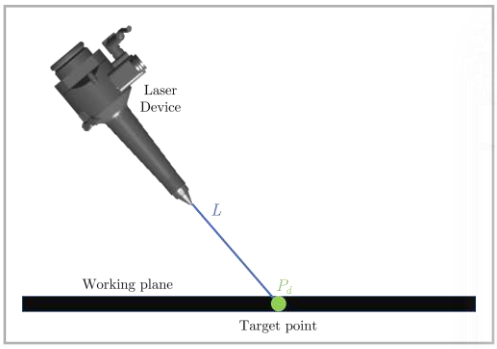
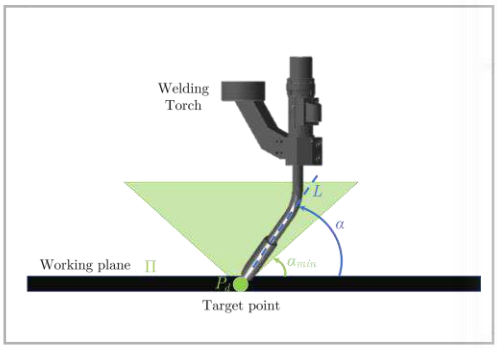
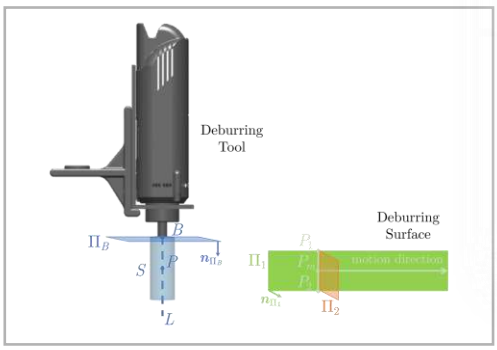
Mario Daniele Fiore

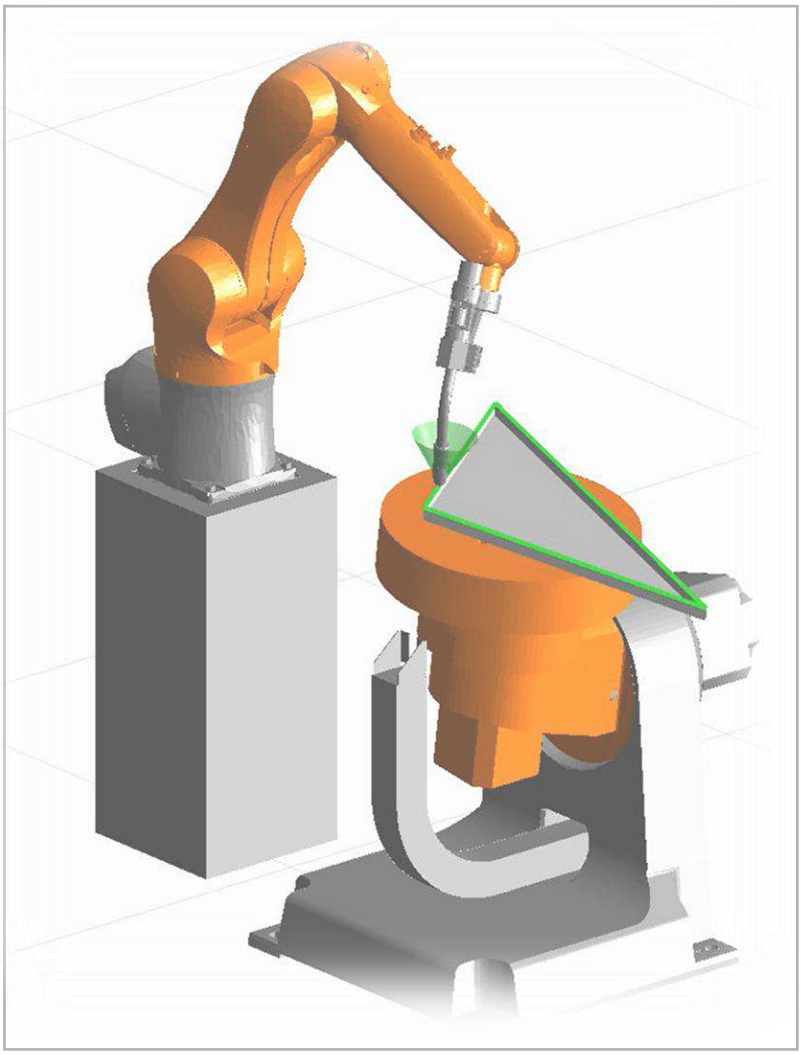
Intuitive Programming of Redundant Robots Leveraging Constraint-Based Techniques

Execution of industrially relevant tasks









Università
degli Studi
della Campania
Luigi Vanvitelli

KUKA

Università
degli Studi
della Campania
Luigi Vanvitelli

Dipartimento di
Ingegneria

Dottorato di Ricerca in
Ingegneria Industriale e dell'Informazione
XXXV Ciclo

Ph.D. Thesis

Intuitive Programming of Redundant Robots Leveraging Constraint-Based Techniques

Execution of industrially relevant tasks

S.S.D. ING-INF/04-Automatica

A.A. 2021/2022

Author	Coordinator	Advisor	Co-Advisor
Mario Daniele Fiore Matr. D129/68	*Prof.* *Oronzio Manca*	*Prof.* *Ciro Natale*	*Dr. Felix* *Allmendinger*

This dissertation has been approved by:

Prof. Ciro Natale *Advisor*	Università degli Studi della Campania "Luigi Vanvitelli"
Dr.-Ing. Felix Allmendinger *Co-advisor*	KUKA Deutschland GmbH
Prof. Gianluca Antonelli *Reviewer*	Università degli studi di Cassino e del Lazio Meridionale
Prof. Fabrizio Caccavale *Reviewer*	Università degli Studi della Basilicata

Graduation Committee:

Advisor:

Prof. Ciro Natale	Università degli Studi della Campania "Luigi Vanvitelli"

Co-advisor:

Dr.-Ing. Felix Allmendinger	KUKA Deutschland GmbH

Committee members:

Prof. Salvatore Pirozzi	Università degli Studi della Campania "Luigi Vanvitelli"
Prof. Fabrizio Caccavale	Università degli Studi della Basilicata
Prof. Adriana Brancaccio	Università degli Studi della Campania "Luigi Vanvitelli"
Prof. Angelo Liseno	Università degli Studi di Napoli "Federico II"
Prof. Francesco Soldovieri	Istituto per il Rilevamento Elettromagnetico dell'Ambiente - CNR Napoli

Date of the final exam: May 15th, 2023

The work presented in this thesis has been conducted at the Engineering Department of Università degli Studi della Campania "Luigi Vanvitelli", and at the Technology & Innovation Center of KUKA Deutschland GmbH.

The work has been developed within the Internet of Construction project, that has received funding from the German Federal Ministry of Education and Research (BMBF) (grant agreement No 02P17D083 - Internet of Construction).

Intuitive programming of redundant robots leveraging constraint-based techniques
by Mario Daniele Fiore

Abstract

This thesis focuses on the programming of redundant robots in industrial contexts, with the goal of developing methods and technologies to make robot programming more intuitive and accessible to untrained users. Industrial robots can be indeed very hard to program, even for experts. Classic task programming methods based on the specification of Cartesian frames fail at offering an intuitive mapping between the task objectives and the robot commands. Furthermore, they often end up generating overconstrained motions, disregarding the possible redundant degrees of freedom, and reducing the probability of success in finding a feasible robot motion.

The work in this thesis starts by defining an *intuitive* task description, which is formulated leveraging common understanding of spatial relations between objects. This way, task requirements can be easily defined and understood by users with very limited expertise in robotics. In fact, the obtained description is independent from the robot(s) appointed to execute the task. Moreover, the considered methodology allows to constrain only a minimum number of degrees of freedom. Robots are then automatically regarded as redundant whenever possible. A novel constraint-based programming framework is proposed, which allows to mathematically express intuitive task descriptions according to a provided formalism. Furthermore, the framework is able to relate the task description to the mathematical model of the robot(s) involved, and automatically generate motion control problems. To solve such problems, this thesis introduces a general method for hierarchical redundancy resolution under arbitrary constraints, in which both kinematic (velocity or acceleration-based) and dynamic (torque-based) control of redundant robots are handled in a unified fashion.

Finally, the thesis presents a software component which implements all the concepts presented in this work, therefore realizing a productive tool for the intuitive specification and the execution of industrial tasks on redundant robots.

Sommario

Questa tesi tratta la programmazione di robot ridondanti in ambito industriale, con l'obiettivo di sviluppare metodi e tecnologie per rendere la programmazione dei robot più intuitiva ed accessibile a utenti non esperti. La programmazione di un task industriale può risultare infatti un'operazione lunga e tediosa. I classici metodi di programmazione, basati sulla specifica di terne cartesiane, non riescono a catturare in maniera intuitiva gli obiettivi del task assegnato e generano spesso moti sovravincolati, trascurando i possibili gradi di libertà ridondanti offerti dal manipolatore.

Il punto di partenza di questo lavoro è la definizione di un modello di descrizione *intuitiva* del task, basato sulla specifica di semplici relazioni spaziali tra oggetti. La descrizione ottenuta è completamente indipendente dai robot incaricati di eseguire il compito assegnato. In questo modo, gli obiettivi del task possono essere facilmente definiti e compresi da utenti con conoscenze limitate di robotica. Inoltre, la metodologia introdotta consente di vincolare un numero minimo di gradi di libertà del manipolatore. I robot sono quindi automaticamente considerati ridondanti, quando possibile. Attraverso un nuovo framework di constraint-based programming, è possibile poi esprimere matematicamente, secondo un preciso formalismo, le descrizioni intuitive introdotte. Il framework proposto è in grado di mettere in relazione la descrizione del task con il modello matematico dei robot coinvolti e di generare automaticamente problemi di controllo del moto. Per risolvere tali problemi, questa tesi introduce un metodo generale per la gestione della ridondanza, che consente sia il controllo cinematico che dinamico di robot.

Infine, la tesi presenta un componente software che implementa tutti i concetti presentati in questo lavoro, realizzando così uno strumento produttivo ed efficace per la specifica e l'esecuzione intuitiva di task industriali su robot ridondanti.

Acknowledgments

It is hard to express in a few words my gratitude to the people that made this journey possible.

First and foremost I am extremely grateful to Dr. Rainer Bischoff, Dr. Uwe Zimmermann, Prof. Ciro Natale, and Dr. Felix Allmendinger for giving me the opportunity of pursuing a PhD while continuing working at KUKA. Towards Prof. Ciro Natale I'm also thankful for the continuous feedback and the precious support. His dedication and passion for robotics are the reason I first stepped into this field, and I feel honored to have had him at my side once again. I would also like to express my deepest appreciation to Dr. Felix Allmendinger. Thank you, Felix, for pushing me towards new research directions, and for all the valuable discussions in the last three years.

During my time at KUKA I have been fortunate to collaborate with many talented colleagues and students. I would like to express my sincere gratitude to Martin Feustel, Dr. Juan David Munoz Osorio, and Guglielmo van der Meer, with whom I shared the development of the KUKA Smart Motion Generator. I would also like to offer my special thanks to Dr. Kirill Safronov for his continuous support, and to Anton Ziese and Gaetano Meli for their enthusiastic contribution.

I could not have undertaken this journey without the lovely support of my family, who never missed to encourage me in the past three years. Special thanks should also go to my beautiful friends, for their unconditional love and support despite the distance. Finally, there is a person I have met during the PhD studies that I need to thank for believing in me, and staying at my side in my best and my worst moments. I feel lucky to have her in my life. Thank you, my love.

Contents

Notation

The symbols in this thesis are chosen according to the following conventions:

- Scalar values or signals are denoted by italic letters such as x;
- Vectors are denoted by boldface lowercase letters such as $\boldsymbol{x}$;
- Matrices are denoted by boldface uppercase letters such as $\boldsymbol{A}$;
- Sets are denoted by calligraphic letters such as $\mathcal{Z}$.

