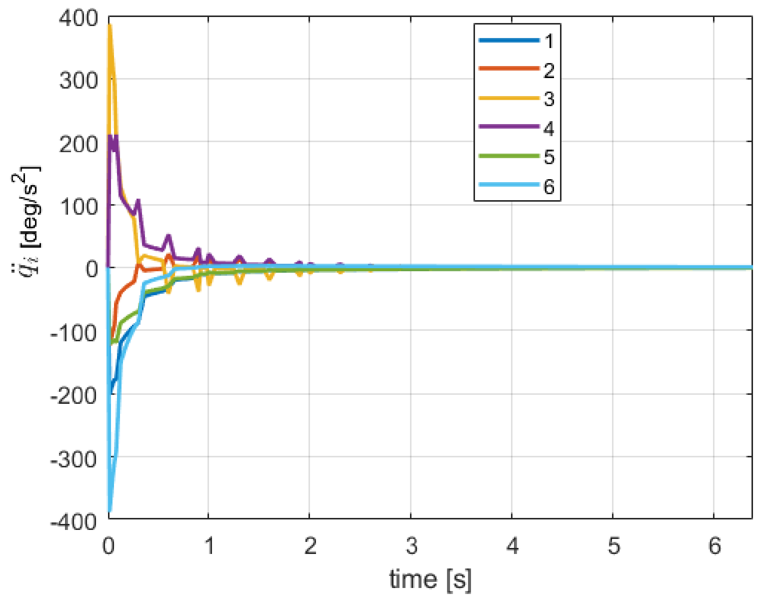

Figure 3.2.46 - Joint accelerations. Regulation pose control. First 6,5 seconds

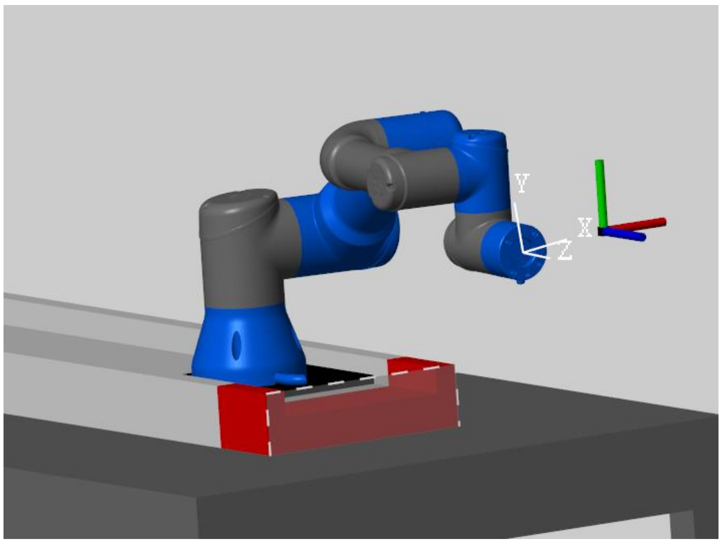

Figure 3.2.47 - Robot configuration. Regulation pose control. t = 5,9 s

3.3 Path motion trajectory

Once that the basics of the methods are known, a path motion trajectory is presented to simulate a Human-Robot interaction: a tool/object passing between two operators through the robot EE. Even if a point-to-point trajectory could be possible, it is better to subdivide the linear path into a segmented one, so the robot can have more time to achieve an enclosed posture and more collaborative motions are obtained. As illustrated in the section "2.2.1 Trajectory planning", the path motion is actually a repeated point-to-point that requires a fixed-time step to work. The time step was chosen equal to 8 ms for every simulation.

The robot starts facing one side of the table (first waypoint) as if it is picking an object from an operator, then crossing the internal waypoints, it moves to the other side of the table (fourth waypoint), where it stops for a while (fifth waypoint), leaving the time for another operator to collect the tool. Subsequently, it goes back to its original position to be ready for another task while going through the same waypoints of the way out. Hence, there are eight waypoints in total that define seven segments. The trajectory to define each segment has the same duration of 4 seconds, so the whole trajectory lasts 28 seconds. The time scaling is made from a fifth polynomial with null boundary conditions for each segment. In this way, each waypoint is reached with a null EE velocity. Accordingly, it is simpler to visually identify when a segment starts and when it finishes.

The waypoints are plotted in Figure 3.3.00. They have a constant altitude of 0.2 m with respect to the ground reference, which is placed on the table; the black line represents the guide axis on the robot base level, which is 50 mm above the ground. Their *x* and *y* coordinates are the following:

Waypoint	**X**	**Y**
First and last	-0.4 m	-0.3 m
Second and seventh	-0.2 m	-0.2 m
Third and sixth	0.2 m	0.2 m
Fourth and fifth	0.4 m	0.3 m

Table - 3.3.0, Waypoints X and Y coordinates

From Table 3.3.0, it is evident a desired symmetry with respect to both the X and Y axes.

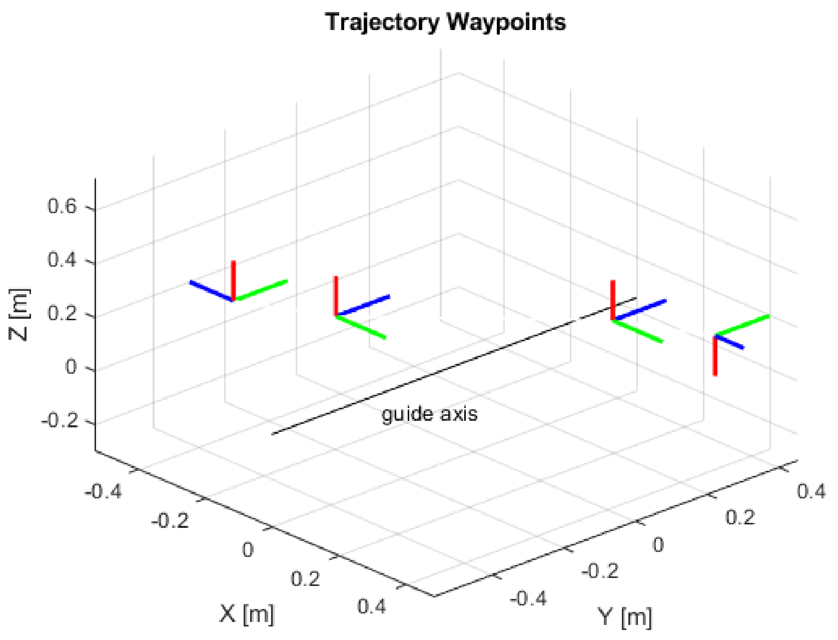

Figure 3.3.00 - Trajectory waypoints

From Figure 3.3.00, it can be noticed that the z-axis of the EE of the external waypoints is perpendicular to the guide axis, meaning that the EE is correctly facing toward the long sides of the table where the operator can easily interact with the robot. Moreover, the orientations of the internal points are identical, meaning that only a translation is needed in that segment.

The starting base position that must be arbitrarily chosen for the initial inverse kinematics is equal to (-) 0.3 m.

From the eight possible starting configurations, four are discarded since the *elbow-down* option cannot be implemented for potential collisions with the working space. *Right-shoulder* solutions are not fit for this specific trajectory, for joint limits reasons. Hence, their presentation in this thesis is not necessary. Finally, two solutions of significant interest are presented: *left-shoulder wrist-down* and *left-shoulder wrist-up*.

3.3.1 Left-shoulder wrist-down

The starting configuration is shown in Figure 3.3.01. Due to the absence of a pose error in the following result, the white virtual EE frame is removed. In fact, the adoption of the pseudo-inverse Jacobian, along with the natural null space contribution, leads to a smooth robot motion: no saturation of mechanical limits, collisions, or singularities happens. Therefore, no significant task errors or position/orientation errors occur. This is also possible thanks to the wrist-down configuration, which is responsible for a straighter and broader robot posture.

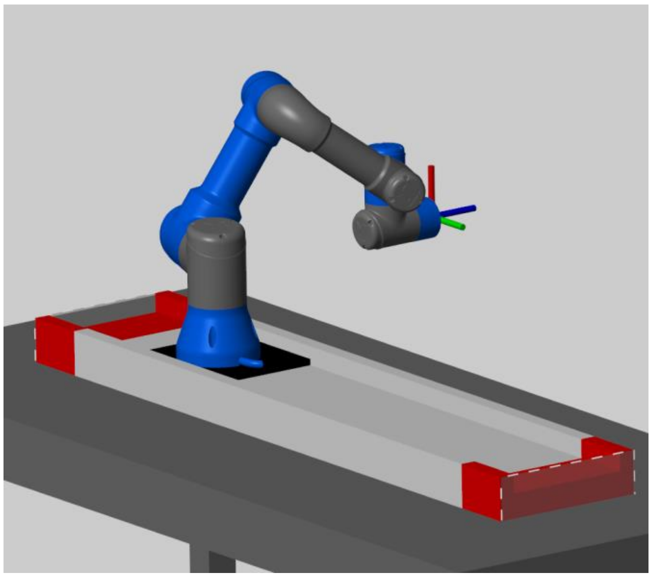

Figure 3.3.01 - Starting robot configuration, left-shoulder wrist-down

To accomplish the second waypoint (Figure 3.3.02), the base goes to the left in proximity to its limit. Even if this seems erroneous and undesirable, it is actually preferable. If it has gone to the middle of the table instead, as the projected gradient method for the guide would prefer, the transition to the third waypoint would happen with the EE pointing at the robot itself. The resulting movement would be much more unnatural and odder. On the contrary, through a torso rotation, the third waypoint is safely achieved, as demonstrated in Figure 3.3.03.

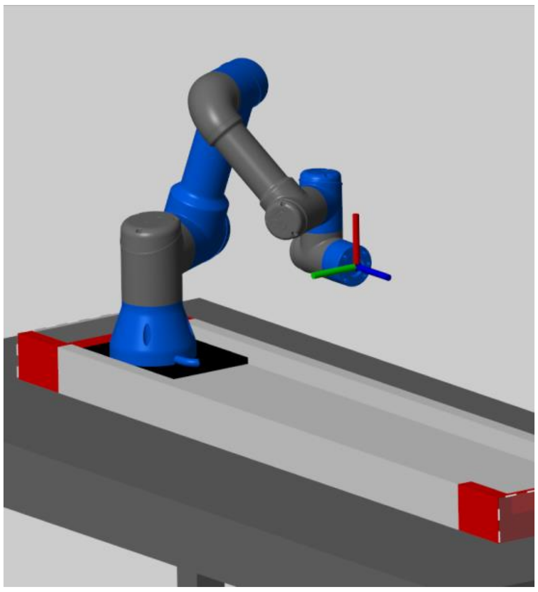

Figure 3.3.02 - Second waypoint robot configuration

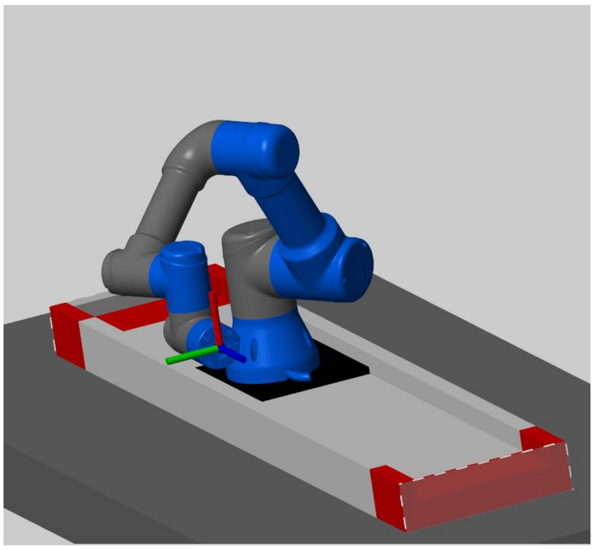

Figure 3.3.03 - Third waypoint robot configuration

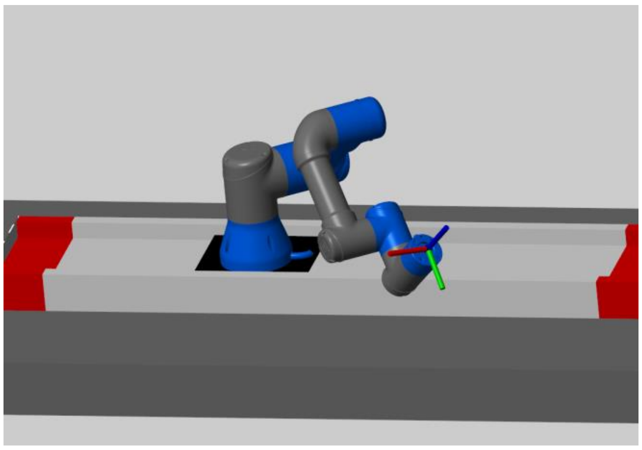

Figure 3.3.04 - Robot configuration. t = 9,88 s

The only excessive movement happens from the third to the fourth waypoint (Figure 3.3.04), where a great rotation of the TCP is required. Indeed, the base moves excessively, more than 0.3 m, and with a change of direction too (Figure 3.3.05). A more compact solution might be solved with a weight on the guide, for example.

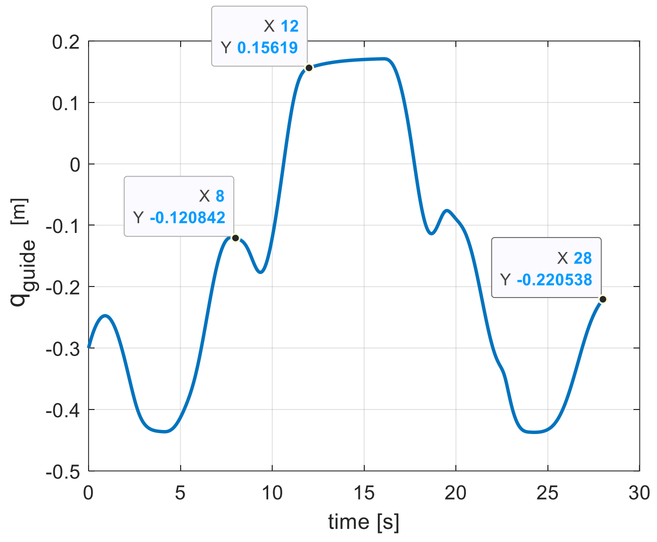

Figure 3.3.05 - Base position, highlight on the third and fourth waypoint, and the final value

The fourth waypoint is smoothly reached anyway (Figure 3.3.06), where the EE must stop (Figure 3.3.07) until the next segment starts. Even if it is not easily detectable by looking at Figures 3.3.06 and 3.3.07, small self-motions occur during the TCP halt, because the projected gradient method is still active. Hence, globally the robot changes its configuration, and it does not remain motionless.

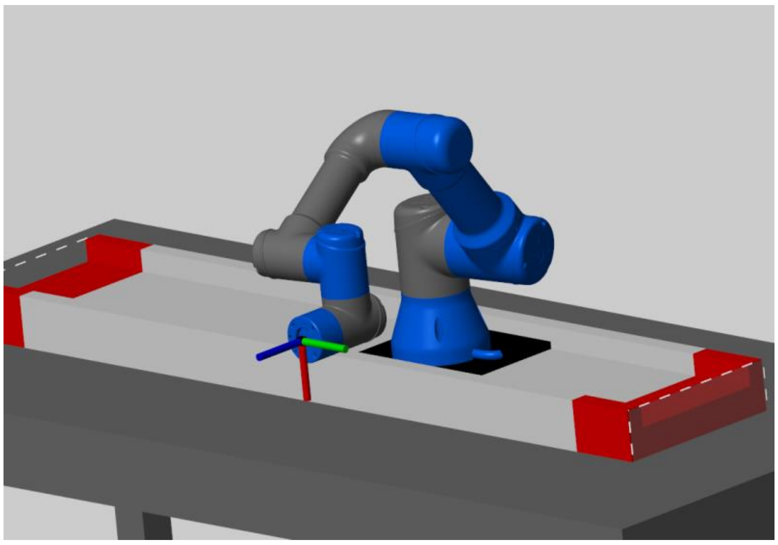

Figure 3.3.06 - Fourth waypoint robot configuration

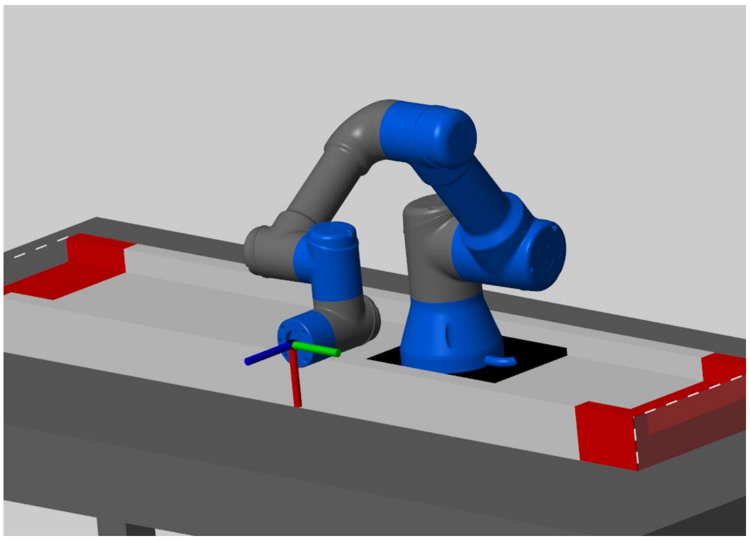

Figure 3.3.07 - Fifth waypoint robot configuration

On the way back, the sixth waypoint is achieved with the robot having a similar posture to what is already shown in Figure 3.3.03 for the third waypoint. Thus, its configuration is not illustrated. However, during the backward translation, there is almost a self-collision due to the elbow getting closer to its limit of 140 degrees (Figure 3.3.08), as demonstrated analytically in 3.3.09. This did not happen on the way out. Indeed, as said in the "2.1 Inverse kinematics" paragraph, cyclic motions in task space do not map to cyclic motions in joint space, so motions are non-repeatable in the joint space. This is evident by looking again at Figure 3.3.09: the joint configurations are not perfectly symmetrical with respect to the time instant of 14 seconds, which marks the two halves of the trajectory. However, in this scenario, they share some degrees of symmetry, and for these reasons, as already happened for the sixth waypoint, the seventh and last configurations are not included as images, due to their similarity to their corresponding solutions of the way out.

Finally, to remark what has just been said, the base finishes its movement occupying a different position with respect to its starting one (Figure 3.3.05).

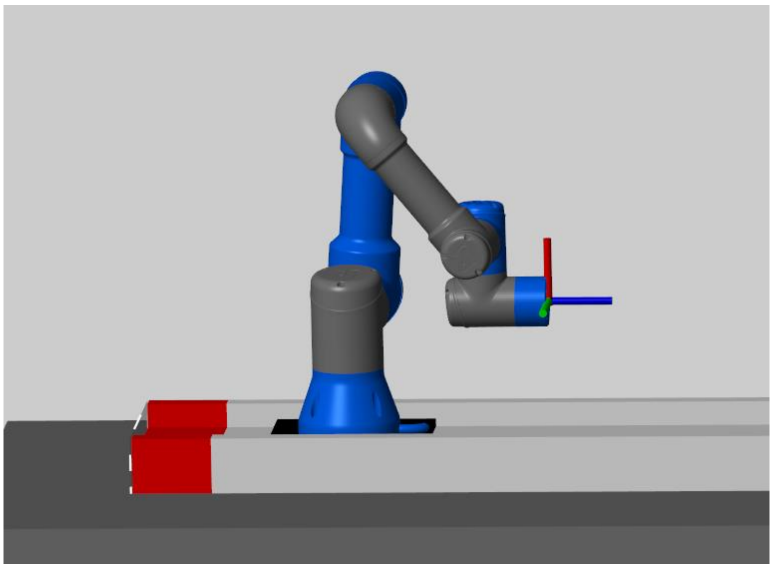

Figure 3.3.08 - Robot configuration. t = 22,54 s

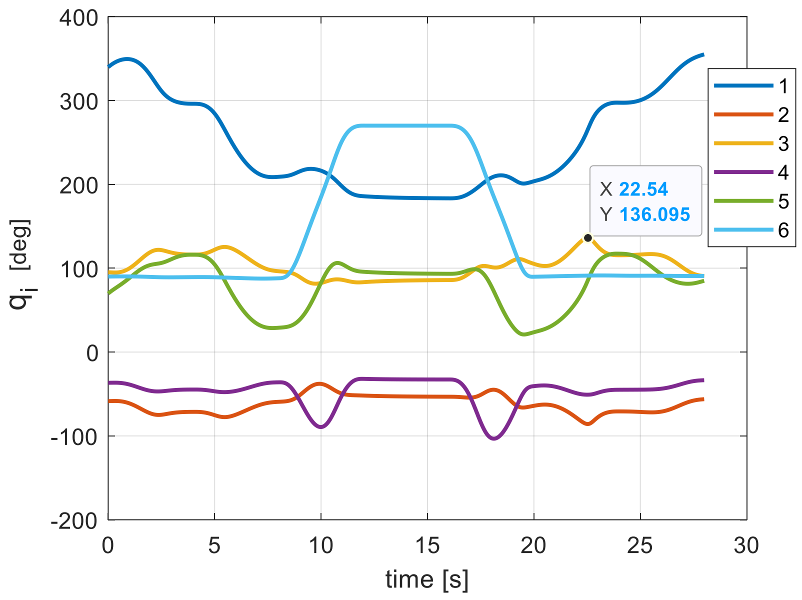

Figure 3.3.09 - Joint configurations, highlight on the elbow.

The succession of the EE orientation obtained during the trajectory is illustrated in Figure 3.3.10

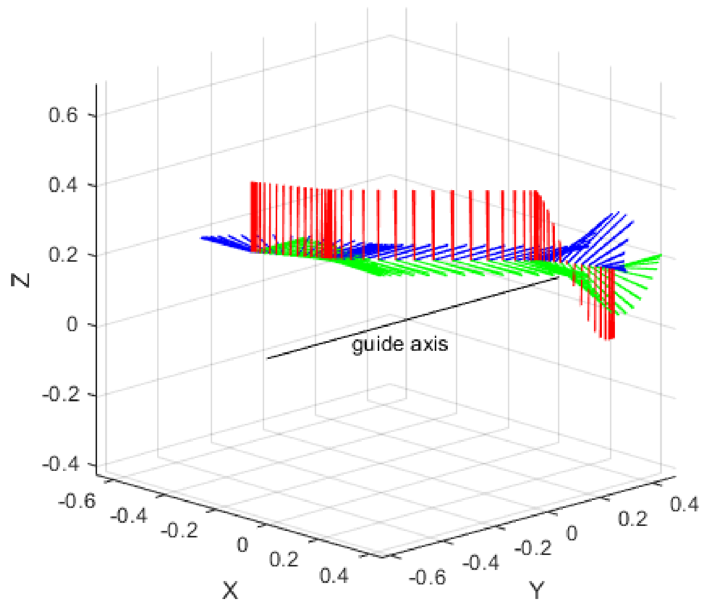

Figure 3.3.10 - EE frame motion

3.3.2 Left-shoulder wrist-up

In this configuration, the robot starts with its wrist up (Figure 3.3.11).

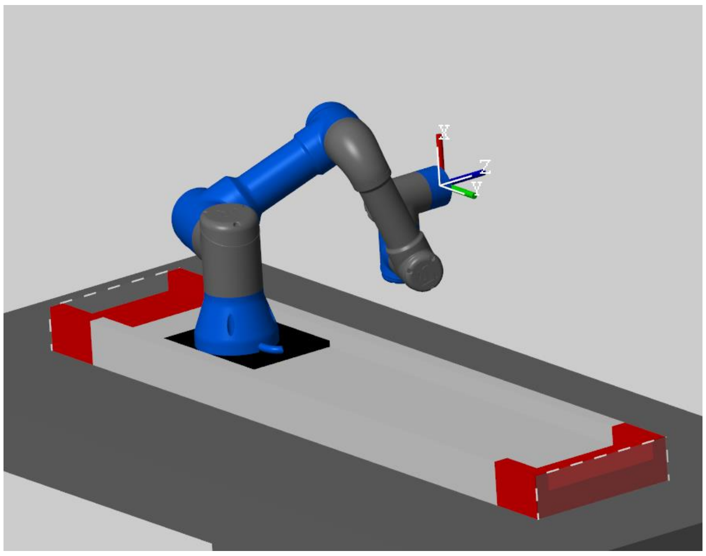

Figure 3.3.11 - Starting robot configuration: left-shoulder wrist-up

It is immediately noticeable that now the robot begins with a more restricted/narrow posture. However, until the reaching of the fifth waypoint (Figures 3.3.12,3.3.13 and 3.3.14), no issue arises. From then, the way back starts to be significantly different from the way out. Indeed, the sixth waypoint is obtained with the wrist in its null configuration

(Figure 3.3.15). This is the reference value for the projected gradient method, so this is a reason for its achievement.