The following acronyms have a special meaning:

CLIK	Closed-Loop Inverse Kinematics
DoF	Degree of Freedom
eSNS	extended Saturation in the Null Space
IoC	Internet of Construction
KSMG	KUKA Smart Motion Generator
PoC	Point of Control
QP	Quadratic Programming
SNS	Saturation in the Null Space
SoT	Stack of Tasks
VKC	Virtual Kinematic Chain

Chapter 1

General Introduction

1.1 Motivation

Since their introduction in the early 70s in shop floors and production lines, industrial robots have been massively employed to execute tasks with high precision, speed, and endurance. Over the years, they have replaced humans in many tedious and dangerous works. Welding, painting, deburring, laser-cutting, palletizing are just some examples of typical industrial tasks that are nowadays fully automated in medium to large companies in the manufacturing sector.

Industrial robots can be used as incredibly effective tools and yet they can be very hard to program. The programming stage can easily take hundreds of hours even for expert programmers, especially when involving complex movements or the coordination of multiple robots. The required effort and costs are often considered tolerable since, once programmed, every robot is very likely to repetitively perform the same motion for a very long time, possibly its entire service life. However, this dramatically reduces the flexibility of these machines and, thus, their potential.

Modern production lines demands for versatility and high flexibility in reprogramming robotic systems, due to frequent changes in the environment or switches of production. Flexibility and ease of use are also crucial aspects for small and medium enterprises. In this sector, the lack of trained robot programmers, the frequent changes in production, and the typically low volumes of required robots cause a large number of tedious operations still to be performed manually by humans. Thus, a huge potential for automation remains unexploited. A similar scenario is observed in the service robotics field. Here, robots are typically required to perform useful tasks for humans. Clear and intuitive programming interfaces are therefore required, and reprogramming should happen with very limited effort. At the current state, all the aforementioned demands remain mostly unmet.

Classic programming methods for industrial robots are based on the specification of Cartesian frames. The robot is required to align a body-fixed frame, usually attached to the robot end effector, to a world-fixed frame, e.g., placed in the free space or on the workpiece. The world-fixed frame can additionally be specified to move on a desired Cartesian trajectory, e.g., to follow a linear path with a given velocity profile. Motion can also be directly commanded to each joint of the robot to move towards the desired

frame according to some interpolation law. In any case, generating robot motion requires solving the inverse kinematics problem for each desired Cartesian frame.

From the methodology described above, it is immediate to realize that, besides having a good knowledge of the process, programming an industrial robot requires a certain training. First, it is not immediate to think of task requirements in terms of Cartesian frames. Thus, a certain effort is typically required to convert the actual task objectives into a sequence of desired frames. Furthermore, there are several reasons why the motion planning could fail. These include running into joint limits or collisions with the environment. A certain level of experience is therefore required when placing Cartesian frames, to prevent the robot to move towards dangerous or unfeasible configurations. Finally, as the vast majority of industrial robots is characterized by a 6-axis serial kinematic chain, the specification of Cartesian frames only leaves a finite and very limited number of alternative postures available when executing a given task. However, a considerably large number of industrial applications does not require constraining all the six degrees of freedom of the robot tool. In such a condition, the robot mechanism could then be regarded as redundant. Thus, classic robot programming tends to overconstrain the mechanical system, significantly limiting the ability of the robot to accomplish the assigned task.

1.2 Research Questions

From the considerations in the previous section, it clearly appears that the way industrial robots are currently programmed is not suitable in many relevant contexts. From a conceptual point of view, making programming of industrial robots more accessible to untrained users would require, first of all, more intuitive interfaces. According to this idea, programming a robot should focus more on how a task is described and accomplished, rather than specifying every single motion the robot should perform. A proper task description should focus on the actual task requirements, and avoid overconstraining the mechanical system. This leads to the first research question of this thesis:

Q1: What is an intuitive task description?

This question is mainly addressed in Chapter 2, where a task is formulated as an operation enforcing specific spatial relations (geometric constraints) among a certain set of involved objects. The proposed formulation is independent from the robot(s) responsible to perform the task. This simplifies the definition and the understanding of task requirements to users with limited robotics expertise. To support the presented concept, three case studies are introduced, which are then analyzed and considered throughout the thesis.

Once a task is described, there is still the problem of computing the robot motion that actually fulfills the given requirements. Therefore, the second research question, directly following from the first one, can be formulated as follows:

Q2: How can the robot motion be automatically generated from the intuitive task description?

This question is mainly addressed in Chapter 3, where a constraint-based mathematical formalism is introduced. Such formalism naturally supports the intuitive task description from Chapter 2. At the same time, it allows through a systematic procedure to build motion control problems. The method naturally scales to collaborative multi-robot applications, which are handled in a unified fashion. Since the intuitive task description aims at constraining only a minimum number of degrees of freedom, the generated motion control problems generally involves redundancy resolution. This typically requires solving constrained optimization problems. However, the different interfaces of robot joint controllers (e.g., position, velocity or torque), together with the various objectives one might wish to achieve, create a certain heterogeneity in the formulation and the solution of such problems. This leads to the third research question:

Q3: What is a general and efficient method to solve redundancy resolution problems?

This question is addressed in Chapter 4, where a general framework for the formulation of redundancy resolution problems is proposed at both kinematic and dynamic level.

Finally, the thesis additionally addresses the question of how a software component for intuitive programming of redundant robots could be structured. To this end, general design guidelines are given in Chapter 2, while Chapter 5 provides the implementation details of a software component developed during the course of the doctorate study.

1.3 Outline

This section briefly describes the outline of this thesis (Fig. 1.1).

Chapter 2: Task Specification

This chapter recalls the state of the art in the field of industrial robot programming, and defines the concept of intuitive task description. Furthermore, it introduces the three case studies considered throughout the thesis. Finally, it outlines the layers of the proposed architecture for intuitive robot programming.

The chapter extends the core ideas presented in the following paper:

M. D. Fiore, Felix Allmendinger, Ciro Natale. "A General Constraint-Based Programming Framework for Multi-Robot Applications", In: *Robotics and Computer Integrated Manufacturing* (currently under review).

Chapter 3: Constraint-Based Programming

This chapter formulates a novel constraint-based programming framework allowing intuitive task descriptions to be mathematically expressed according to a suitable formalism, which can handle single and multi-robot applications in a unified fashion. It also illustrates a systematic procedure to automatically derive motion control optimization problems, which can be solved to generate robot motion.

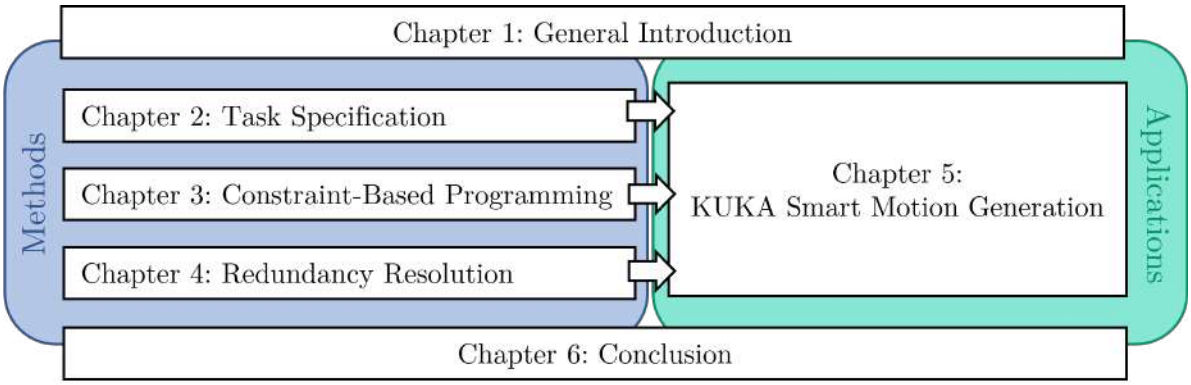


Figure 1.1: Thesis outline.

The chapter is based on and extends the following publications:

M. D. Fiore, Felix Allmendinger, Ciro Natale. "A General Constraint-Based Programming Framework for Multi-Robot Applications", In: *Robotics and Computer Integrated Manufacturing* (currently under review)

J. Lachner, V. Schettino, F. Allmendinger, M. D. Fiore, F. Ficuciello, B. Siciliano, and S. Stramigioli (2020). "The influence of coordinates in robotic manipulability analysis". In: *Mechanism and machine theory* 146, p. 103722.

Chapter 4: Redundancy Resolution

This chapter focuses on general methods to solve the constrained optimization problems originating from Chapter 3. In particular, it proposes a novel general framework for hierarchical redundancy resolution under arbitrary constraints. Moreover, it addresses some critical aspects of discrete-time implementations of the redundancy resolution solver.

The chapter is based on and extends the following publications:

V. Schettino, M. D. Fiore, C. Pecorella, F. Ficuciello, F. Allmendinger, J. Lachner, S. Stramigioli, and B. Siciliano (2020). "Geometrical Interpretation and Detection of Multiple Task Conflicts using a Coordinate Invariant Index". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6613–6618

A. Ziese, M. D. Fiore, J. Peters, U. E. Zimmermann, and J. Adamy (2020). "Redundancy resolution under hard joint constraints: a generalized approach to rank updates". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7447–7453

M. D. Fiore and C. Natale (2021). "Discrete-time closed-loop inverse kinematics: A comparison between Euler and RK4 methods". In: *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 584–589

M. D. Fiore, G. Meli, A. Ziese, B. Siciliano, and C. Natale (2023). "A General Framework for Hierarchical Redundancy Resolution Under Arbitrary Constraints". In: *IEEE Transactions on Robotics*, pp. 1–20

Chapter 5: KUKA Smart Motion Generator

This chapter presents a software component, named KUKA Smart Motion Generator, developed during the course of the doctorate study. The component embeds the concepts and the methodologies developed throughout the thesis, implementing a complete software framework for intuitive task programming, from symbolic task specification to automatic robot motion generation.

Chapter 6: Conclusion

This chapter discusses the results of this work and gives directions for possible future extensions.

Chapter 2

Task Specification

2.1 State Of The Art In Industry

Industrial robots have been developed for decades thinking of them as extremely productive machines operating in fixed cells, moving their end effector to carry out some planned tasks. Over the years, significant progress has been made on the mechatronic side to deploy high-payload and, more recently, low-cost and lightweight robots. With the spread of cobots, these machines are finally required to overstep working cells and work side-by-side with humans. In this context, easy programming interfaces have also become crucial to the overall user experience. In the last years, robot manufactures, as well as independent software companies, have invested considerable efforts in the development of graphic frameworks for application programming, enabling users with no expertise in robot programming languages to program robotic applications. These frameworks typically present a set of parameterized function blocks that simplify the programming of new applications or the changing of existing ones. Moreover, an integrated scene viewer is generally provided to visualize features of interest and preview the outcome of the programmed motion through simulation (Fig. 2.1).

In spite of the improved user experience, however, not much has changed in the way the actual robot motion is specified. Indeed, when it comes to motion programming, any robot interface substantially offers two possible commands (Fig. 2.2):

- **joint motion commands**, mainly used for approach/departure motions before/after task execution: every joint of the robot is commanded to a specific value or to follow a specific joint space trajectory;

- **Cartesian motion commands**, mainly used for task execution: a frame of interest on the robot end effector is commanded to a desired pose or to follow a Cartesian trajectory; the actual joint motion is then obtained via inverse kinematics techniques.

Task programming via Cartesian frames imposes the control of the 6 Degrees of Freedom (DoFs) of the end-effector body. However, many industrial tasks present tool symmetry, workpiece geometry or final objectives that allow partial or complete relaxation of some

Figure 2.1: Modern programming frameworks for industrial robots (Source: KUKA, https://www.kuka.com/en-de/future-production/iiqka-robots-for-the-people/robotic-republic).

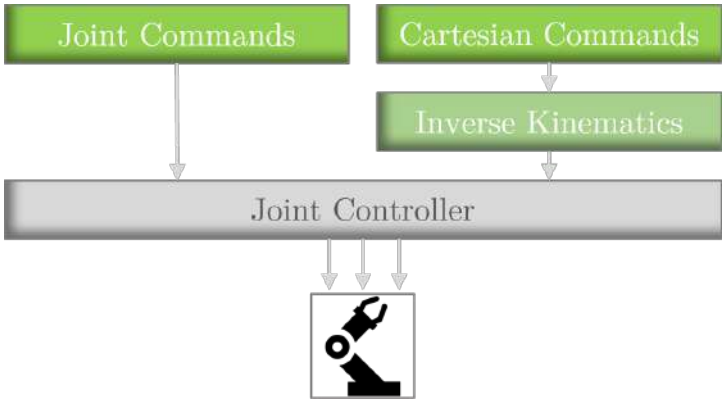


Figure 2.2: Possible motion commands in common user interfaces for industrial robots.

of the DoFs. Thus, even common 6-axis serial manipulators can be regarded as redundant for a variety of industrial applications.

Figure 2.3 shows the redundant DoFs offered by common industrial tools for the execution of tasks such as laser tracing, welding, and deburring. In the first case, the task can be performed with a variety of possible tool poses, as long as the laser beam is pointing to target point (highlighted in green). Depending on the task requirements, the distance of the tool from the target point may also be allowed to vary. The welding application allows a relaxation of the tool orientation, as the major task requirement is that the tip of the welding torch is at the target point. Finally, the deburring application prescribes for the drill bit (represented by the blue cylinder) to slide over a target surface (green rectangle). Also this operation allows for a certain relaxation of the orientation (the rotation about the blue axis of the cylinder frame can be disregarded) and some translational freedom in the positioning of the tool along the vertical direction.

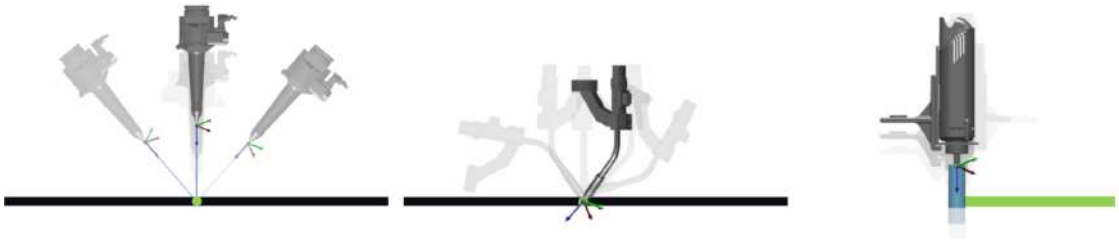


Figure 2.3: Redundant degrees of freedom offered by some typical industrial applications: laser tracing (left), welding (center), and deburring (right).

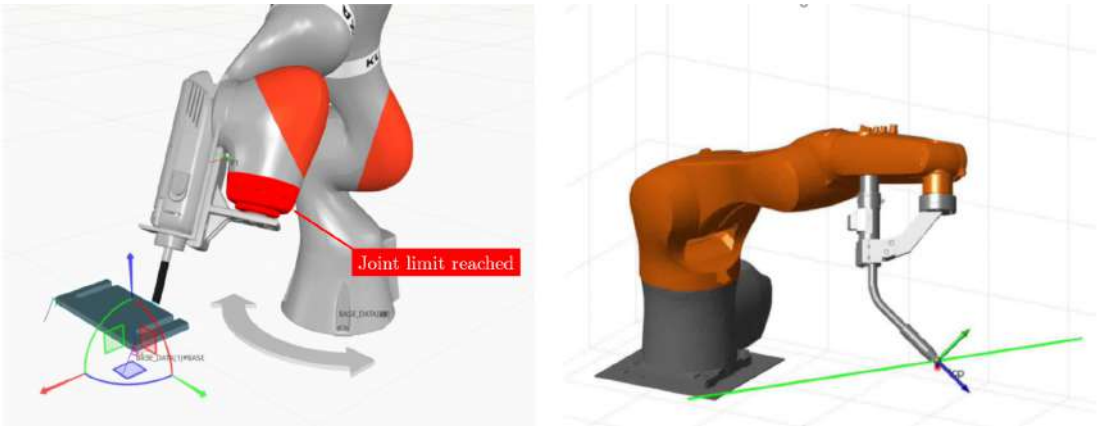


Figure 2.4: Failures during motion planning due to the robot reaching joint limits (left) or outstretched configurations (right).

Based on the above considerations, it is easy to realize that programming industrial applications based on Cartesian frames means in the majority of cases fixing the tool to just one of the infinite possible poses that would accomplish the task, therefore overconstraining the robotic system. This dramatically reduces the robot motion capabilities and, consequently, the probability of finding feasible motions for the given task. Figure 2.4 show some typical failures in which the assigned Cartesian trajectory leads the robot into unfeasible configurations, i.e., beyond joint limits, or out of its reachable workspace. Another reason for failure during motion planning is that the desired Cartesian trajectory corresponds to robot postures that cause collisions with the environment or the robot itself (self-collisions).

In addition to the already discussed aspects, it is also useful to consider that mapping the actual task objectives into desired Cartesian frames is not always an intuitive and straightforward process. In fact, since no task is naturally described in terms of frames, the required effort might become significant, and finally lead to over-engineered solutions. Often very long trial-and-error procedures are needed, until a feasible motion is found. The process can even last hundreds of hours, and become more tedious and cumbersome in the case of multi-robot applications. A typical industrial example in this field is the case in which a first mechanism, here indicated as the *positioner*, is responsible for positioning the workpiece in space, whereas a second robot, the *executor*, performs

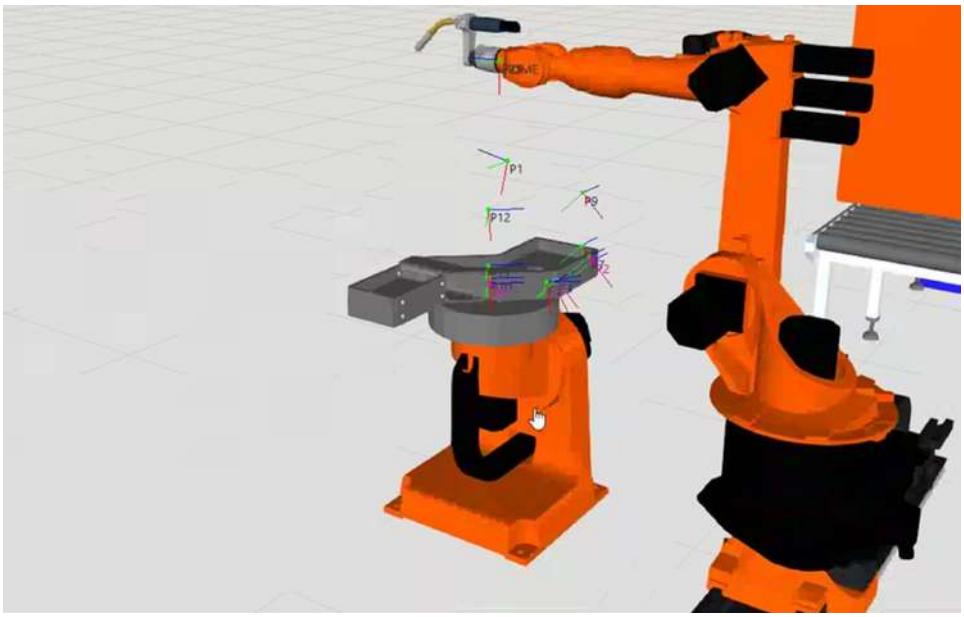


Figure 2.5: Multi-robot welding application programmed using Cartesian frames. A 2-DoFs pan-tilt table works as positioner while a 6-axis executor is responsible for the task execution.

the task. Commonly, the positioner is represented by a specialized unit with a small number of DoFs, such as a linear axis or a pan-tilt table (Fig. 2.5). Depending on the application, there might also be multiple executors for one positioner. Programming such applications typically requires the following sequence of steps:

1. a Cartesian trajectory is specified for the executor, while the positioner is at a reference configuration;

2. a desired (joint or Cartesian) trajectory is provided to the positioner, therefore generating a fixed motion for this unit;

3. the trajectory originally planned for the executor is adjusted by adding an offset deriving from the relative motion of the positioner.

Besides generating long and laborious programming sessions, the described procedure completely ignore the additional redundant DoFs offered by the positioner to the total robotic systems. Such redundancy could instead be efficiently exploited in the search for feasible motions.

All of the above mentioned considerations highlight how task descriptions based on Cartesian frames do not exploit possibly existing redundant DoFs, and produce long and tedious programming sessions. The resulting effort is often considered tolerable considering that, once programmed, every robot is very likely to repetitively perform the same motion for a considerably long time. However, this approach is not suitable for a variety of applications (better discussed in Sect. 1.1), which demand flexible production lines and solutions that can be easily re-programmed. In such a context, new programming

paradigms are needed, which allow for more intuitive task descriptions enabling non-experts to work with robots. Additionally, the task description should guarantee the exploitation of redundant DoFs.

In recent years, research has started to focus on object geometric features and relations, as the key to capture only the relevant DoFs that need to be constrained in the execution of a task. Geometric constraints have been found to be powerful tools in different contexts (Bartels, Kresse, et al. 2013; Kresse and Beetz 2012; Somani, Gaschler, et al. 2015), also to characterize the task requirements in a way that is suitable also for users with limited robotics knowledge. Following these principles, the next sections present a formal description of industrial tasks based on geometric constraints, structured on two levels of abstraction.

2.2 Task Description Based On Geometric Constraints

Reasoning about what is an intuitive task description for untrained robot users, the immediate question arises on how a human would commonly explain the requirements of a certain industrial process using natural language. Considering the tasks in Fig. 2.3, it is easy to imagine that the description for the laser tracing application would be something similar to "the laser beam must hit the target point". Similarly, the characterization of the welding task would probably be "the tip of the welding torch must be at the target point". In other words, the objectives of the considered industrial tasks are naturally described as spatial relations between two objects, i.e., the tool and the workpiece. The same applies for the deburring case, and for a variety of other industrial tasks (milling, polishing, laser cutting, ...). More specifically, the desired spatial relations consist of a set of geometric constraints that should be enforced between certain features of the objects involved. In the laser tracing application, for example, the laser beam can be regarded as a geometric feature of the tool, whereas the target point is a feature of the workpiece.

Building on these simple observations, the key idea behind this work is that an industrial task always consists in accomplishing certain spatial relations between objects. Such relations, and their further characterization through geometric constraints and features, also represent the most natural way of describing the task objectives. Therefore, a task description can be regarded as intuitive if it directly allows the specification of geometric constraints between object features. While translating, at any instant of time, the spatial relations between two objects into desired and controlled Cartesian frames is not always straightforward and likely to result in overconstrained task specifications, geometric constraints between features can be easily described through mathematical expressions. Furthermore, while task specification through Cartesian frames fully constrain the 6 DoFs of the relative pose between two objects, geometric constraints allow to capture the minimum number of DoFs to constrain in order to fulfill the task. Finally, the mathematical expressions describing the geometric constraints enable the formulation of motion control problems with automatic redundancy exploitation, as it will be shown in Chapter 3.