Subsequently, in the next segment, a self-collision occurs (Figure 3.3.16). This is due to the non-optimal wrist displacement that obliges the robot to reach an even narrower posture. Therefore, the elbow limit is attained, and its long saturation (Figure 3.3.17) represents the willingness of the algorithm to overcome it to fulfill the task. From Figure 3.3.17, it is also noticeable that during the translation of the EE q_5 keeps its zero value throughout the segment. This could also be dangerous for a wrist singularity. However, since an angular velocity to the TCP is not requested, it does not happen.

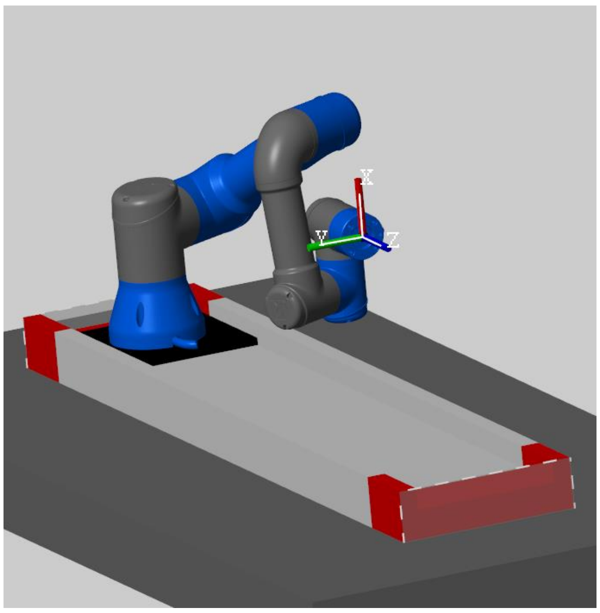

Figure 3.3.12 - Second waypoint robot configuration

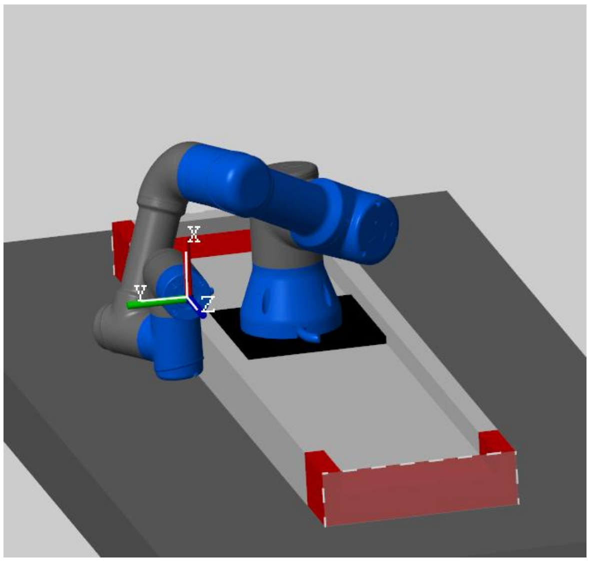

Figure 3.3.13 - Third waypoint robot configuration

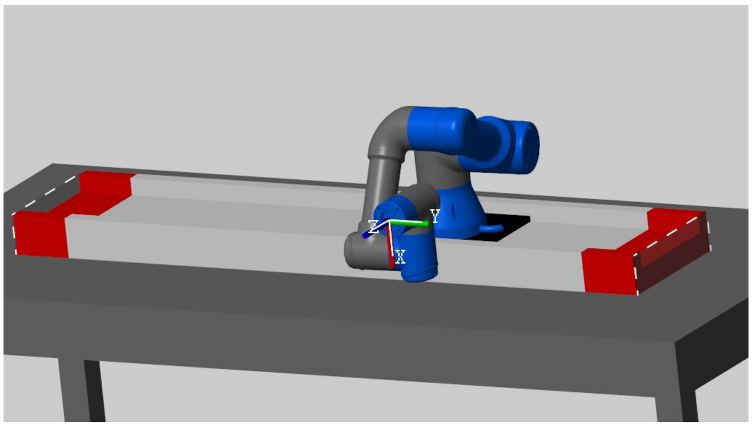

Figure 3.3.14 - Fourth (and fifth) waypoint robot configuration

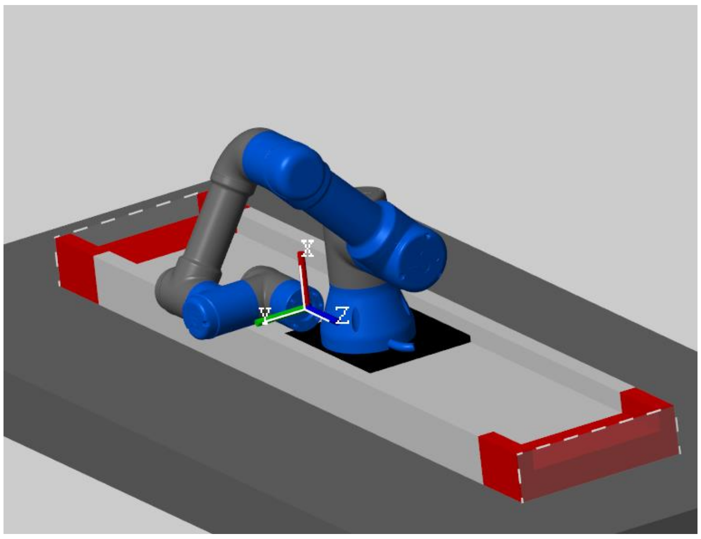

Figure 3.3.15 - Sixth waypoint robot configuration

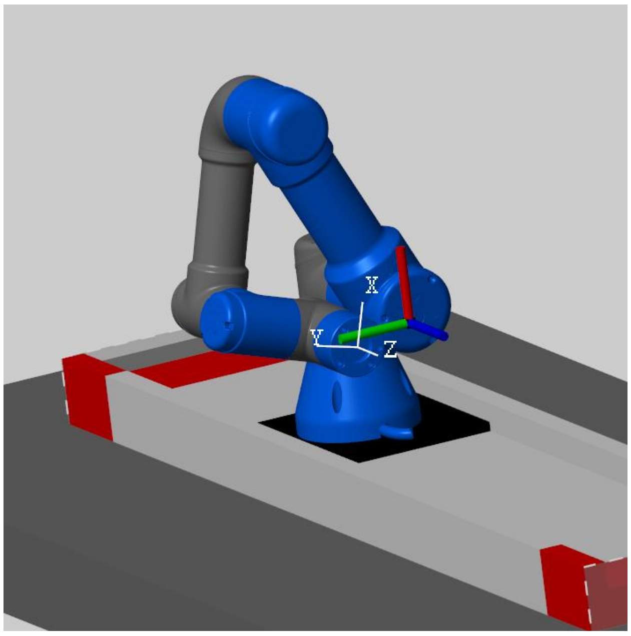

Figure 3.3.16 - Self-collision. t = 21,9

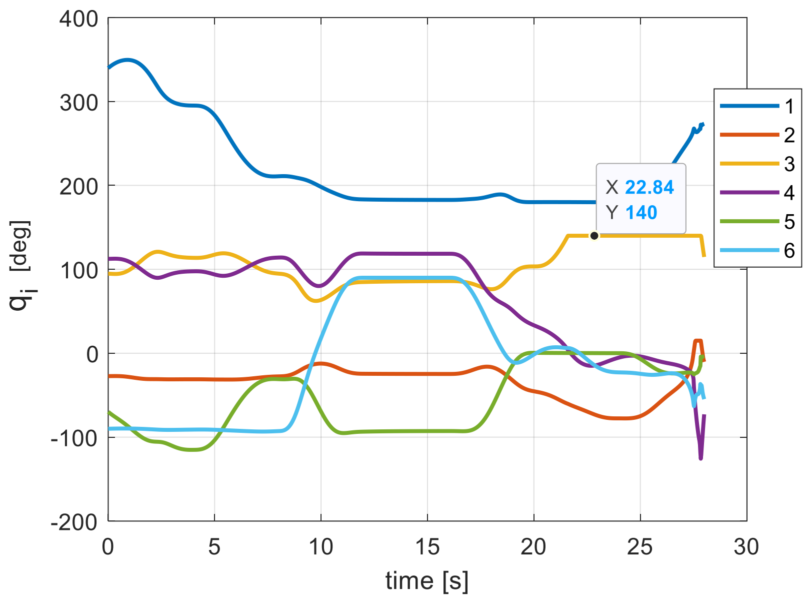

Figure 3.3.17 - Joint configurations, highlight on the elbow saturation

A possible solution for avoiding the elbow mechanical joint limit is by simply raising its null space velocity gain k_0. Therefore, from the common reference value of 100, a value of 150 is chosen only for the elbow. Recalling that the reference value for the elbow is associated with the arm in its full stretch form, this implies a more elongated robot posture in general.

To obtain the sixth waypoint, the wrist does not rotate, so it remains in its up configuration (Figure 3.3.18). In this manner, the way back is naturally achieved with a close symmetry to the way out.

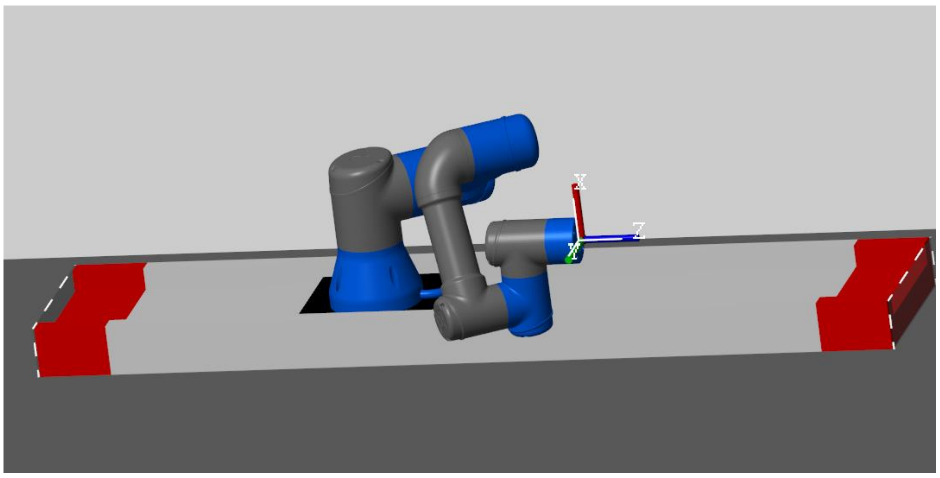

Figure 3.3.18 - Sixth waypoint robot configuration. Elbow $k_0 = 150$

However, the robot occupies more space by being wider; consequentially, the base has to move more to accomplish the task. This leads to the saturation of the guide (Figure 3.3.19), which happens twice: briefly on the way out, and longer on the way back (Figure 3.3.20). For this reason, a pose error occurs (Figure 3.3.21), which is compensated by the controller. Now a true kinematic control would be possible since the compensation does not generate great accelerations for long periods (Figures 3.3.22 and 3.3.23); although a small discontinuity still happens at the time instant of 22,76 seconds, caused by the "impact" with the end stroke (Figure 3.3.19).