For the sake of completeness, it should be noticed that the interaction between two

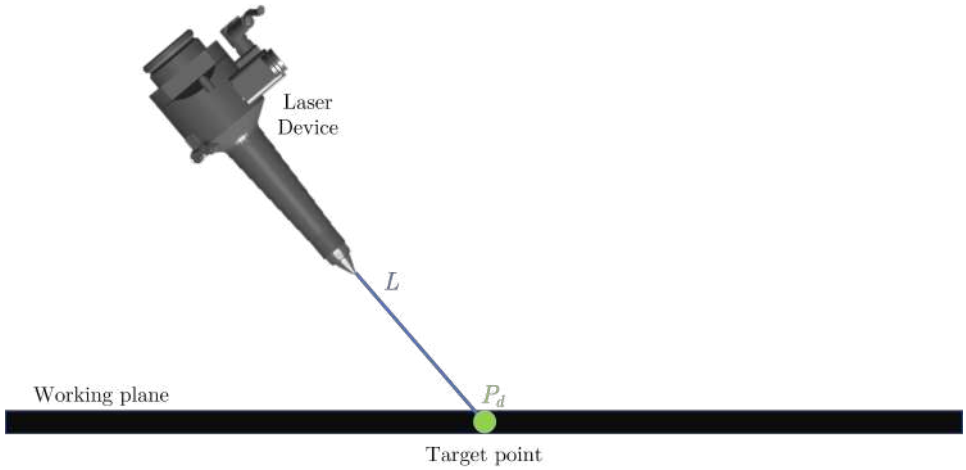


Figure 2.6: Laser tracing task specification.

objects, and therefore also their spatial relation, depends in general on the forces that they exchange. However, in the vast majority of industrial applications the force exchange can be disregarded, as the contact between the objects is absent, intentionally rigid or handled through mechanical impedance embedded in the robot tool.

The remainder of this section is dedicated to the analysis of the three case studies in Fig. 2.3, and the derivation of the geometric constraints describing their objectives. These three specific applications have been chosen for the generality of the resulting geometric constraints, which could be easily applied also to other industrial tasks.

2.2.1 Laser tracing

As already anticipated, the objective of the laser tracing application is for the laser beam to hit the target point. Thus, a geometric constraint is easily obtained imposing that the line L on which the laser beam lies on passes through the desired point $P_d(x_d, y_d, z_d)$, as shown in Fig. 2.6. This constraint can be symbolically expressed as

$$\text{Coincident}(L, P_d). \tag{2.1}$$

Recalling that a line is fully specified by an origin point, $P(x, y, z)$, and a unit vector indicating its direction, $\boldsymbol{d}$, (2.1) can be mathematically expressed by the 2-DoFs equality constraint

$$\text{null}(\boldsymbol{d}^T) \cdot (P - P_d) = \mathbf{0}, \tag{2.2}$$

where the operator null$(\cdot)$ returns an orthonormal basis for the nullspace of the matrix or row-vector it is applied to. Depending on the specific application, additional constraints might also be considered. For example, the angle between the laser beam and the working plane is only allowed to vary in a small range in laser welding/cutting applications. Also the distance between the point of emission and the target point may be subject to constraints given by the laser beam focusing. However, as similar requirements are included in the following case studies, the laser tracing application will only involve the

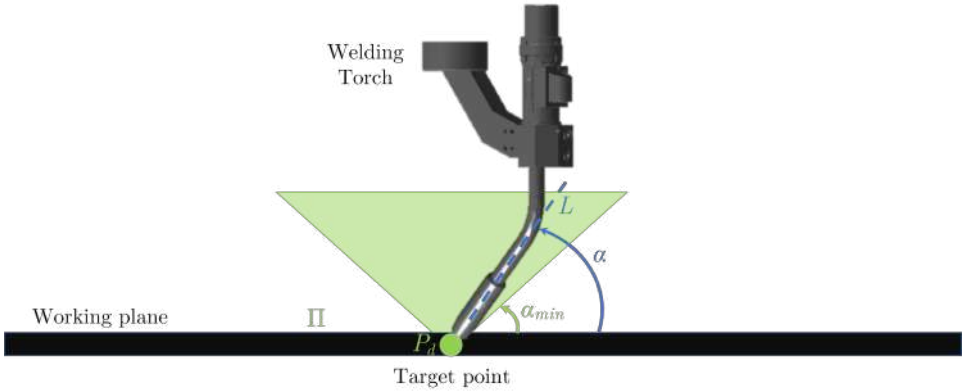


Figure 2.7: Welding task specification.

constraint (2.2) in the remainder of this work. This will allow to present a set of case studies with an increasing level of complexity. Finally, it should be noticed that the target point could also be required to vary its position, e.g., in tracking applications. Nevertheless, the proposed task description would still be valid at each instant of time, considering a correctly updated target point.

2.2.2 Welding

The main requirement of the welding application is for the tip of the welding torch, indicated by the point P, to be at a desired point P_d. Additionally, it is typically required that the task is performed with a certain inclination of the tool. In other words, the angle between the tool main axis, L, and the working plane, Π, should be maintained in a given range (Fig. 2.7). This last requirement is derived from process quality specifications, but it also helps preventing undesired collisions between the tool and the environment.

The described constraints can be symbolically written as

$$\begin{matrix} \text{Coincident}(P, P_d) \\ \text{Angle}(L, \Pi) > \alpha_{min} \end{matrix}, \tag{2.3}$$

with $\alpha_{min} \in [0, \frac{\pi}{2}]$ indicating a minimum desired angle between L and Π. Mathematically, (2.3) can be expressed by the following set of constraints

$$\begin{cases} P - P_d = 0 \\ \alpha - \alpha_{min} \geq 0 \end{cases}, \tag{2.4}$$

where $\alpha \in [0, \frac{\pi}{2}]$ represents the angle between L and Π. Indicating with $\boldsymbol{d}_L$ and $\boldsymbol{n}_\Pi$ the unit direction vector of L and the unit normal of Π, respectively, it is possible to compute $\alpha = \text{asin}(\boldsymbol{d}_L \cdot \boldsymbol{n}_\Pi)$. It should be noticed that (2.4) only constrains 4 DoFs, leaving two redundant DoFs to the process execution. Nevertheless, only 3 DoFs are fixed while

Figure 2.8: Metal workpiece considered for the deburring application (left), cylindrical drill bit (center), and determination of the deburring surface from process parameters (right).

a certain freedom is left on the value of α. Finally, as in the previous case study, the proposed task description can be employed as instantaneous description of the welding task in case the target point needs to vary over time.

2.2.3 Deburring

The objective of deburring is to remove burrs from the edges of metal workpieces. This operation is performed by allowing the rotating drill bit of the deburring tool to slide over time on a surface of interest, here referred to as *deburring surface*. This surface is a virtual area, generally internal to the workpiece, corresponding to the processed edge once the deburring operation is completed. As such, it can be determined out of process parameters, such as the desired penetration (depth) and inclination (angle) of the deburring tool during operation. In particular, the deburring use case analyzed throughout this thesis considers a metal workpiece with straight edges and a cylindrical drill bit. Thus, the deburring surface will be characterized as a rectangle (Fig. 2.8).

Differently from the previously analyzed industrial tasks, the requirements of the deburring application intrinsically prescribe a motion of the tool over time. Nevertheless, a task description can be obtained as in the previous case studies, which provides an instantaneous specification of the geometric constraints characterizing the task. In particular, at each instant of time, the process requirements impose that the lateral surface S of the cylindrical drill bit must be tangent to the deburring surface at a very specific line segment, $\overline{P_1P_2}$, which is orthogonal to the direction of motion (Fig. 2.9). The description of the overall deburring operation can then be obtained simply by translating the segment $\overline{P_1P_2}$ over time on the deburring surface along the direction of motion. The described instantaneous constraints can be summarized as

$$\text{Tangent}(S, \Pi_1, \overline{P_1P_2}), \tag{2.5}$$

where Π_1 is the plane the deburring surface lies on. However, it is not straightforward to convert the constraint (2.5) into mathematical expressions. It is therefore necessary to further detail the tangency condition.

A primary condition for the surface S to be tangent with Π_1 at $\overline{P_1P_2}$ is that the cylinder axis L must be parallel to $\overline{P_1P_2}$, and a generic point P belonging to L must be

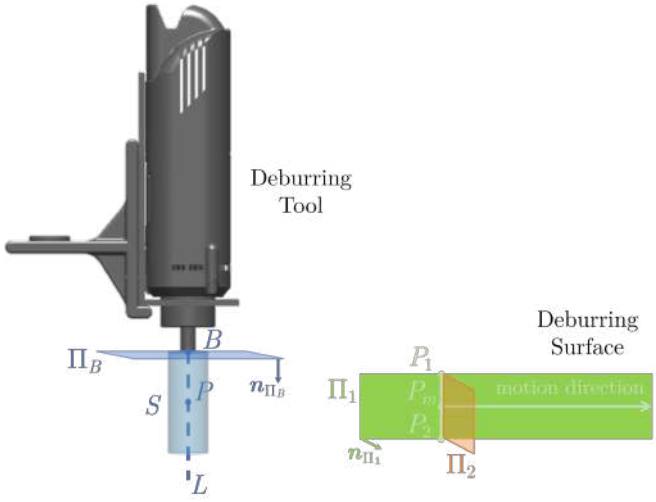


Figure 2.9: Deburring task specification.

at a distance r from Π_1, with r being the cylinder radius. More specifically, a signed distance function should be employed to ensure that the cylinder fulfills the tangency condition while being on the correct side of Π_1.

Imposing that the intersection between S and Π_1 is exactly the segment $\overline{P_1P_2}$ requires two additional conditions. First, L should be located on Π_2, defined as the plane embedding $\overline{P_1P_2}$ and orthogonal to Π_1. Then, the center of $\overline{P_1P_2}$, indicated as P_m, should maintain a certain distance from the plane embedding the cylinder base B. Such distance must be in the range $[\frac{l}{2}; h - \frac{l}{2}]$, with l being the length of $\overline{P_1P_2}$, and h being the cylinder height. More specifically, a signed distance function should be used in this case to ensure that the specified range realizes the desired tangency condition.

Given the above considerations, the symbolic constraint (2.5) can be refined and expressed as

$$\text{Parallel}(L, \overline{P_1P_2}) \tag{2.6a}$$
$$\text{SignedDistance}(P, \Pi_1) = r \tag{2.6b}$$
$$\text{Coincident}(P, \Pi_2) \tag{2.6c}$$
$$\frac{l}{2} \leq \text{SignedDistance}(P_m, \Pi_B) \leq h - \frac{l}{2}, \tag{2.6d}$$

where Π_B indicates the plane embedding the cylinder base B. Based on simple geometry analysis, it is now possible to find mathematical expressions for each of the constraints (2.6). Indeed, the parallel condition (2.6a) can be mathematically expressed by the 2-DoF constraint equation

$$\text{null}(\boldsymbol{d}_L^T) \cdot \boldsymbol{d}_{\overline{P_1P_2}} = \boldsymbol{0},$$

with $\boldsymbol{d}_L$ and $\boldsymbol{d}_{\overline{P_1P_2}}$ being the unit direction vectors of L and $\overline{P_1P_2}$, respectively. The signed distance relation (2.6a) is instead a 1-DoF constraint, which can be written as

$$\boldsymbol{n}_{\Pi_1} \cdot (P - P_{\Pi_1}) = r,$$

where $\boldsymbol{n}_{\Pi_1}$ and P_{Π_1} are the unit normal vector and a point of Π_1, respectively. Similarly, the coincident relation (2.6c) constrains one DoF, and can be written as

$$\boldsymbol{n}_{\Pi_2} \cdot (P - P_{\Pi_2}) = 0,$$

where $\boldsymbol{n}_{\Pi_2}$ and P_{Π_2} are the unit normal vector and a point of Π_2, respectively. Finally, the condition (2.6d) is a 1-DoF inequality constraint that can be expressed as

$$\frac{l}{2} \leq \boldsymbol{n}_{\Pi_B} \cdot (P_m - P_{\Pi_B}) \leq h - \frac{l}{2},$$

with $\boldsymbol{n}_{\Pi_B}$ and P_{Π_B} being the unit normal vector and a point of Π_B, respectively. Thus, the deburring task description constrains 5 DoFs of the tool, leaving one redundant DoF. More precisely, four of the constrained DoFs are fixed, while a certain freedom is allowed on the last one.

Lastly, it should be noticed that the obtained task description is not limited to rectangular deburring surfaces. In fact, even curved edges could be handled by defining Π_1 as the plane embedding the line segment $\overline{P_1P_2}$ and the vector of the motion direction. A different shape of the drill bit (e.g, conical or spherical) would instead require deeper modifications to the set of the geometric constraints.