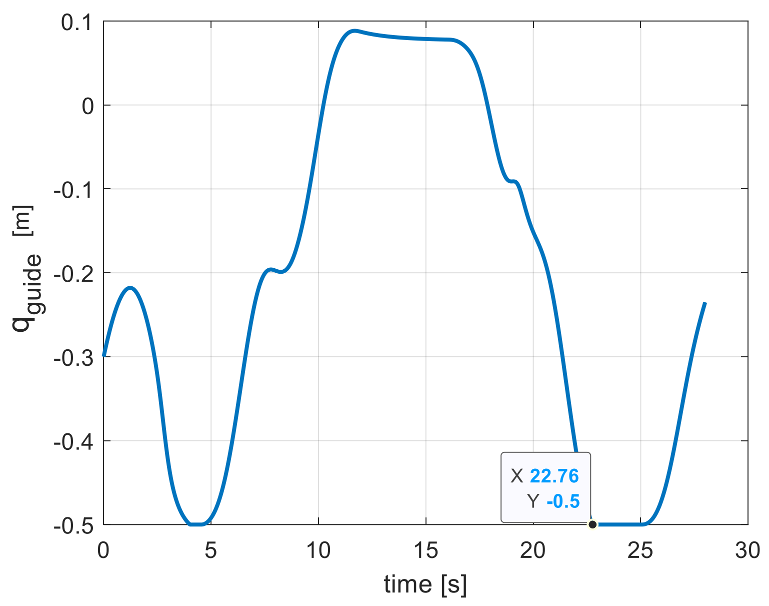

Figure 3.3.19 - Base position with two saturations, highlight on the second one. Elbow $k_0 = 150$

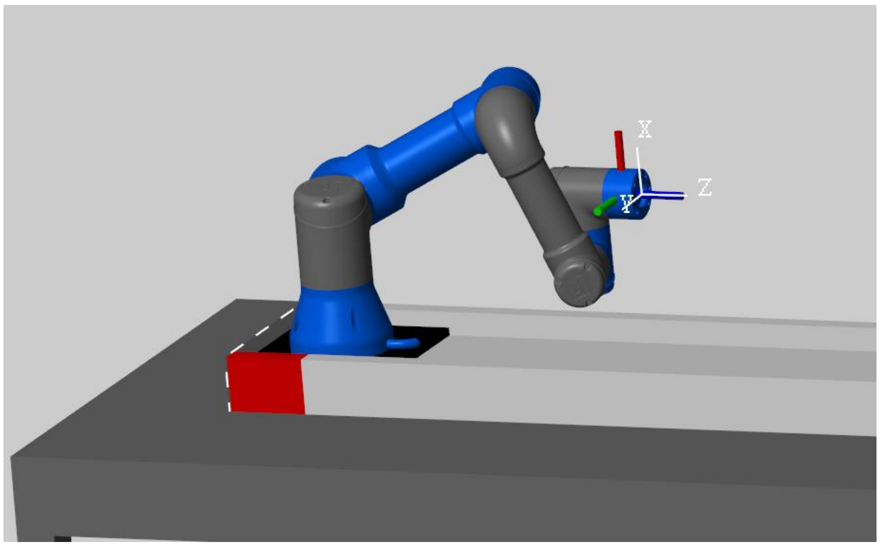

Figure 3.3.20 - Robot configuration on the way back. t = 23,9 s. Elbow $k_0 = 150$

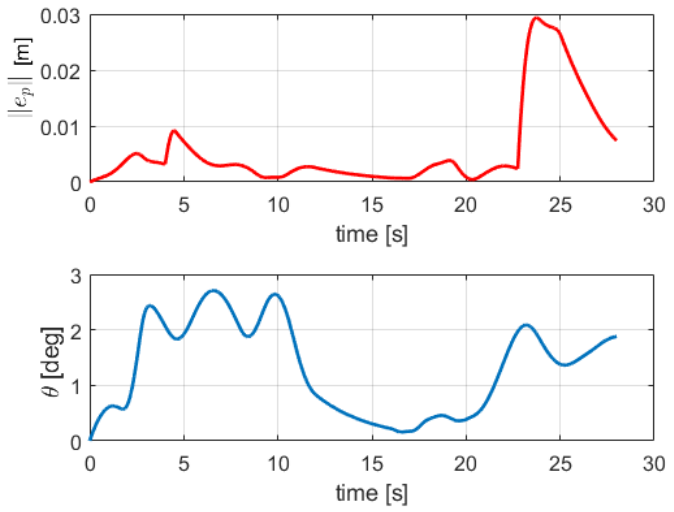

Figure 3.3.21 - Pose error caused by the guide saturation. Elbow $k_0 = 150$

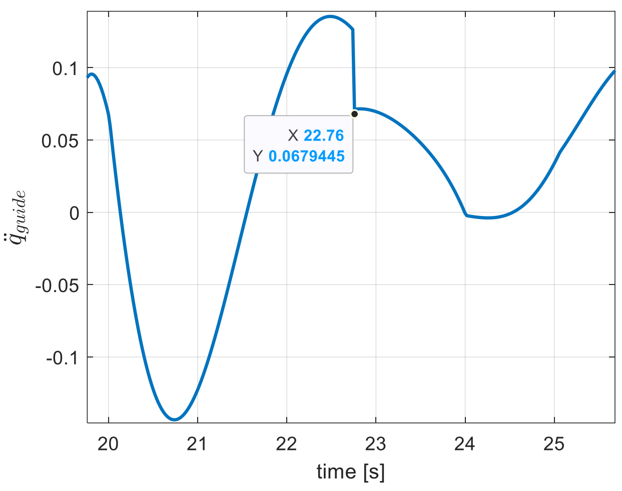

Figure 3.3.22 - Base acceleration, zoom around t = 22.76. Elbow $k_0 = 150$

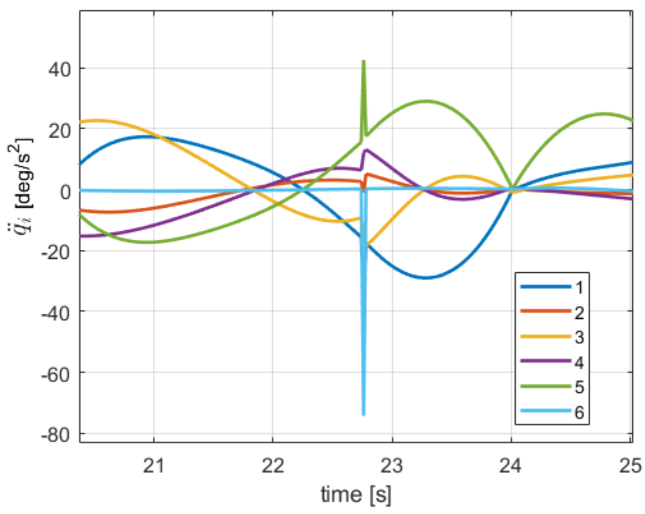

Figure 3.3.23 - Joint accelerations, zoom around t = 22.76 s. Elbow $k_0 = 150$

Another solution could be to act on the trajectory planning by changing the desired orientation of the seventh waypoint (Figure 3.3.24) for a better wrist movement. Indeed, in addition to the translation, the EE has a rotation movement too on the way back.

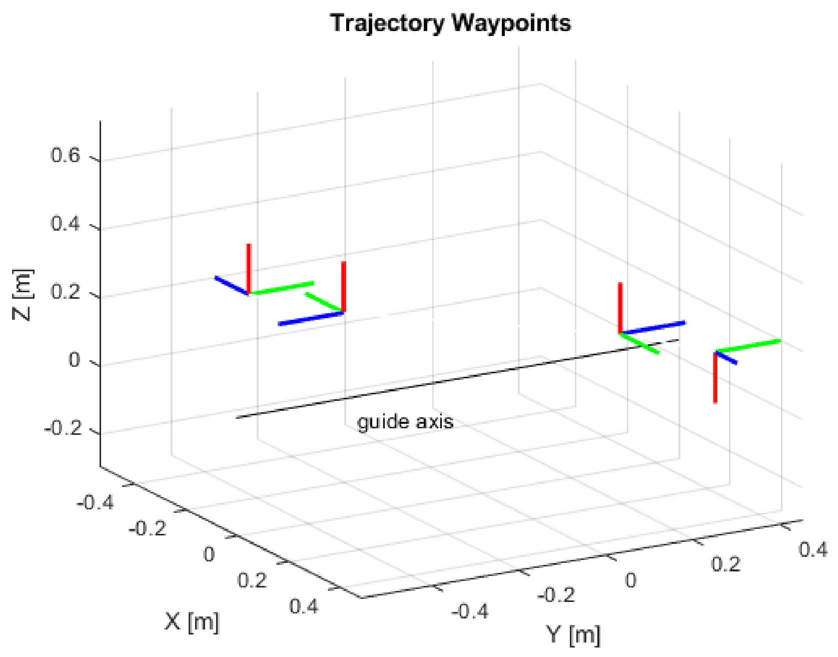

Figure 3.3.24 - Way back trajectory waypoints

Obviously, the way out does not change so the configuration of the sixth waypoint is the same as Figure 3.3.15. From then, to accomplish the task, since an angular velocity is requested to the EE, the wrist rotates around its first joint, while its second and critical one remains in its null configuration.

However, this is dangerous for a potential wrist singularity, which eventually takes place (Figure 3.3.25). Indeed, the robot should have to do an instantaneous wrist flip from the down configuration to its up (Figure 3.3.26), which surely cannot be achieved due to the physical limits of the actuators (Figure 3.3.27). This naturally implies a pose error, which cannot be recovered by the controller action, because in a real application the robot would immediately stop as soon as the singularity happens.

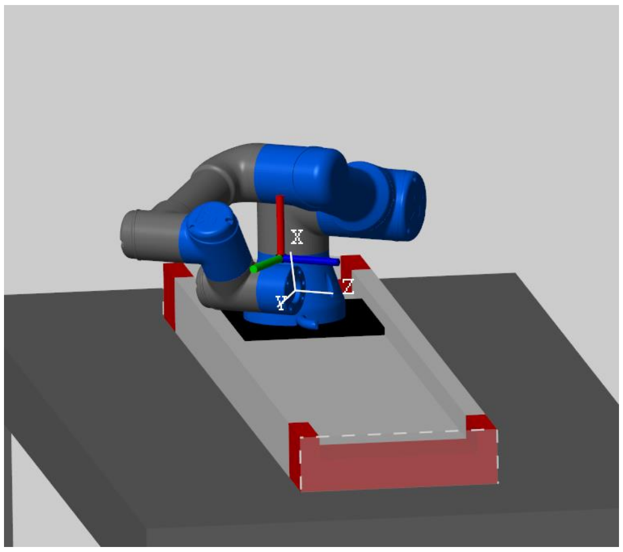

Figure 3.3.25 - Robot configuration as soon as the singularity occurs. t = 21,58 s

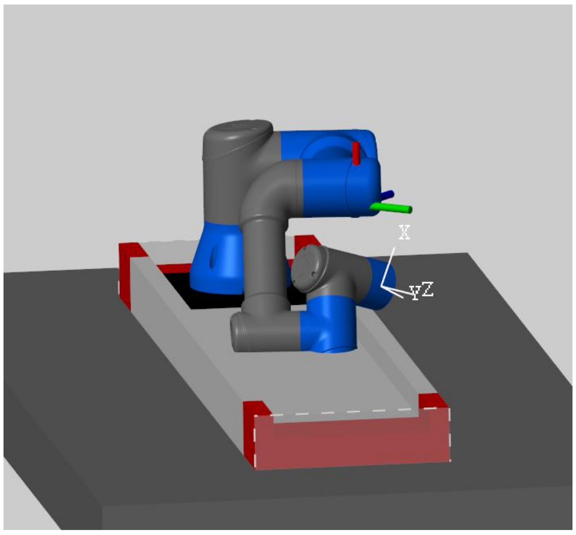

Figure 3.3.26 - Robot in its wrist-up solution after the wrist-flip

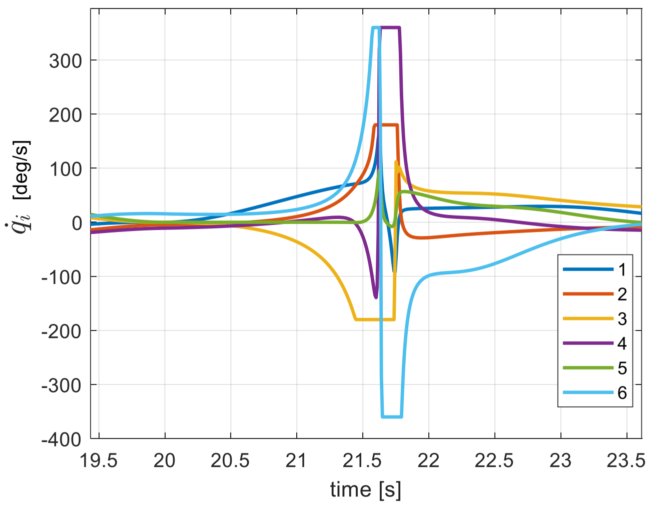

Figure 3.3.27 - Joint velocities during the singularity

Analytically, as presented in the “Model” chapter, the proximity of a singularity can be monitored through the last singular value σ. In Figure 3.3.28, it can be noticed the value plummeted to almost zero during the singularity occurrence. The counter in Figure 3.3.29, which detects the number of time instants where σ goes below the threshold value of 0.1., reaches a value of 90, after it is reset since σ returns to acceptable levels.

To overcome the problem of inverting differential kinematics in the neighborhood of a singularity, the missing DLS method is implemented here for the first time in this work. Starting for example with a damping factor μ equal to 0.03, but without the controller being active, the intrinsic error deriving from the method just keeps growing throughout the trajectory (Figure 3.3.30) despite the saturation being prevented. Obviously, this is not sustainable in the long term, so further discussion of results for this type of solution is not worthwhile.