2.3 Specification Layer

In the previous sections, the observation was made that, compared to established methods based on Cartesian frames, typical industrial tasks are more intuitively described by a set of geometric constraints. It has also been shown that formulating a symbolic task description grounded in geometric features and constraints allows a *minimal* task specification, meaning that the corresponding mathematical expressions are able to strictly constrain the minimum number of DoFs that accomplishes the task objectives. Furthermore, it has also been anticipated that such methodology enables the automatic exploitation of redundant DoFs during the generation of the robot motion. Finally, the design of geometric constraints has been practically shown for three relevant industrial applications.

While programming an application via Cartesian frames might require for the user to have both process and robot expertise to succeed, the proposed task description only focuses on the core constraints that characterize a task and is completely independent from the specific robot at hand. In fact, the description would not change whether the actual task execution involves the control of one or multiple robots. However, a certain level of process knowledge is still needed. Furthermore, the design of the minimal set of geometric constraints might require some experience for more complex tasks, such as the deburring application from Sect. 2.2.3. On the other hand, the geometric constraints describing a task basically depend on the geometry of the objects involved and some additional process parameters. In the considered case studies, for example, the task always involves two objects, i.e., a tool and a workpiece. Geometric constraints are always imposed between features of these two objects and depend on object properties (e.g,. the radius and height of the cylinder in the deburring application) or process

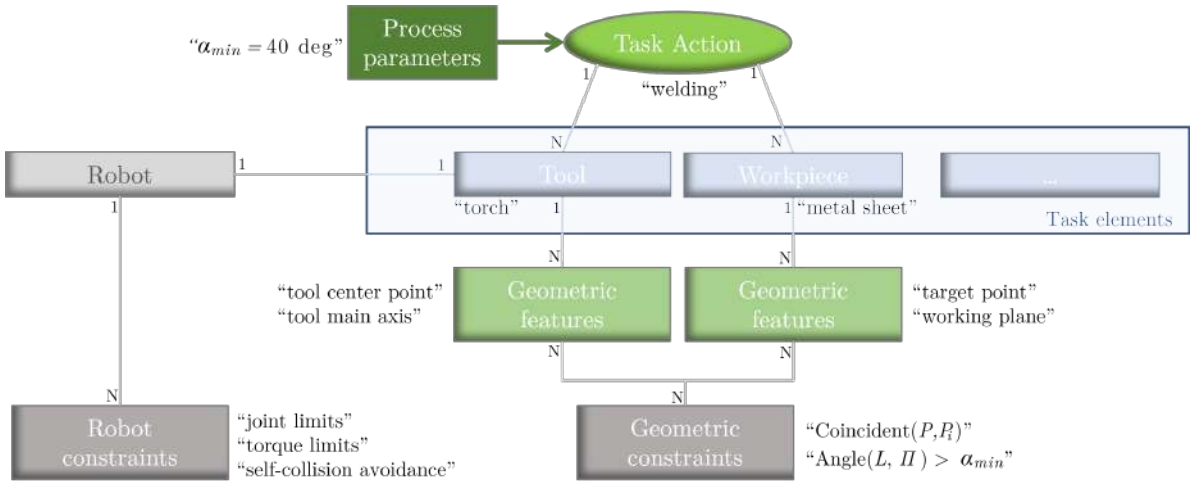


Figure 2.10: Task characterization in the specification layer.The welding application is used as example.

parameters (like the minimum angle of approach in the welding application). Thus, once the relevant geometries and parameters are known, the generation of the geometric constraints could also be automated for a known task. This enables the realization of a *specification layer* in which a task is characterized on a higher level of abstraction as an *action* involving a certain number of *elements* (Fig. 2.10). In this context, a task element represents a general entity presenting geometric features that are relevant to the task description. Thus, a task element could represent a physical object, like a tool or a workpiece, but also the generic robot environment in the case of trajectory-following applications in free space. The task action, on the other hand, symbolically indicates the operation to perform, and presents a set of parameters that characterize the task. Additionally, the indication of the task action allows to recover from the involved elements the geometric features that are relevant to the constraint-based task description. Thus, a symbolic list of geometric constraints can be correctly generated. Moreover, since every possible robot involved in the task will be connected to one of the task elements (typically, the tool), it might also be possible to retrieve the information about additional constraints imposed by the specific mechanism, such as joint limitations or self-collision avoidance. This information can be used during the motion generation phase to ensure the computation of feasible movements.

Certainly, the idea of a specification layer as illustrated above contemplates the existence of known tasks actions and elements, for which the design of the geometric constraints has already been carried out. Thus, it is possible to imagine a multi-layer structure for the definition and the execution of robot tasks that offers two different APIs. The first one, on the lower level, allows for the direct specification of symbolic geometric constraints. This interface is meant for users with a certain knowledge of the process and possibly some experience in designing minimal sets of geometric constraints. On a higher level, the second API allows for the specification (and parameterization) of tasks from a set of known resources, i.e., combinations of actions and elements for which the geometric constraints have been already designed. This interface is meant for users

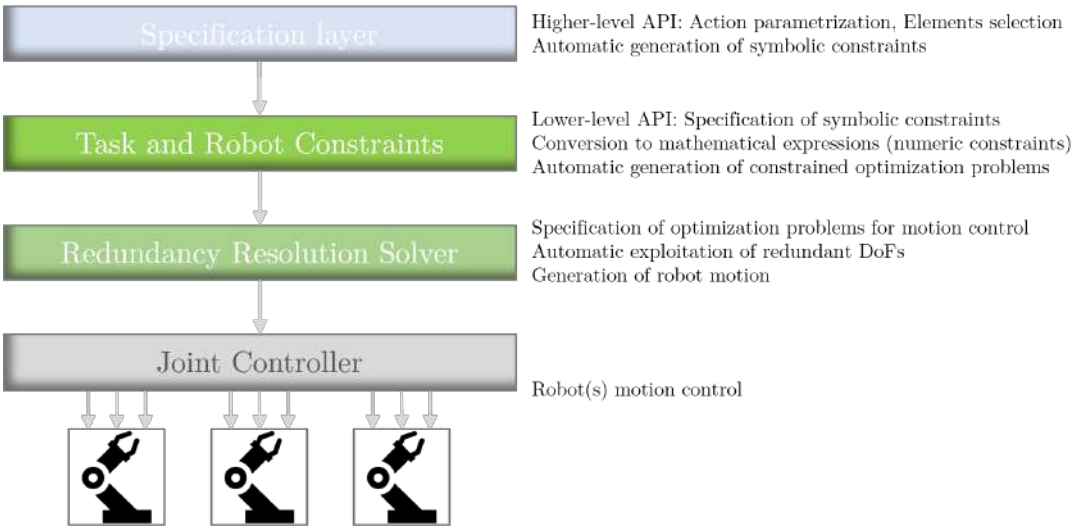


Figure 2.11: New stack for the intuitive programming of industrial tasks.

with limited process knowledge and no experience in designing geometric constraints. At this level, such users would simply select an action and the elements involved in the process, provide values for the parameters associated with the action, and benefit from the constraint-based task description without having to design any geometric constraint or understand robot redundancy. Figure 2.11 summarizes the proposed stack for task programming, which can be used as a replacement of the classic scheme illustrated in Fig. 2.2.

Chapter 3

Constraint-Based Programming

3.1 Introduction

The previous chapter discussed the advantages of constraint-based task descriptions over classic specification methods based on Cartesian frames. In particular, it was shown that tasks are intuitively described as sets of geometric constraints, which can be expressed through mathematical relations between certain features of the elements involved. It was also anticipated that such relations support the generation of (constrained) optimization problems for the computation of feasible robot motions. However, a suitable formalism is required to relate the mathematical expressions describing the task to the robots appointed to execute it, and consequently generate proper optimization problems. In other words, a systematic procedure is required to define tasks in a way that enables the automatic generation of robot motion. To this end, this chapter formulates a general constraint-based programming framework for the specification of tasks as minimum sets of constraints and the automatic generation of motion optimization problems. The framework can handle general constraints, whether they are related to the task description or the robotic mechanism, and includes an explicit time dependency, to better handle time-varying quantities. The proposed formalism naturally scales to robotic applications with multiple robots, on which multiple points of control might be of interest.

As some constraint-based programming frameworks already exists in the literature, this chapter also includes a detailed theoretical comparison with prior work in this field. It will also be shown that classic robot programming based on Cartesian frames is not excluded by the proposed framework. In fact, it will represent a special case within the introduced formalism.

The validity and the effectiveness of the proposed approach is numerically supported by two illustrative examples, involving both single and multi-robot applications. Finally, the three case studies from Sect. 2.2 are reconsidered in light of the proposed formalism, and solved with the methodology provided by the framework.

3.1.1 Related work

Although constraint-based programming frameworks have been mainly developed in recent years, the idea of programming tasks using a minimum number of constraints pre-

dates their formulation by some decades. Basic concepts are already present in the work by Ambler and Popplestone (1975), in which positioning tasks are characterized for the first time through geometric relations between two objects. However, after inferring a valid relative pose of the two objects, a desired Cartesian frame is computed for the robot end-effector that has to assemble the two objects.

The formulation of the so-called *task function approach* by Samson, Espiau, et al. (1991) has been another significant step, introducing a formalism to describe tasks as positioning problems characterized by a vector function of joint positions and time. Solving the task then results into a regulation problem of the task function. Since the robot often presents more DoFs than needed to perform the assigned tasks, numerical methods based on the operational space formulation (Dietrich and Ott 2019; Fiore, Meli, et al. 2023; Khatib 1983; Sentis and Khatib 2004; Siciliano and J.-J. Slotine 1991) are needed to solve the control problem. This approach has been later used to successfully solve different classes of applications (Espiau, Chaumette, et al. 1992; Fruchard, Morin, et al. 2006; Marchand, Chaumette, et al. 1996; Mezouar and Chaumette 2002).

Another method for the formalization of task descriptions was proposed by Bruyninckx and De Schutter (1996). This work introduces the *task frame*, a frame of interest in which position and/or force constraints can be conveniently expressed. Based on similar ideas, Basile, Caccavale, et al. (2012) proposed a task-oriented motion planning for cooperative multi-robot applications. However, these methods highly relies on the chosen frames and mostly apply for simple task geometries.

A first attempt at formalizing a constraint-based programming framework has been given in the work by De Schutter, De Laet, et al. (2007), in which the iTaSC (instantaneous Task Specification using Constraints) framework is introduced. The work proposes a general and systematic procedure for task specification and control, including handling of sensor data and geometric uncertainties. The approach is based on the definition of a set of auxiliary variables, called *feature coordinates*. These coordinates are defined with respect to additional *object frames* and *feature frames*, suitably chosen to simplify the task specification. Geometric uncertainties are also explicitly modeled through an additional set of coordinates. The task specification is directly coupled to the generation of constrained optimization problem. Specific work on the framework has been dealing with particular control aspects, like the handling of inequality constraints (Decre, Smits, et al. 2009), time-independent trajectories (Decré, Bruyninckx, et al. 2013), and human-robot collaboration scenarios (Vanthienen, De Laet, et al. 2012).

The use of feature coordinates and, more specifically, the direct correspondence of these coordinates to Virtual Kinematic Chains (VKCs), is also the basis of the framework by Kresse and Beetz (2012), which combines a specification layer based on symbolic instructions with the iTaSC framework. However, feature coordinates have shown significant drawbacks related to an additional computational burden and the possibility of representation singularities. For this reason a number of other constraint-based programming frameworks has been proposed in recent years, which avoids the use of such coordinates. The work by Mansard, Stasse, et al. (2009), for example, presents a framework based on the task function approach that generalizes the inverse kinematics problem. Nonetheless, no strict methodology is provided for specifying tasks and generate the control problem. The framework by Bartels, Kresse, et al. (2013) instead, avoids

feature coordinates by introducing an explicit dependency of the task function from the pose of the robot end effector. The introduced formalism, however, assumes fixed manipulated objects and does not consider the possibility of joint space constraints. Differently from the above-mentioned frameworks, the eTaSL/eTC (expressiongraph-based Task Specification/expressiongraph-based Task Controller), attempts at using feature coordinates exclusively to simplify the constraint expression, avoiding their use in the definition of position closure equations. While avoiding representation singularities and computational issues, this solution leads to optimization problems with mixed units, whose drawbacks are better discussed in Sect. 3.3.2.

3.2 Mathematical Preliminaries

Constraint-based programming is based on techniques similar to those typically employed to carry out the kinematic analysis of a robot, i.e., to perform the computation of the position, velocity, and acceleration of a given robotic mechanism (Nikravesh 1988). Without loss of generality, this section focuses on the analysis of serial open-chain mechanisms that only present revolute or prismatic single-DoF joints.

Consider the robot in Fig. 3.1. The pose of all bodies (*links*) composing the mechanism structure can be uniquely specified by a set of coordinates. An infinite variety of sets exists to describe the same kinematic structure (Lachner, Schettino, et al. 2020). However, some choices are preferable, since they significantly simplify the mathematical equations. In particular, a common choice is to select Lagrangian coordinates that describe the pose of each body relatively to the pose of the previous one. Moreover, as the kinematic analysis is always applied to a closed chain of bodies, Cartesian coordinates are typically chosen to describe the pose of the nth body of the mechanism with respect to a fixed global coordinate system. Including the dependency on the time t, the described set of coordinates can be expressed as

$$\boldsymbol{\theta}(t) = (\boldsymbol{q}(t), \boldsymbol{p}(t)),$$

with $\boldsymbol{q} \in \mathcal{Q} \subseteq \mathbb{R}^n$ being the vector of joint angles and/or translations describing the robot configuration, and $\boldsymbol{p} \in \mathcal{P} \subset \mathbb{R}^p$ a vector representing the pose of a chosen frame fixed to the nth body. The position analysis consists in obtaining the value of $\boldsymbol{\theta}$ at any given instant of time.

Since the mechanism joints impose certain conditions on the relative motion of the links they connect, a certain relation exists between the coordinate vectors $\boldsymbol{q}$ and $\boldsymbol{p}$. This can be expressed by a set of h constraint equations

$$\boldsymbol{\Phi}(\boldsymbol{q}(t), \boldsymbol{p}(t)) = \boldsymbol{0}, \tag{3.1}$$

known as *natural constraints* of the mechanism (Mason 1981). Since they describe position closure equations, the constraints in (3.1) are also known as *kinematic loop constraints* or *position loop constraints* (De Schutter, De Laet, et al. 2007). Considering (3.1), the value of $n + p - h$ coordinates must be known at any given instant of time to perform the position analysis. In this way, the constraint equations can be solved for the

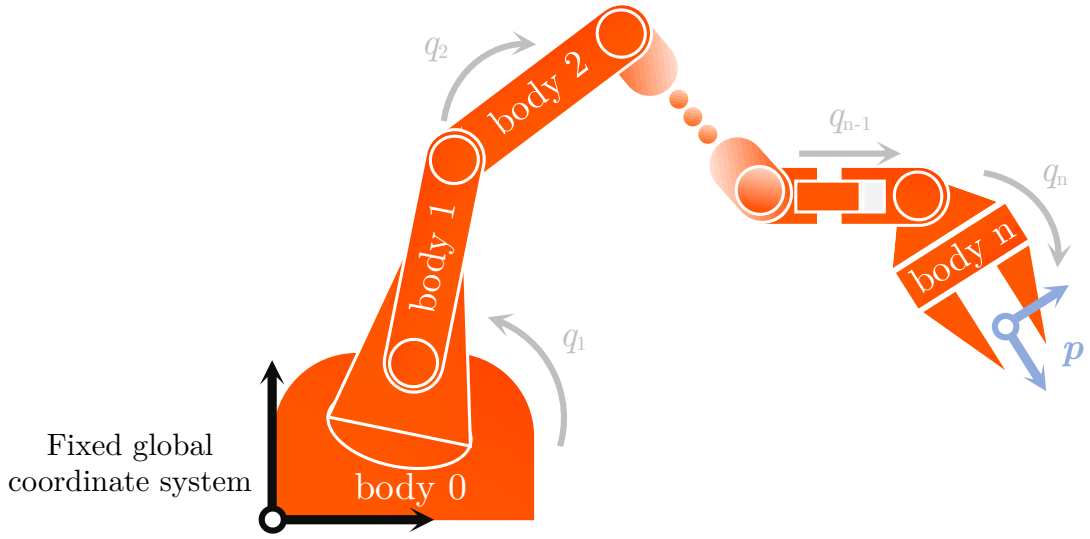


Figure 3.1: A serial open-chain mechanism and the set of coordinates chosen to describe the pose of all its bodies: $\boldsymbol{q} = [q_1 \quad q_2 \quad \dots \quad q_n]^T$ is the set of joint coordinates describing the robot configuration, whereas $\boldsymbol{p}$ expresses the pose of a frame fixed to the nth body with respect to the fixed global coordinate system.

other (unknown) coordinates. A simple, yet effective technique, is the so-called method of *Appended Driving Constraints* (Nikravesh 1988). In this method, additional constraint equations, named *driving constraints*, are introduced to explicitly specify $n+p-h$ independent coordinates as functions of time. Thus, the analysis is carried out by solving the system of equations composed by natural and driving constraints. Finally, considering the first and second time derivatives of this system of equations allows to carry out the velocity and the acceleration analyses.

It should be noticed that for the considered kinematics, (3.1) can be rewritten using the function

$$\boldsymbol{f} : \boldsymbol{q} \in \mathcal{Q} \subseteq \mathbb{R}^n \rightarrow \boldsymbol{p} \in \mathcal{P} \subseteq \mathbb{R}^p, \quad \boldsymbol{f}(\boldsymbol{q}(t)) = \boldsymbol{p}(t),$$

also known as *forward kinematics function*. In this case, p independent natural constraints exist, namely $h = p$. Therefore, n driving constraints are necessary to perform the kinematic analysis. In particular, explicitly specifying the coordinates $\boldsymbol{q}$ as driving constraints ($\boldsymbol{q}(t) = \boldsymbol{q}_d(t)$) returns the well-known *forward kinematics problem*

$$\begin{cases} \boldsymbol{f}(\boldsymbol{q}(t)) - \boldsymbol{p}(t) = \boldsymbol{0} & \text{natural constraints} \\ \boldsymbol{q}(t) - \boldsymbol{q}_d(t) = \boldsymbol{0} & \text{appended driving constraints} \end{cases}.$$

3.3 Constraint-Based Programming Framework

Starting from the methodology introduced in the previous section, it is possible to derive a systematic procedure for building systems of constraint equations, whose solution represents a suitable joint motion, $\boldsymbol{q}(t)$, that allows the robot to perform a given task. In

the remainder of this chapter, it is assumed $\boldsymbol{p} = [\boldsymbol{p}_p^T \quad \boldsymbol{p}_o^T]^T$, with $\boldsymbol{p}_p \in \mathcal{P}_p \subseteq \mathbb{R}^3$ denoting the position of the frame, and $\boldsymbol{p}_o \in \mathcal{P}_o \subseteq \mathbb{R}^4$ expressing its orientation in terms of unit quaternions represented by 4-dimensional vectors.

3.3.1 Constraint-based task formulation

The basic idea behind the presented constraint-based programming framework is that a task can be mathematically specified as a set of m constraints, which have the generic form

$$\boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) = \boldsymbol{0}, \tag{3.2}$$

where

$$\boldsymbol{e} : (\boldsymbol{q} \in \mathcal{Q}, \boldsymbol{p} \in \mathcal{P}, t \in \mathbb{R}^+) \rightarrow \boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) \in \mathbb{R}^m \tag{3.3}$$

can be seen as a task function with explicit dependency on the frame $\boldsymbol{p}$ and the time t. For the scope of this work, the frame represented by the vector of coordinates $\boldsymbol{p}$ will also be referred to as *Point of Control (PoC)*.

Being derived merely from the task description, the constraints in (3.2) do not depend on the specific robot structure. Thus, they are referred to as *artificial constraints* (De Schutter, De Laet, et al. 2007), as opposed to the natural ones. Nevertheless, it is possible to use artificial constraints as appended driving constraints to build the system of constraint equations

$$\begin{cases} \boldsymbol{f}(\boldsymbol{q}(t)) - \boldsymbol{p}(t) = \boldsymbol{0} & \text{natural constraints} \\ \boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) = \boldsymbol{0} & \text{artificial constraints} \end{cases}, \tag{3.4}$$

and solve it for $\boldsymbol{q}(t)$ in order to find the robot motion fulfilling the given task specification, if there exist one. Depending on the specific combination of task and kinematic structure, there might exist one or even multiple solutions. However, the typical nonlinearity of the functions $\boldsymbol{f}(\cdot)$ and $\boldsymbol{e}(\cdot)$ makes the system (3.4) very difficult to solve in practice. Closed-form solutions might be available only for special cases. Furthermore, if $m < n$, i.e., the robot is redundant with respect to the assigned task, infinite solutions might exist and the setup of a nonlinear optimization problem would be required for any instant of time.

Given the aforementioned considerations, it is worth shifting (3.4) to the first-order differential level. In this way, it is possible to obtain equations that are linear with respect to the time derivatives of $\boldsymbol{q}$ and $\boldsymbol{p}$. Assuming $\boldsymbol{f}(\cdot)$ and $\boldsymbol{e}(\cdot)$ are of class $\mathcal{C}^1$, and differentiating (3.4) with respect to the time yields

$$\begin{cases} \boldsymbol{J}_a(\boldsymbol{q}(t))\,\dot{\boldsymbol{q}}(t) - \dot{\boldsymbol{p}}(t) = \boldsymbol{0} \\ \boldsymbol{E}_q(\boldsymbol{q}(t), \boldsymbol{p}(t), t)\dot{\boldsymbol{q}}(t) + \boldsymbol{E}_{p_a}((\boldsymbol{q}(t), \boldsymbol{p}(t), t)\dot{\boldsymbol{p}}(t) + \\ \qquad\qquad + \boldsymbol{e}_t(\boldsymbol{q}(t), \boldsymbol{p}(t), t) = \boldsymbol{0} \end{cases}, \tag{3.5}$$

with $\boldsymbol{J}_a = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} \in \mathbb{R}^{7\times n}$, $\boldsymbol{E}_q = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{q}} \in \mathbb{R}^{m\times n}$, $\boldsymbol{E}_{p_a} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{p}} \in \mathbb{R}^{m\times 7}$, and $\boldsymbol{e}_t = \frac{\partial \boldsymbol{e}}{\partial t} \in \mathbb{R}^m$. The matrix $\boldsymbol{J}_a$ is also known as the robot (analytical) Jacobian matrix related to the PoC (Siciliano, Sciavicco, et al. 2009). For the sake of readability, dependencies are omitted

in the remainder of this chapter, unless essential for the characterization of new functions and variables.