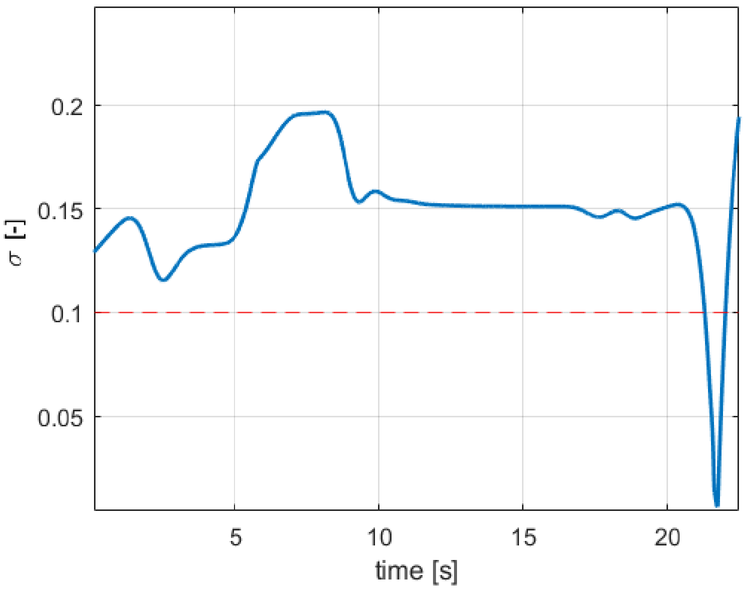

Figure 3.3.28 - Last singular value (Pseudo-inverse)

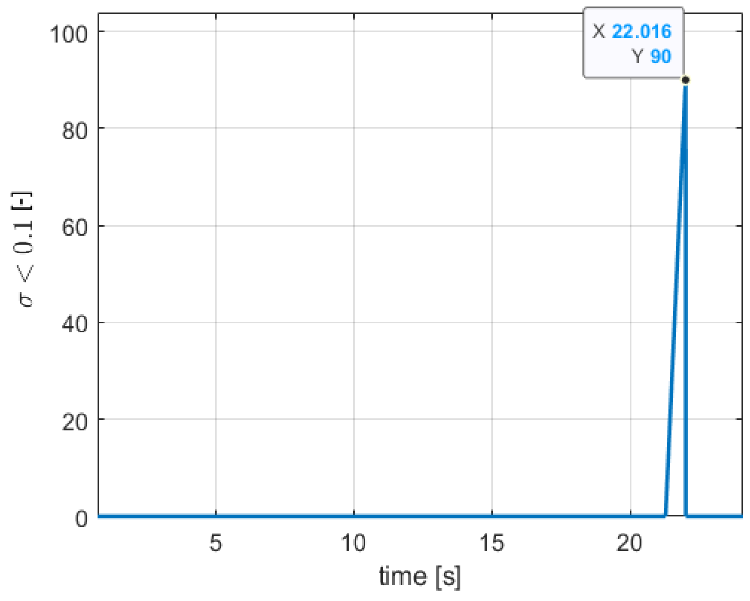

Figure 3.3.29 - Singularity counter (Pseudo-inverse)

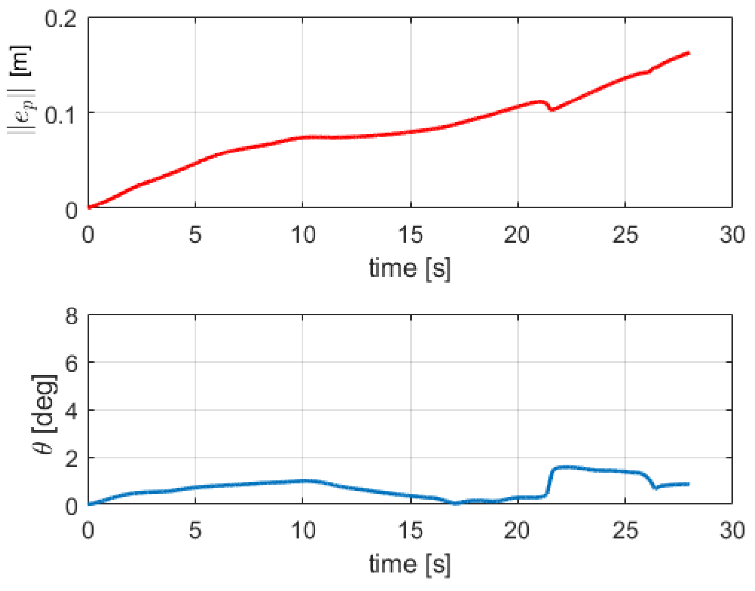

Figure 3.3.30 - Pose error, $\mu = 0.03$, while the controller being deactivated

To cope effectively with such an issue, a closed-loop solution must be found. Nevertheless, only from $\mu = 0.03$ and beyond, the saturation is successfully avoided. Indeed, if for example the damping factor is equal to 0.02, its damping effect on the joint velocities is not quick enough (Figure 3.3.31).

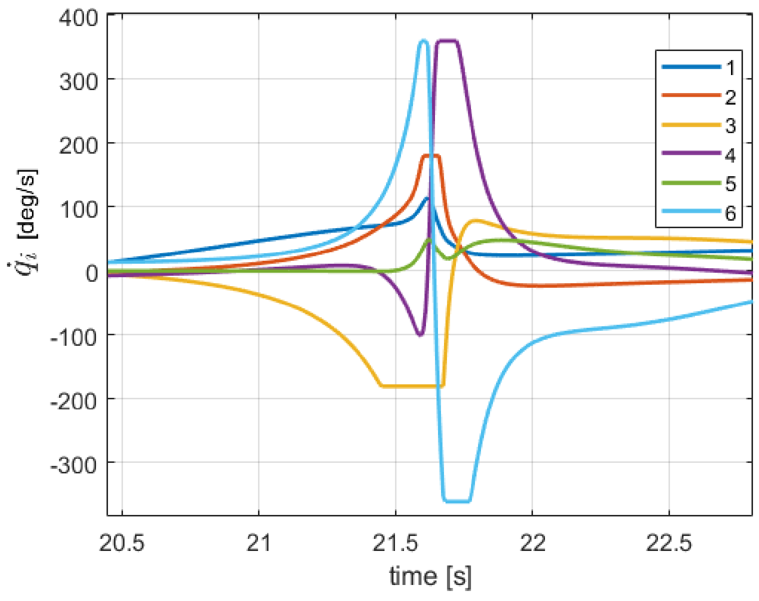

Figure 3.3.31 - Joint velocities, μ = 0.02, five saturations occur

With μ = 0.03, the saturation is reached briefly only for the elbow and the linear guide (Figures 3.3.32 and 3.3.33).

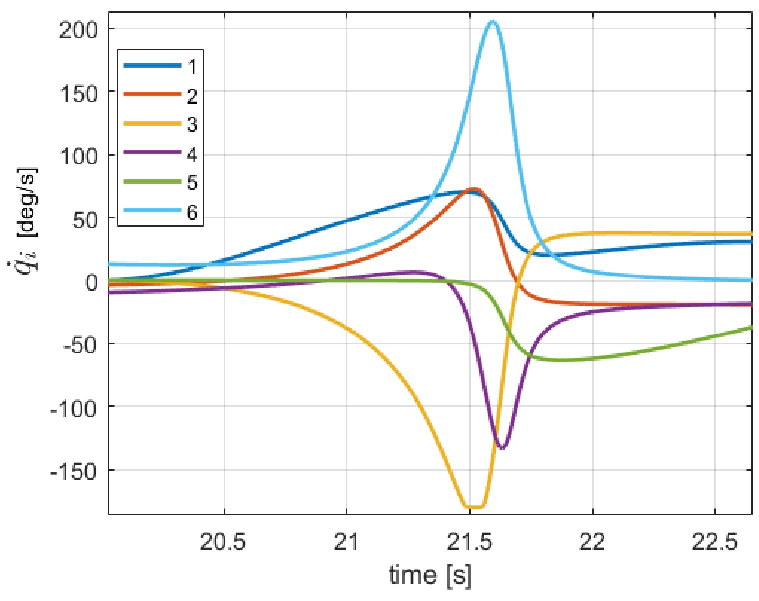

Figure 3.3.32 - Joint velocities, μ = 0.03

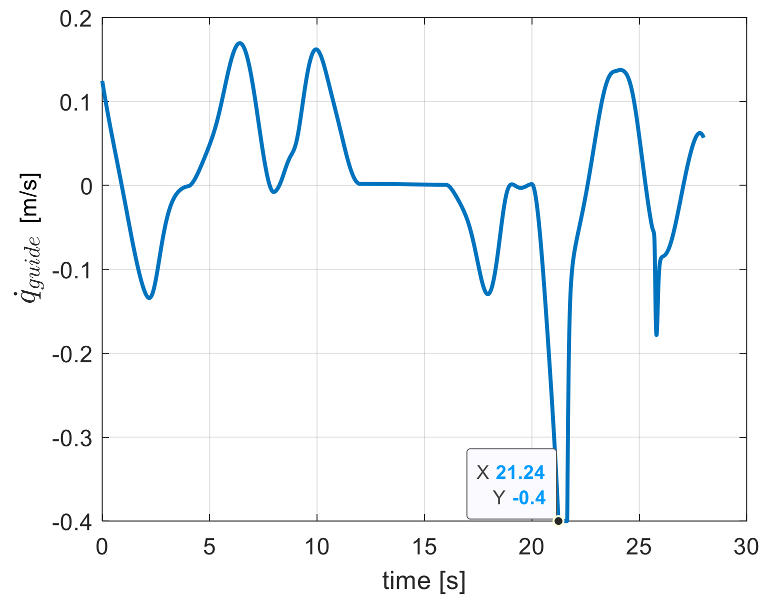

Figure 3.3.33 - Base velocity, μ = 0.03

Overall, this is not a significant issue, the resulting pose error is very low: 0.04 m for the EE position, and 1.17 degrees for the orientation (Figure 3.3.34). Indeed, the motion is still smoothly crossing the critical time instant, as shown in Figure 3.3.35.

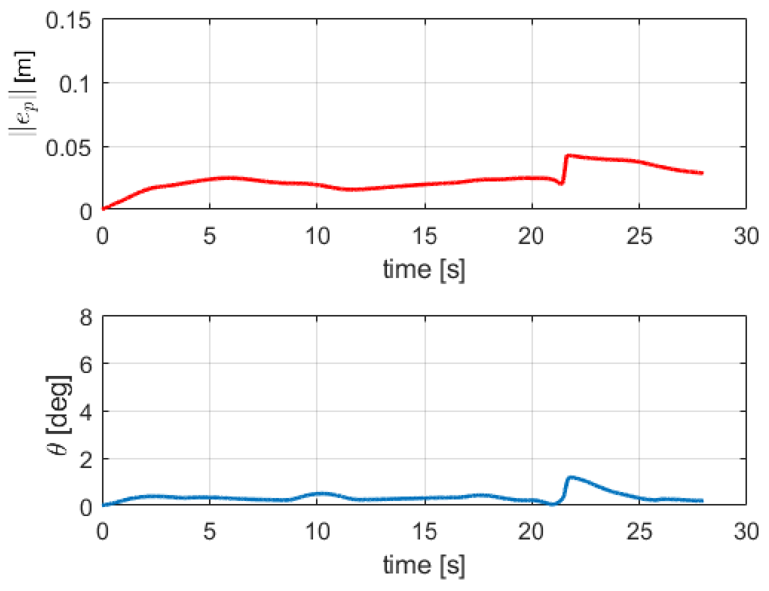

Figure 3.3.34 - Pose error, μ = 0.03

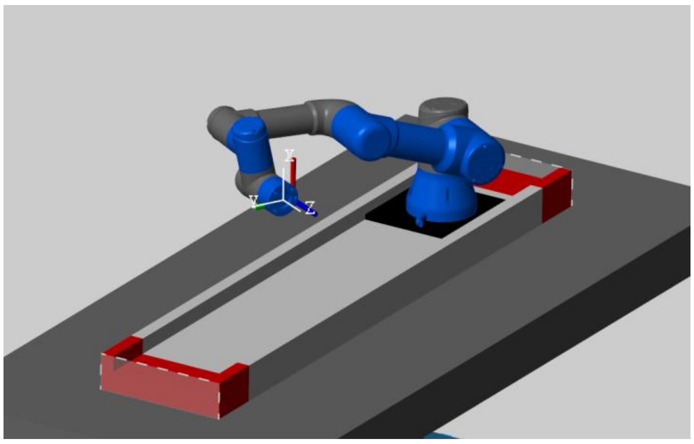

Figure 3.3.35 - Robot configuration in proximity of a singularity. t = 21,67. μ = 0.03

The robot is able to complete its requested rotation, and finally, the seventh waypoint is achieved (Figure 3.3.36). Subsequently, the last desired waypoint is obtained (Figure 3.3.37).