It should be noticed that $\dot{\boldsymbol{p}} = [\dot{\boldsymbol{p}}_p^T \quad \dot{\boldsymbol{p}}_o^T]^T$ contains the time derivative of the unit quaternion represented by $\boldsymbol{p}_o$. From a robot control perspective, a more suitable choice to describe the changing rate of the orientation $\boldsymbol{p}_o$ is represented by the body angular velocity, $\boldsymbol{\omega} \in \mathbb{R}^3$. Nevertheless, a well-known relation exists between $\dot{\boldsymbol{p}}_o$ and $\boldsymbol{\omega}$. In fact, expressing the orientation as $\boldsymbol{p}_o = [\eta \quad \epsilon_x \quad \epsilon_y \quad \epsilon_z]^T$, with η and $\boldsymbol{\epsilon} = [\epsilon_x \quad \epsilon_y \quad \epsilon_z]^T$ being the scalar and vector part of the quaternion, respectively, it is possible to write

$$\dot{\boldsymbol{p}}_o = \frac{1}{2}\boldsymbol{T}\boldsymbol{\omega}, \tag{3.6}$$

with

$$\boldsymbol{T} = \begin{bmatrix} -\epsilon_x & -\epsilon_y & -\epsilon_z \\ \eta & \epsilon_z & -\epsilon_y \\ -\epsilon_z & \eta & \epsilon_x \\ \epsilon_y & -\epsilon_x & \eta \end{bmatrix}.$$

Substituting (3.6) in (3.5), and defining

$$\boldsymbol{T}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \frac{1}{2}\boldsymbol{T} \end{bmatrix} \in \mathbb{R}^{7\times 6}, \quad \boldsymbol{T}'' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & 2\boldsymbol{T}^T \end{bmatrix} \in \mathbb{R}^{6\times 7}, \tag{3.7}$$

with $\boldsymbol{I}$ being the identity matrix, yields

$$\begin{cases} \boldsymbol{J}\dot{\boldsymbol{q}} - \boldsymbol{v} = \boldsymbol{0} \\ \boldsymbol{E}_q\dot{\boldsymbol{q}} + \boldsymbol{E}_p\boldsymbol{v} + \boldsymbol{e}_t = \boldsymbol{0} \end{cases}, \tag{3.8}$$

where $\boldsymbol{J} = \boldsymbol{T}''\boldsymbol{J}_a \in \mathbb{R}^{6\times n}$ is the well-known geometric robot Jacobian matrix related to the PoC, $\boldsymbol{v} = [\dot{\boldsymbol{p}}_p^T \quad \boldsymbol{\omega}^T]^T \in \mathbb{R}^6$ is the vector of linear and angular PoC velocity, and $\boldsymbol{E}_p = \boldsymbol{E}_{p_a}\boldsymbol{T}' \in \mathbb{R}^{m\times 6}$. Another way to visualize (3.8) is in its matrix form

$$\begin{bmatrix} \boldsymbol{J} & -\boldsymbol{I} \\ \boldsymbol{E}_q & \boldsymbol{E}_p \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ -\boldsymbol{e}_t \end{bmatrix}.$$

The matrix

$$\boldsymbol{M}_c = \begin{bmatrix} \boldsymbol{J} & -\boldsymbol{I} \\ \boldsymbol{E}_q & \boldsymbol{E}_p \end{bmatrix} \tag{3.9}$$

will be referred to as *constraint matrix* in the remainder of this chapter.

The system of equations (3.8) is linear with respect to $\dot{\boldsymbol{q}}$ and $\boldsymbol{v}$ and, therefore, can be easily solved. Isolating the PoC velocity $\boldsymbol{v}$ in the first equation of (3.8) and substituting it in the second equation yields

$$\left(\boldsymbol{E}_q + \boldsymbol{E}_p\boldsymbol{J}\right)\dot{\boldsymbol{q}} + \boldsymbol{e}_t = \boldsymbol{0}. \tag{3.10}$$

Denoting with $\boldsymbol{J}_c = \boldsymbol{E}_q + \boldsymbol{E}_p\boldsymbol{J}$ and $\boldsymbol{r}_{v_c} = -\boldsymbol{e}_t$, eq. (3.10) can be finally rewritten as

$$\boldsymbol{J}_c\dot{\boldsymbol{q}} = \boldsymbol{r}_{v_c}. \tag{3.11}$$

The matrix $\boldsymbol{J}_c = \boldsymbol{J}_c(\boldsymbol{q}(t), t)$ will be referred to as *constraint Jacobian matrix*, whereas $\boldsymbol{r}_{v_c} = \boldsymbol{r}_{v_c}(t)$ will be annotated as *constraint reference velocity*.

Assuming $\boldsymbol{J}_c$ to be nonsingular, it can be easily noticed that an inversion of the constraint Jacobian matrix would allow to compute the evolution of the joint velocities that satisfy (3.11). However, in case of redundant robots, $\boldsymbol{J}_c$ is a rectangular matrix and, thus, not invertible. Nevertheless, (3.11) can be solved in a minimum-norm sense by formulating the optimization problem

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \ & \frac{1}{2}\dot{\boldsymbol{q}}^T\boldsymbol{H}\dot{\boldsymbol{q}}, \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t. } & \boldsymbol{J}_c\dot{\boldsymbol{q}} = \boldsymbol{r}_{v_c} \end{aligned}, \tag{3.12}$$

which has a quadratic cost function and linear (equality) constraints. Depending on the initial value of the function (3.3), however, fulfilling (3.11) might not guarantee the satisfaction of the original constraints (3.2). To solve this issue, a desired evolution of the task function can be imposed by modifying the constraint reference velocity $\boldsymbol{r}_{v_c}$. For example, the choice (Balestrino, De Maria, et al. 1984; Sciavicco and Siciliano 1986)

$$\boldsymbol{r}_{v_c} = -\boldsymbol{e}_t - \boldsymbol{K}\boldsymbol{e}, \quad \boldsymbol{K} > \boldsymbol{0} \tag{3.13}$$

imposes an evolution of the task function as a first-order linear system

$$\dot{\boldsymbol{e}} + \boldsymbol{K}\boldsymbol{e} = \boldsymbol{0},$$

with a convergence rate depending on the eigenvalues of $\boldsymbol{K}$.

One last aspect to consider is that many tasks are more naturally described through inequality constraints. In such case, the artificial constraints (3.2) take the form

$$\boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) \leq \boldsymbol{0}, \tag{3.14}$$

where inequalities are intended component-wise. Thus, Eq. (3.4) becomes

$$\begin{cases} \boldsymbol{f}(\boldsymbol{q}(t)) - \boldsymbol{p}(t) = \boldsymbol{0} & \text{natural constraints} \\ \boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{p}(t), t) \leq \boldsymbol{0} & \text{artificial constraints} \end{cases},$$

which, by time differentiation, returns

$$\begin{cases} \boldsymbol{J}\dot{\boldsymbol{q}} - \boldsymbol{v} = \boldsymbol{0} \\ \boldsymbol{E}_q\dot{\boldsymbol{q}} + \boldsymbol{E}_p\boldsymbol{v} + \boldsymbol{e}_t \leq \boldsymbol{0} \end{cases}. \tag{3.15}$$

Following the same mathematical steps seen for the case of equality constraints yields

$$\boldsymbol{J}_c\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{v_c}, \tag{3.16}$$

with $\overline{\boldsymbol{r}}_{v_c} = -\boldsymbol{e}_t$ being the *constraint velocity upper bound*, which can be modified to impose a desired evolution of the task function and, consequently, the satisfaction of (3.14). For example, the choice (Aertbeliën and De Schutter 2014)

$$\overline{\boldsymbol{r}}_{v_c} = -\boldsymbol{e}_t - \boldsymbol{K}\boldsymbol{e}, \quad \boldsymbol{K} > \boldsymbol{0} \tag{3.17}$$

imposes an evolution of the task function that is no faster than a first-order linear system with a convergence rate depending on the eigenvalues of $\boldsymbol{K}$. Finally, the optimization problem (3.12) takes the form

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \ & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{H} \dot{\boldsymbol{q}}, \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t. } & \boldsymbol{J}_c \dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{v_c} \end{aligned}. \tag{3.18}$$

Considering the possibility of having both equality and inequality constraints, a more general form of the optimization problem computing the robot joint motion is

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}} \ & \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{H} \dot{\boldsymbol{q}}, \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t.} \quad & \boldsymbol{J}_{eq,c} \dot{\boldsymbol{q}} = \boldsymbol{r}_{v_c}, \quad \underline{\boldsymbol{r}}_{v_c} \leq \boldsymbol{J}_{iq,c} \dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{v_c} \end{aligned}, \tag{3.19}$$

where the subscripts *eq* and *iq* indicate the Jacobian matrices that are related to the equality and inequality constraints, respectively, and the inequality constraints have been expressed with respect to lower and upper bounds. Techniques to solve the class of optimization problems in (3.19) are thoroughly discussed in Chap. 4.

3.3.2 Relation to previous work

The systematic procedure introduced in the previous section allows translating constraint-based task specifications into optimization problems from which it is possible to obtain the robot joint motion accomplishing the given task. The proposed methodology is based on a general constraint definition, which directly employs the set of coordinates typically used for the kinematic analysis of robots. Moreover, by introducing the concept of PoC, the proposed framework can consider constraints involving frames fixed to any of the robot body and it is, therefore, not restricted only to constraints on the end-effector pose.

The method is clearly inspired by the task function approach by Samson, Espiau, et al. (1991). However, the function introduced in (3.3) has an explicit dependency on the PoC pose, $\boldsymbol{p}$, which allows a full separation between task-related and robot-related variables. Indeed, the specific kinematic chain only contributes to the definition of the natural constraints in (3.8), while all the other terms involved in the system, namely $\boldsymbol{E}_q$, $\boldsymbol{E}_p$, and $\boldsymbol{e}_t$, are completely robot-independent. On the other hand, the explicit time dependency of (3.3) enables an appropriate handling of time-varying constraints, improving the overall task execution. It is also worth noticing that if the artificial constraints are chosen as

$$\boldsymbol{p} - \boldsymbol{p}_d(t) = \boldsymbol{0},$$

with $\boldsymbol{p}_d$ being a desired pose for the PoC, the system (3.4) reduces to the well-known *inverse kinematics problem*. Therefore, classic task programming based on Cartesian frames is not excluded by the proposed constraint-based task description. On the contrary, it represents a special case within the proposed framework. The same applies for pure joint motion specifications, which can be obtained with a task function $\boldsymbol{e}(\boldsymbol{q}, t) = \boldsymbol{q} - \boldsymbol{q}_d(t)$.

As pointed out in Sect. 3.1.1, literature already offers a number of constraint-based approaches for task programming, presenting similar features as the one introduced in Sect. 3.3.1. In particular, the work on the iTaSC framework by De Schutter, De Laet, et al. (2007) and Decre, Smits, et al. (2009) presents the same systematic modeling procedure and a similar formalism. However, one main difference lies in the choice of the set of coordinates describing the pose of the PoC. In place of Cartesian coordinates, iTaSC introduces a new set of generic coordinates, named *feature coordinates*, here indicated by the vector $\boldsymbol{\chi} \in \mathcal{X} \subseteq \mathbb{R}^6$. The choice of the feature coordinates is left to the user and can be done based on the specific application at hand. Considering a generic set of feature coordinates, the artificial constraints (3.14) can be expressed as

$$\boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{\chi}(t), t) \leq \boldsymbol{0}. \tag{3.20}$$

It should be noticed that, although (3.20) describes inequality constraints, the considerations made in this section are not limited to this kind of constraints. In fact, since the considered inequalities are nonstrict, it is always possible to put equality constraints in the form (3.20) with some suitable manipulation.