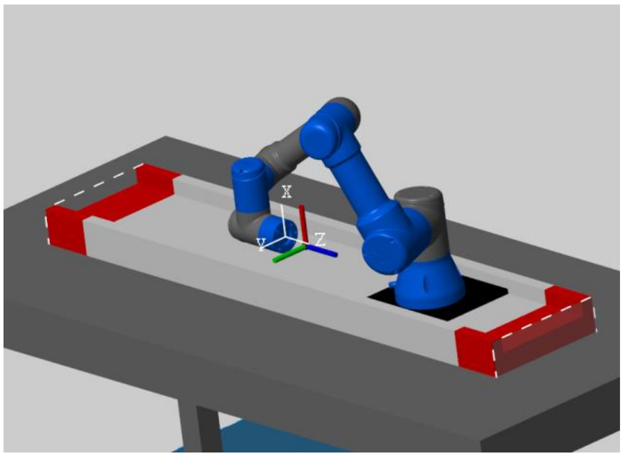

Figure 3.3.36 - Seventh waypoint robot configuration. μ = 0.03

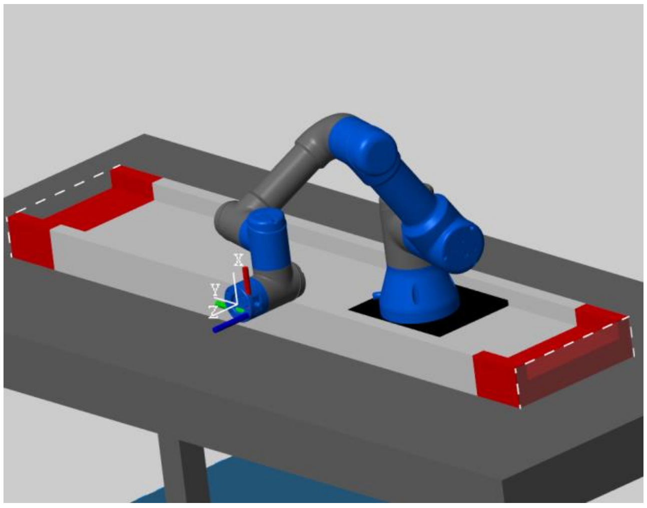

Figure 3.37 - Final robot configuration. μ = 0.03

However, in this last segment, q_5 crosses its null value again while the wrist rotates (Figure 3.3.38). Thus, velocities briefly rise for a second time but with much lower values (Figure 3.3.39).

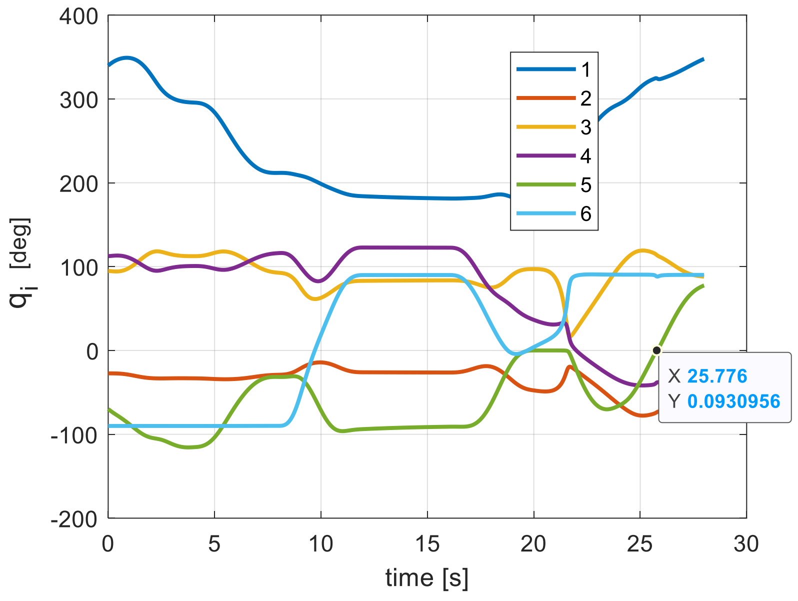

Figure 3.3.38 - Joint configurations, highlight on $q_5 = 0$. *μ = 0.03*

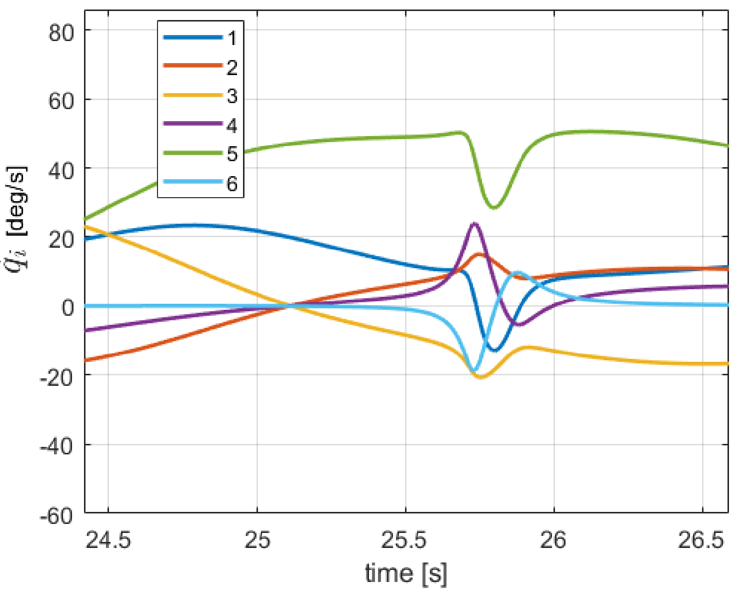

Figure 3.3.39 - Joint velocities, zoom on the second raise. μ = 0.03

The DLS robust behavior while crossing singularities can be also seen in Figure 3.3.40. In this case σ does not get close to zero, where the actual singularity happens, as much as before for the Pseudo-inverse method (Figure 3.3.28).

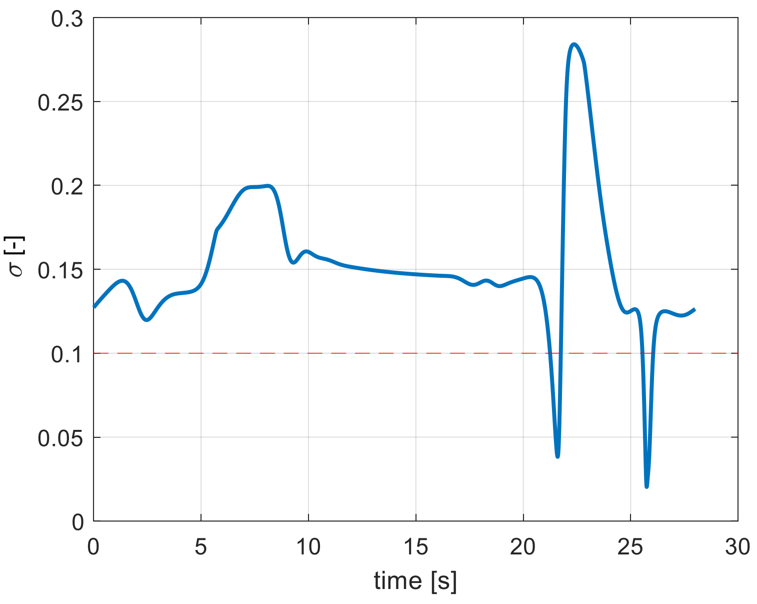

Figure 3.3.40 - Last singular value, μ = 0.03

By increasing the damping factor up to 0.05, the saturation of the elbow is avoided (Figure 3.3.41). Nevertheless, it still happens for the linear guide for a very short period of 0.3 s (Figure 3.3.42).

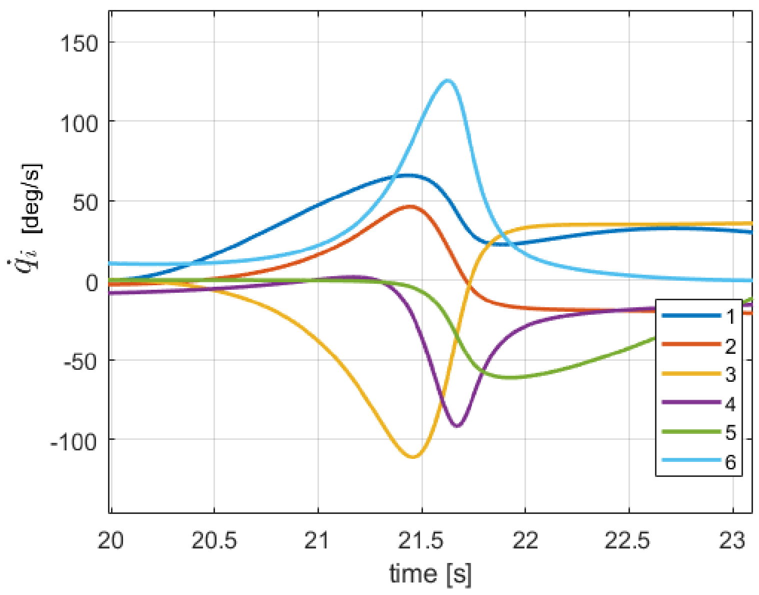

Figure 3.3.41 - Joint velocities, μ = 0.05

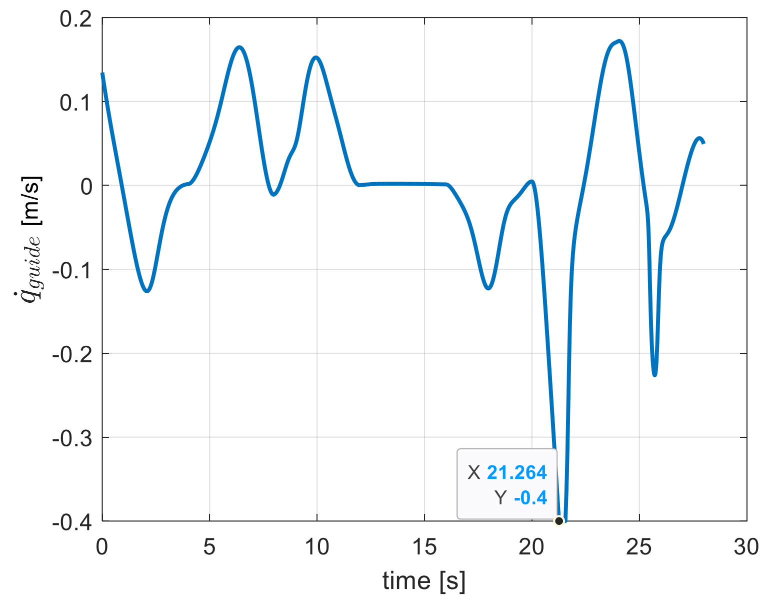

Figure 3.3.42 - Base velocity. μ = 0.05

The pose error can easily be considered still tolerable (Figure 3.3.43). It is worth marking that it is derived from the intrinsic task error produced by the DLS method (Figure 3.3.44). Indeed, as explained in the model chapter, the error is generated from a velocity point of view, and it is recovered from a pose point of view by the controller.

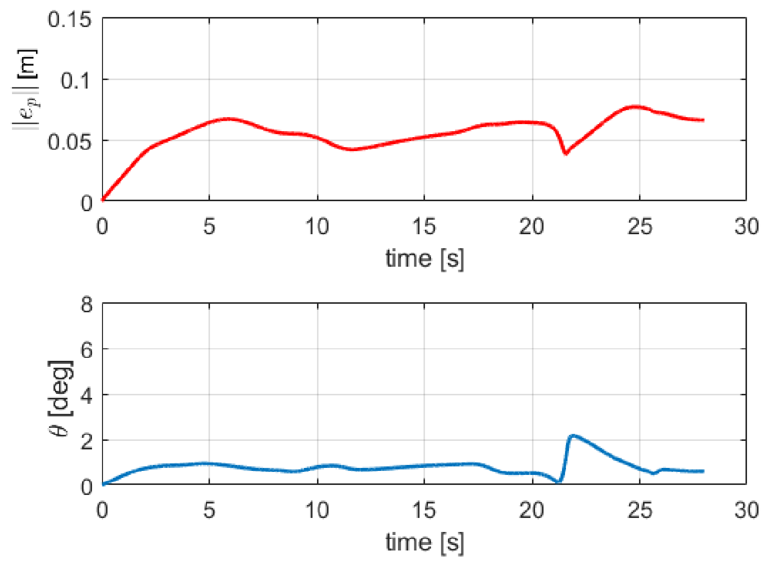

Figure 3.3.43 - Pose error, μ = 0.05

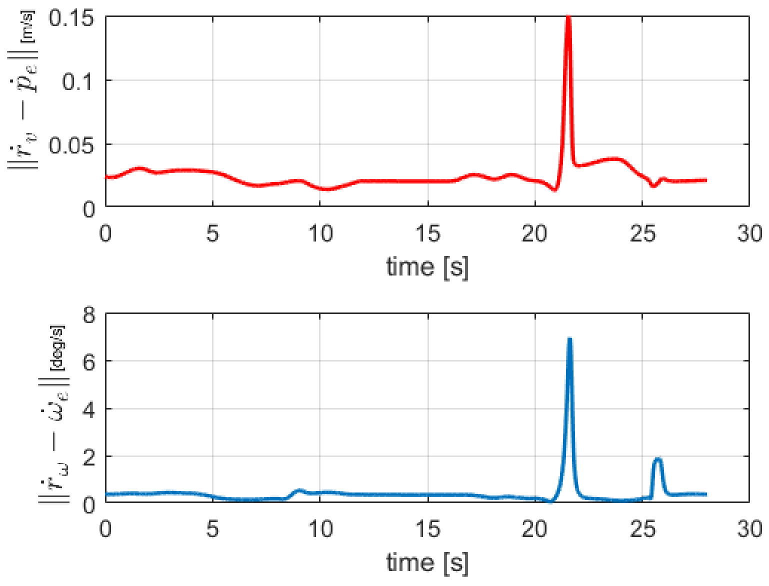

Figure 3.3.44 - Task error, μ = 0.05

With $\mu = 0.07$, even the base velocity does not approach its limit anymore (Figure 3.3.45). Nevertheless, the pose error cannot be considered acceptable. Particularly, the position error repeatedly exceeds 0.1 m, a quantity which can be arbitrarily thought of as inadmissible (Figure 3.3.46).

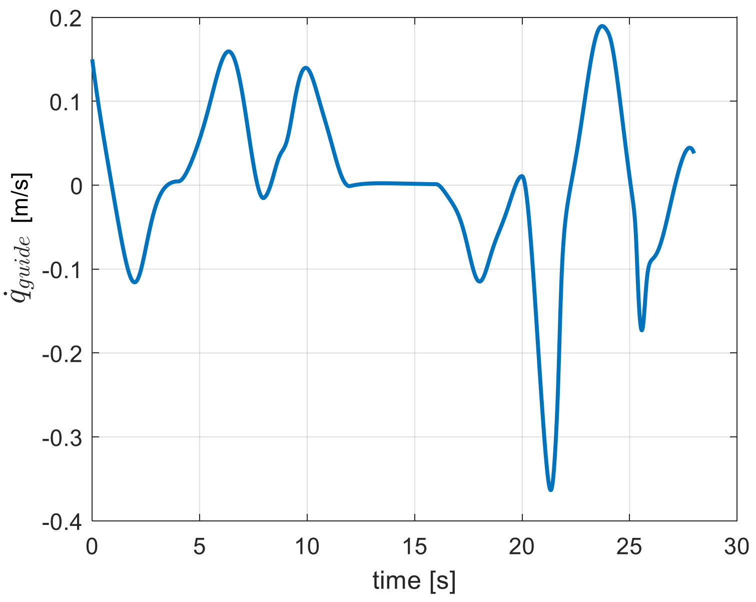

Figure 3.3.45 - Base velocity, μ = 0.07

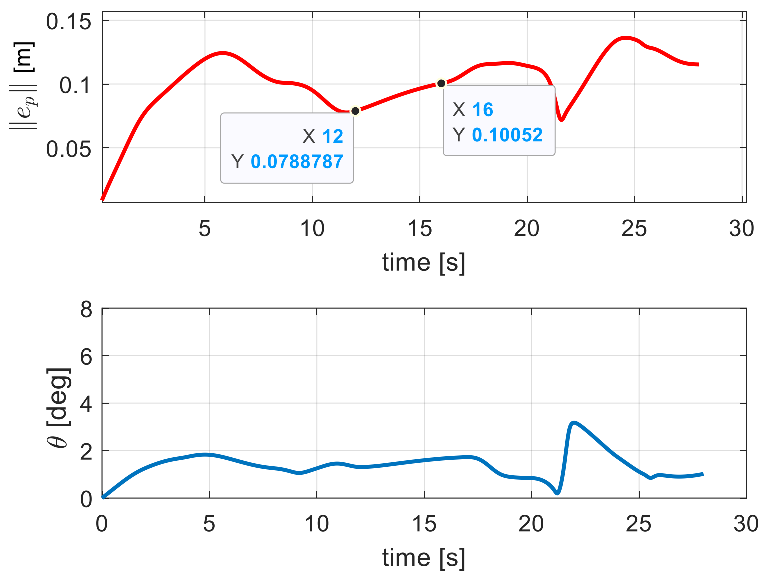

Figure 3.3.46 - Pose error, μ = 0.07

As stated in the "Inverse kinematics" paragraph, the projected gradient method applied in the DLS does not produce proper self-motions, since it induces a small EE movement. This can be easily detectable when the TCP must be still, as it occurs from the fourth waypoint (Figure 3.2.47) to the fifth (Figure 3.2.48). Analytically with μ = 0.07, during the segment in question, the norm of the position error increases by just 21 mm (Figure 3.2.46), thanks to the controller action. This error is naturally proportional to the damping factor, and overall it can be considered neglectable for lower values.

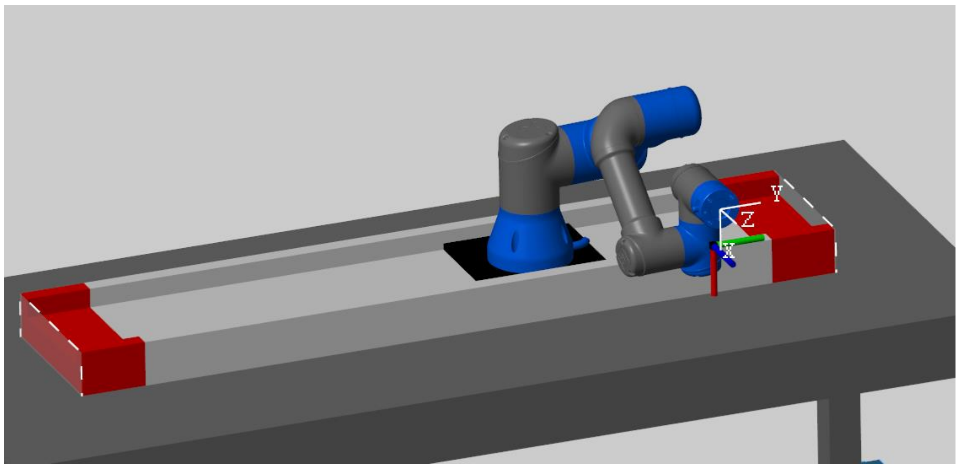

Figure 3.3.47 - Fourth waypoint robot configuration, μ = 0.07

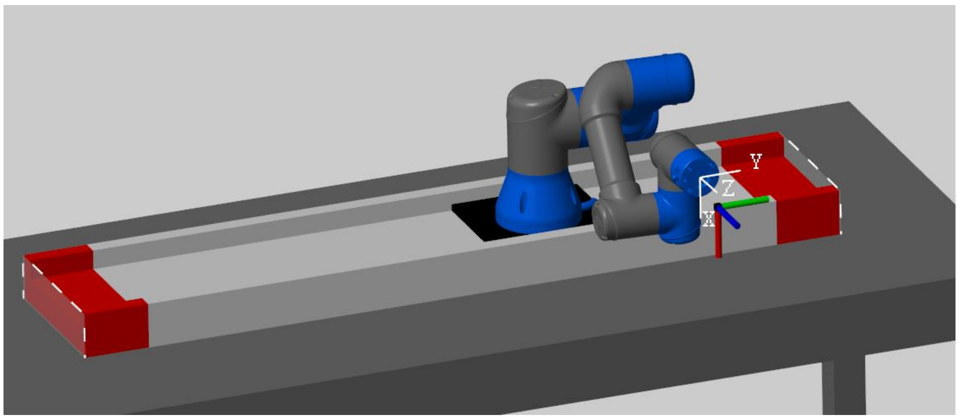

Figure 3.3.48 - Fifth waypoint robot configuration, $\mu = 0.07$

So far, the controller gains $\boldsymbol{K_P}$ and $\boldsymbol{K_O}$ have been always equal to one third. By increasing them for example to 1.5 and returning to $\mu = 0.05$, it is possible to almost eliminate the intrinsic error before the singularity occurrence (Figure 3.3.49).

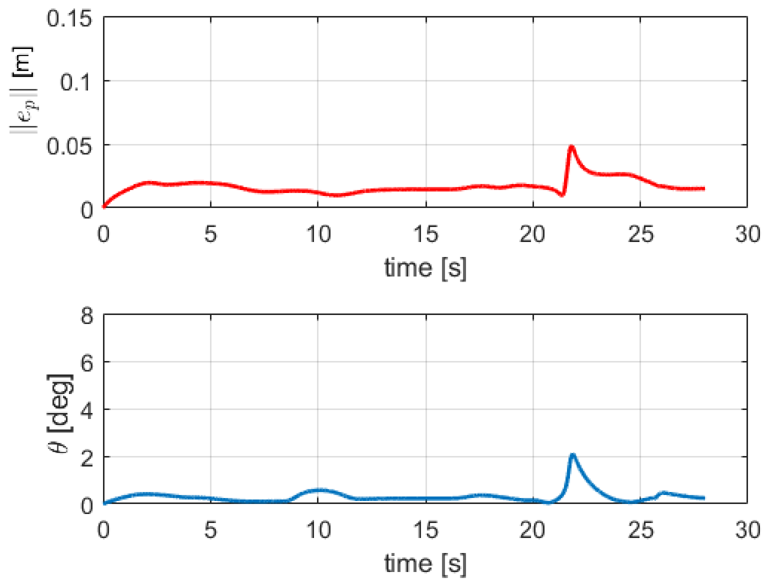

Figure 3.3.49 - Pose error. $\mu = 0.05$. $K_P = K_O = 1.5$

It is possible to consider the kinematic controller sufficient if its effect does not generate an elevated EE acceleration. To analyze this, a comparison is made between the obtained acceleration and the desired one (Figure 3.3.50).

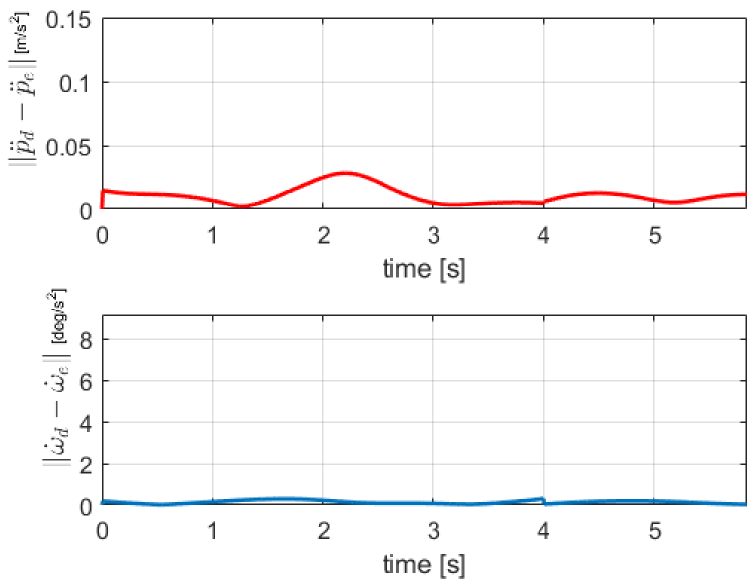

Figure 3.3.50 - "Task" acceleration error. μ = 0.05. $K_P = K_O = 1.5$

This is not properly a task error since the EE acceleration is not imposed. Indeed, the algorithm is not of the second order, so an acceleration command does not exist. By looking at Figure 3.3.50, it is evident that its action on the acceleration is almost non-existent.

However, for the dynamics effects it is better to analyze the obtained joint accelerations (Figure 3.3.51), which are not great anyway during the first seconds of the trajectory.

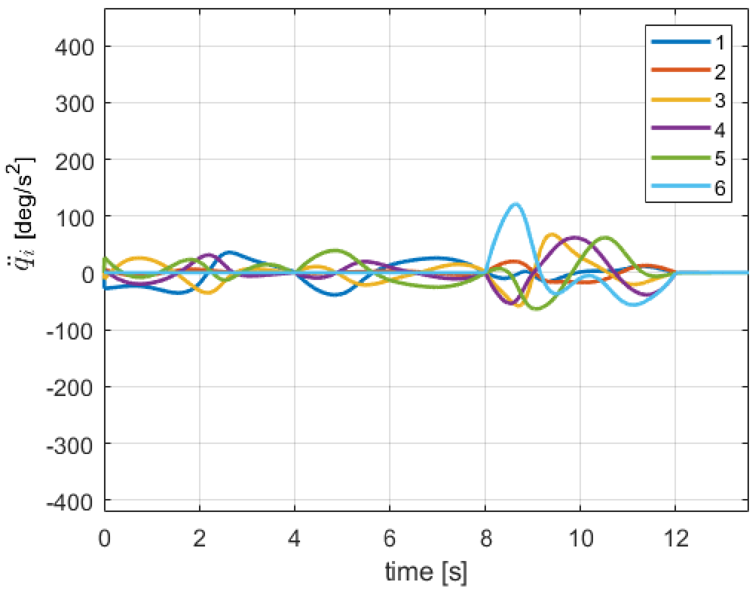

Figure 3.3.51 - Joint accelerations. μ = 0.05. $K_P = K_O = 1.5$

Of course, when the singularity occurs, the acceleration increases sharply (Figure 3.3.52).

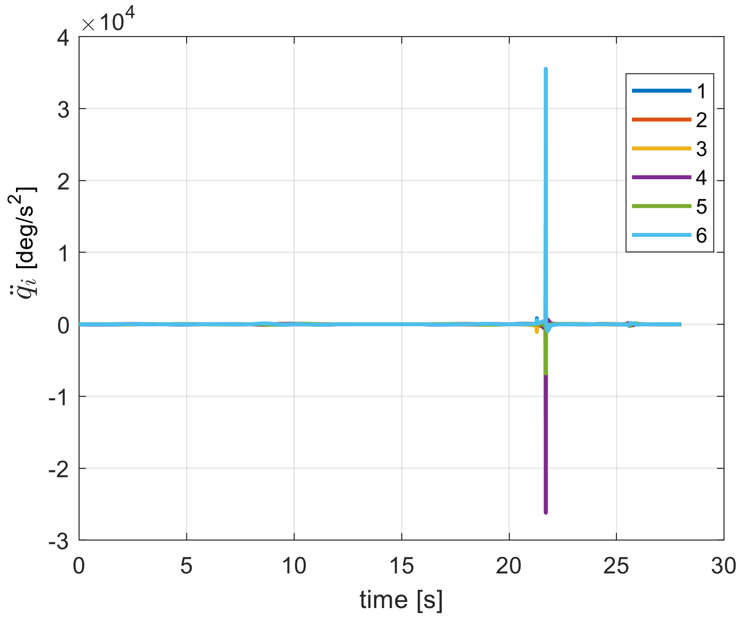

Figure 3.3.52 - Joint accelerations during the singularity. μ = 0.05. $K_P = K_0 = 1.5$

The huge values are not only due the singularity itself but also to the high gains of the controller. In fact, with $\boldsymbol{K}_{P,i} = \boldsymbol{K}_{0,\mathrm{i}} = \frac{1}{3}$, even if the joint (and base) accelerations are still huge, they do not exceed a thousand of deg/s^2 (Figures 3.3.53 and 3.3.54). Obviously, even with lower gains, during the singularity, the kinematic controller would not be enough.

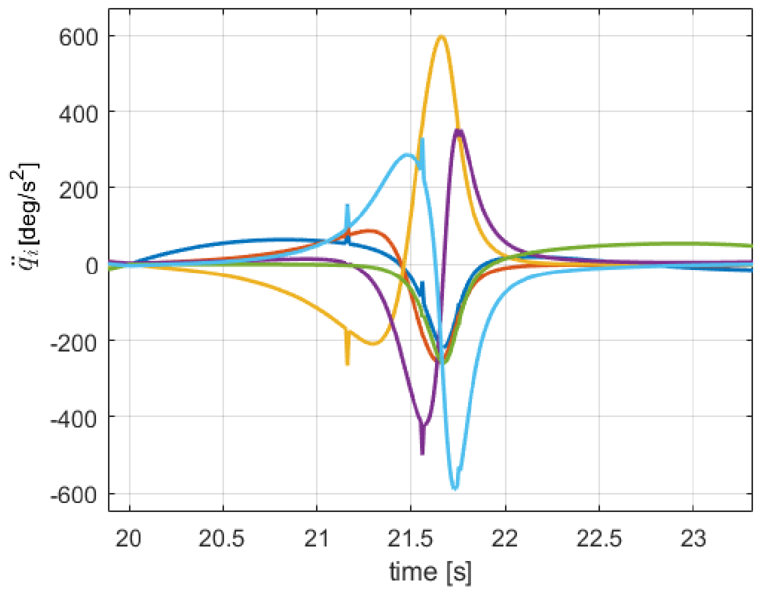

Figure 3.3.53 - Joint accelerations during the singularity. μ = 0.05. $K_P = K_0 = \frac{1}{3}$

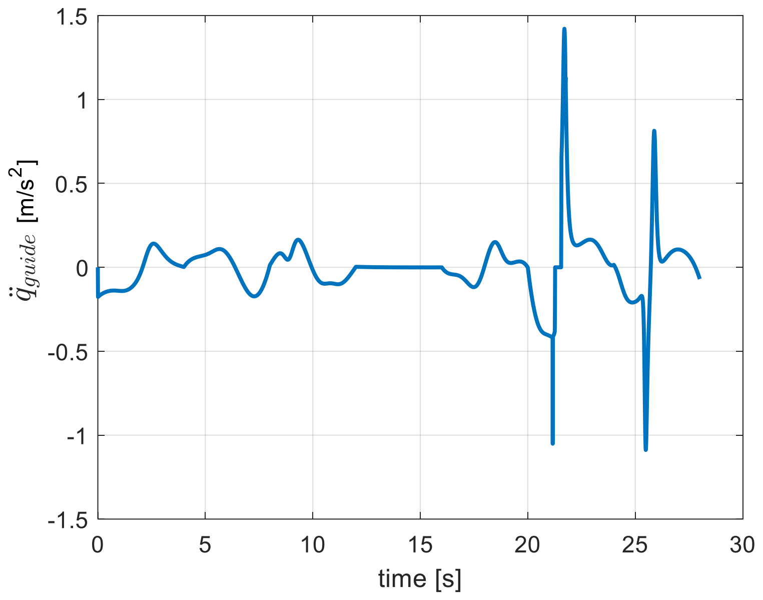

Figure 3.3.54 - Base acceleration. μ *= 0.05.* $K_P = K_0 = \frac{1}{3}$

CONCLUSIONS

In this section, conclusions are presented considering the aim of this master thesis, which was to evaluate the integration of a linear guide under the base of a collaborative robot for a pHRC.

The described analytical methods for the inverse kinematics, the Pseudo-inverse, weighted Pseudo-inverse, and DLS inverse, along with the projected gradient contribution, are all Jacobian-based methods that minimize the joint velocities norm. Consequently, they are suited for an HRI. However, as demonstrated in the previous chapter, some issues can arise during trajectory execution. For example, self-collisions can still occur even while the projected gradient aiming at minimizing joint limits index is active. A possible solution could be to dynamically adapt the gain $\boldsymbol{k}_0$ based on the proximity of a joint limit. If a joint gets too close to one of its limits, its gain progressively rises to avoid the saturation. Furthermore, joint limits defined in (2.31) and (2.32) are not always suitable, particularly for q_5, because in some configurations q_5 can be higher without causing a self-collision. Nevertheless, this depends on the value of q_4. Thus, joint limits should dynamically change too, becoming dynamic inputs for the projected gradient method. Concerning the singularity, taking into account all the considerations introduced for the DLS method, the best value for μ was proven to be equal to 0.05, which represents the best compromise between large joint velocity occurring in the proximity of a singularity and task accuracy. This solution could be adopted, along with a proper position controller gain such as $\boldsymbol{K}_{P,i} = \boldsymbol{K}_{0,i} = 1.5$, only when a singularity is detected. In this way, it is possible to adopt the Pseudo-inverse solution when the configuration is far from the singularity, without accumulating task error, whereas the DLS solution can be activated in proximity of a singularity.

The gains $\boldsymbol{K}_P$ and $\boldsymbol{K}_0$ might be changed based on the size of the pose error to avoid excessive accelerations. High accelerations can also be requested if fast velocities changing are needed during motion. Therefore, imposing four seconds for each segment could not be appropriate in some scenarios.

Indeed, even if in a first-order algorithm joint velocities have limits, this is not guaranteed for joint accelerations, which may produce fast robot changes in motion unpredictable for human operators. Therefore, it is important to focus on the dynamic aspects too, by also modeling them in the algorithm, for safer and more accurate robot behavior.

All these considerations are made to find a better solution for a time-varying trajectory, that is requested for a real pHRC. Indeed, during the trajectory shown in the 3.3 section as an example, if an operator moves along the table while being tracked by a camera or a sensor, the robot should adapt by modifying the desired position of the external waypoints. Consequently, online motion planning is requested, without the need of a human intervention. For this reason, the algorithm parameters should dynamically and automatically adapt.

On the contrary, internal waypoints have been introduced only to constrain the motion, and they have been considered fixed. However, it was demonstrated how the desired internal orientations can be crucial both for singularity and self-collision problems. Nevertheless, if orientations are not specified, the EE angular velocity is not imposed. Therefore, the order of the redundancy increases up to four, and more complex solutions are needed, such as the *Task augmentation* method.

Finally, to check if the implementation of the kinematic controller is correct and the assumption that a first-order dynamic model is sufficient to represent robot dynamics, experimental tests are required. Accordingly, dynamic effects can be effectively analyzed by comparing the actual robot pose to the desired one.

REFERENCES

[1] E. Galati. "Cobot" lecture from *Sistemi integrati di produzione*, a.a. 2022-2023, Politecnico di Torino,Turin.

[2] C.Tartara (2023), “Braccio robotico collaborativo ridondante con settimo asse lineare.”(Tesi di laurea magistrale, Politecnico di Torino)

[3] M. H. Zafar, E. F. Langås, and F. Sanfilippo, “Exploring the synergies between collaborative robotics, digital twins, augmentation, and industry 5.0 for smart manufacturing: A state-of-the-art review,” Oct. 01, 2024, Elsevier Ltd. doi: 10.1016/j.rcim.2024.102769.

[4] M. C. Zizic, M. Mladineo, N. Gjeldum, and L. Celent, “From Industry 4.0 towards Industry 5.0: A Review and Analysis of Paradigm Shift for the People, Organization and Technology,” Jul. 01, 2022, MDPI. doi: 10.3390/en15145221.

[5] N. Berx, W. Decré, J. De Schutter, and L. Pintelon, “A harmonious synergy between robotic performance and well-being in human-robot collaboration: A vision and key recommendations,” Annu Rev Control, vol. 59, p. 100984, 2025, doi: 10.1016/j.arcontrol.2024.100984.

[6] https://www.rollon.com

[7] A. Silwal, J. R. Davidson, M. Karkee, C. Mo, Q. Zhang, and K. Lewis, “Design, integration, and field evaluation of a robotic apple harvester,” J Field Robot, vol. 34, no. 6, pp. 1140–1159, Sep. 2017, doi: 10.1002/rob.21715.

[8] A. Sridhar Reddy, V. V. M. J. Satish Chembuly, and V. V. S. Kesava Rao, “Collision free Inverse Kinematics of Redundant Manipulator for Agricultural Applications through Optimization Techniques,” International Journal of Engineering, Transactions A: Basics, vol. 35, no. 7, pp. 1343–1354, Jul. 2022, doi: 10.5829/ije.2022.35.07a.13.

[9] Y. Tong, J. Liu, Z. Ju, Y. Liu, and L. Fang, “Dynamic precision analysis of a redundant sliding manipulator,” Robotics and Rehabilitation intelligence, 2020.

[10] Y. Tong, J. Liu, Y. Liu, and Y. Yuan, “Analytical inverse kinematic computation for 7-DOF redundant sliding manipulators,” Mech Mach Theory, vol. 155, Jan. 2021, doi: 10.1016/j.mechmachtheory.2020.104006.

[11] SLIDEKIT 2.0, Universal Robots: https://www.universal-robots.com/marketplace/products/01tP40000071NNMIA2/

[12] Movotrak Cobot transfert unit, Universal Robots: https://www.universal-robots.com/marketplace/products/01tP40000071NiUIAU/

[13] B. Siciliano *et al.*, “Robotics Modelling, Planning and Control”. Springer, 2000.

[14] A. De luca., "Kinematic control" and “Kinematic redundancy part 1” lectures from *Robotics 1* and *Robotics 2,* a.a. 2024-2025, Università degli Studi di Roma “La Sapienza”, Rome.

[15] Wall, Michael E., Andreas Rechtsteiner, and Luis M. Rocha. "Singular value decomposition and principal component analysis." A practical approach to microarray data analysis. Boston, MA: Springer US, 2003. 91-109.

[16] Simulink file "manipTransformTrajectoryTimeScaling.slx", from *Trajectory Planning for Robot Manipulators*, MathWorks. https://it.mathworks.com/videos/trajectory-planning-for-robot-manipulators-1556705635398.html

[17] A. S. Deo and I. D. Walker, “Overview of Damped Least-Squares Methods for Inverse Kinematics of Robot Manipulators,” 1995.