Considering the vector of feature coordinates, the system of all constraints acting on the robot can be expressed as

$$\begin{cases} \boldsymbol{f}_{q,\chi}(\boldsymbol{q}(t), \boldsymbol{\chi}(t)) = \boldsymbol{0} & \text{natural constraints} \\ \boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{\chi}(t), t) \leq \boldsymbol{0} & \text{artificial constraints} \end{cases}, \tag{3.21}$$

where the natural constraints now describe position closure equations depending on $\boldsymbol{q}$ and $\boldsymbol{\chi}$. Another way of writing (3.21) is

$$\begin{cases} \boldsymbol{f}(\boldsymbol{q}(t)) - \boldsymbol{g}(\boldsymbol{\chi}(t)) = \boldsymbol{0} & \text{natural constraints} \\ \boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{\chi}(t), t) \leq \boldsymbol{0} & \text{artificial constraints} \end{cases}, \tag{3.22}$$

with $\boldsymbol{g} : \boldsymbol{\chi} \in \mathcal{X} \subseteq \mathbb{R}^6 \to \boldsymbol{p} \in \mathcal{P} \subseteq \mathbb{R}^p$, $\boldsymbol{g}(\boldsymbol{\chi}(t)) = \boldsymbol{p}(t)$. Accordingly, the first-order differential constraints can be written as

$$\begin{cases} \boldsymbol{J}\dot{\boldsymbol{q}} - \boldsymbol{J}_\chi \dot{\boldsymbol{\chi}} = \boldsymbol{0} \\ \boldsymbol{E}_q \dot{\boldsymbol{q}} + \boldsymbol{E}_\chi \dot{\boldsymbol{\chi}} + \boldsymbol{e}_t \leq \boldsymbol{0} \end{cases}, \tag{3.23}$$

with $\boldsymbol{J}_\chi = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\chi}} \in \mathbb{R}^{6\times 6}$, and $\boldsymbol{E}_\chi = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\chi}} \in \mathbb{R}^{m\times 6}$. Therefore, the constraint matrix (3.9) becomes

$$\boldsymbol{M}_c = \begin{bmatrix} \boldsymbol{J} & -\boldsymbol{J}_\chi \\ \boldsymbol{E}_q & \boldsymbol{E}_\chi \end{bmatrix},$$

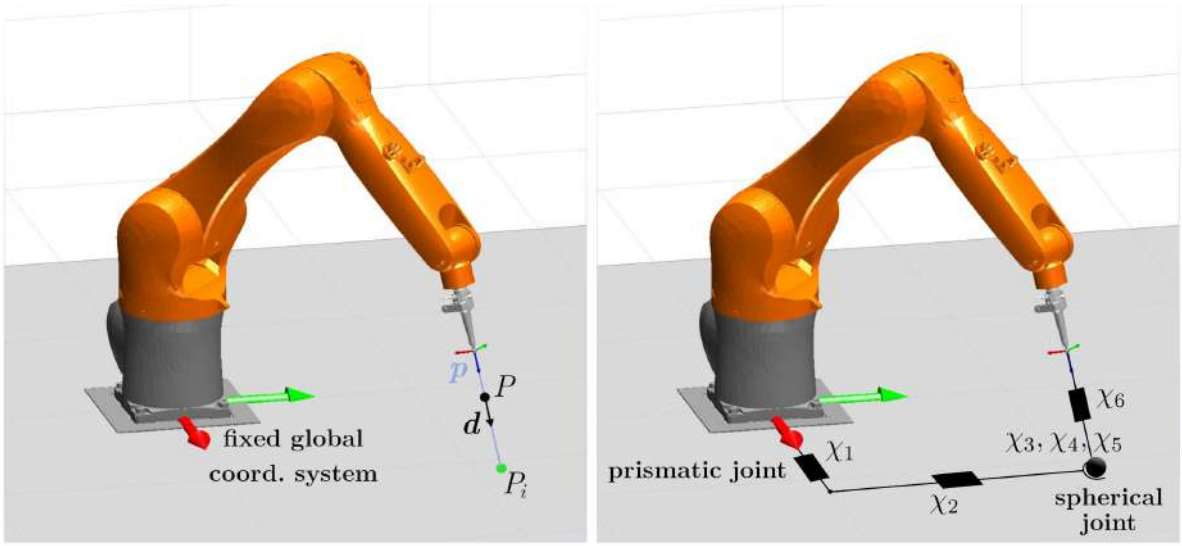


Figure 3.2: Application setup (left) and a possible VKC (right) for the laser tracing application.

and the system (3.23) can be brought to the form (3.16) by defining the constraint Jacobian matrix as

$$\boldsymbol{J}_c = \left(\boldsymbol{E}_q + \boldsymbol{E}_\chi \boldsymbol{J}_\chi^{-1} \boldsymbol{J}\right). \tag{3.24}$$

In the original work by De Schutter, De Laet, et al. (2007), the feature coordinates are defined after introducing a set of reference frames in which to express relative motions of the objects involved in a certain task. However, by looking at the (3.22) and (3.23), it is easy to think of feature coordinates as the joints of a 6-DoFs Virtual Kinematic Chain (VKC), characterized by forward kinematic function $\boldsymbol{g}(\boldsymbol{\chi})$ and (analytical) Jacobian $\boldsymbol{J}_\chi(\boldsymbol{\chi})$. The idea of VKCs has been extensively exploited within the iTaSC framework and other later works (Kresse and Beetz 2012; Vanthienen, De Laet, et al. 2012), since it allows to link the specification and the execution of a task to the definition of a (virtual) kinematic structure that simply needs to be controlled in the joint space. This strategy is illustrated in Fig. 3.2 for a laser tracing application. In this case, the vector of feature coordinates $\boldsymbol{\chi} = [\chi_1 \quad \dots \quad \chi_6]$ can be chosen such that χ_1 and χ_2 represent the x and y coordinates (in a suitably chosen coordinate system fixed to the work table) of the point located at the intersection of the laser beam with the work table, P_i. Therefore, the first two joints of the associated VKC are prismatic, allowing translation along the x and y direction, respectively. The coordinates $\chi_3 - \chi_5$ can instead be chosen so as to form a 3-DoFs spherical joint. That allows a full reorientation of the last joint of the VKC, which is chosen as a prismatic joint with the sliding axis along the direction of the laser beam.

The employment of a VKC surely helps visualizing the elementary translations/rotations describing the pose of the PoC according to the specific feature coordinates definition. Even more importantly, a suitable definition of the VKC (or, equivalently, of the feature coordinate set) can significantly simplify the expression of the artificial constraints. As

Algorithm 3.1 Fulfillment of the position closure equation in case of VKCs

1: $\begin{pmatrix} \text{a solution } \dot{\boldsymbol{q}} \text{ is computed for a robot with joint configuration } \boldsymbol{q}_{prev}, \\ \text{which corresponds to a value } \boldsymbol{\chi}_{prev} \text{ of the feature coordinate vector} \end{pmatrix}$
2: $\boldsymbol{q}_{next} = \boldsymbol{q}_{prev} + T\dot{\boldsymbol{q}}$
3: $\dot{\boldsymbol{\chi}} = \boldsymbol{J}_{\chi}^{-1}(\boldsymbol{\chi}_{prev})\boldsymbol{J}(\boldsymbol{q}_{prev})\dot{\boldsymbol{q}}$
4: $\boldsymbol{\chi}_{next} = \boldsymbol{\chi}_{prev} + T\dot{\boldsymbol{\chi}}$
5: **repeat**
6: $\quad \Delta_p = \text{compute\_displacement}(\boldsymbol{f}(\boldsymbol{q}_{next}), \boldsymbol{g}(\boldsymbol{\chi}_{next}))$
7: $\quad \dot{\boldsymbol{\chi}} = \boldsymbol{J}_{\chi}^{-1}(\boldsymbol{\chi}_{next})\Delta_p$
8: $\quad \boldsymbol{\chi}_{next} = \boldsymbol{\chi}_{prev} + T\dot{\boldsymbol{\chi}}$
9: $\quad \boldsymbol{\chi}_{prev} = \boldsymbol{\chi}_{next}$
10: **until** $\boldsymbol{f}(\boldsymbol{q}_{next}) - \boldsymbol{g}(\boldsymbol{\chi}_{next}) = \boldsymbol{0}$

an example, assuming for the laser tracing application a desired target point $P_d(x_d, y_d, 0)$, the artificial constraints describing the task can be simply written as

$$\begin{cases} \chi_1 - x_d = 0 \\ \chi_2 - y_d = 0 \end{cases} .$$

On the other hand, VKCs reveals some of the critical issues of using feature coordinates. In fact, great care is generally necessary when choosing the six feature coordinates for a given robot application, as they must be always representative of the 6-DoFs PoC pose. Moreover, even meaningful sets might contain representation singularities, i.e., specific values of one or more coordinates for which the PoC pose is no longer uniquely defined by the chosen set. In the laser tracing example, this is the case when $\chi_4 = 0 \pm k\pi$, $k \in \mathbb{Z}$, i.e., when the laser beam is parallel to the work table. This corresponds to a well-known singularity of spherical joints. Indeed, considering the parallelism between feature coordinates and VKCs, it should be noticed that a representation singularity of the feature coordinates actually corresponds to a singular configuration of the associated VKC. In such configuration, the Jacobian matrix $\boldsymbol{J}_{\chi}$ loses rank, and its inversion in (3.24) is no longer possible.

Another drawback in handling the task execution through VKCs derives from the additional computational burden that is typically required to ensure that the natural constraints in (3.22) hold at every instant of time. In fact, while (3.23) allows to compute a solution $\dot{\boldsymbol{q}}$ and, thus, to obtain the value of the VKC joint velocity $\dot{\boldsymbol{\chi}}$, numerical integration is needed in practice to update the value of $\boldsymbol{q}$ and $\boldsymbol{\chi}$ at a given time. As the solution of such operation suffers from numerical drift, multiple steps might be necessary to make sure that the position closure equation in (3.22) actually holds. To give a better idea, Alg. 3.1 shows a simple method that is typically employed to implement such steps. The method is based on forward Euler integration (performed at a sampling time T) and Newton-Raphson iterations.

Compared to the iTaSC formulation, the framework proposed in Sect. 3.3.1 could be seen as a special case in which a specific set of coordinates returns a differential task

description where $\boldsymbol{J}_\chi = \boldsymbol{I}$. However, this special case offers several benefits. Indeed, resorting to Cartesian coordinates provides a singularity-free representation of the PoC pose and velocity. As a result, the computation and the consequent inversion of an additional Jacobian matrix is avoided. Moreover, no additional computational steps are necessary to ensure the fulfillment of the position closure equations. On the other hand, the artificial constraints might present more complex expressions when Cartesian coordinates are employed. Recalling the geometric constraints formulated for the laser tracing application in Sect. 2.2.1, for example, the artificial constraints for this case study can be written as

$$\text{null}(\boldsymbol{d}^T(\boldsymbol{p})) \cdot (P(\boldsymbol{p}) - P_d) = \boldsymbol{0}, \tag{3.25}$$

with P and $\boldsymbol{d}$ being the point and the unit direction defining the laser beam line, respectively, and $\boldsymbol{p}$ indicating a PoC fixed to the laser device. In this work, however, artificial constraints are automatically generated out of the symbolic task descriptions (see lower level API in Fig. 2.11). Furthermore, their differentiation can be performed using automatic differentiation techniques (Rall 1981). Thus, the complexity of their mathematical expression is not considered a crucial aspect. Finally, another important aspect that can be noticed in the expression of (3.25), is that the formulation of artificial constraints in the presented framework does not depend on the selection of special reference coordinate systems.

Following similar considerations, Bartels, Kresse, et al. (2013) had already proposed a constraint-based task description favoring the use of Cartesian coordinates, as opposed to the employment of VKCs. However, their work focused on describing the spatial relationship between a robot tool and a fixed object to be manipulated. In other words, indicating with $\boldsymbol{p}$ and $\boldsymbol{p}_O$ the pose of the robot tool and the manipulated object, respectively, the artificial constraints considered in their work can be expressed as

$$\boldsymbol{c}(\boldsymbol{p}, \boldsymbol{p}_O) - \boldsymbol{c}_d(t) \leq \boldsymbol{0},$$

where $\boldsymbol{c}_d$ is a vector of set values for the function $\boldsymbol{c} : (\boldsymbol{p} \in \mathcal{P}, \boldsymbol{p}_O \in \mathcal{P}) \rightarrow \boldsymbol{c}(\boldsymbol{p}, \boldsymbol{p}_O) \in \mathbb{R}^m$. The assumption of fixed manipulated object yields $\dot{\boldsymbol{p}}_O = \boldsymbol{0}$. Therefore, at the first-order differential level the system of natural and artificial constraints can be written as

$$\begin{cases} \boldsymbol{J}\dot{\boldsymbol{q}} - \boldsymbol{v} = \boldsymbol{0} \\ \boldsymbol{C}_p\boldsymbol{v} - \dot{\boldsymbol{c}}_d \leq \boldsymbol{0} \end{cases}, \tag{3.26}$$

where $\boldsymbol{C}_p = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{p}} \boldsymbol{T}' \in \mathbb{R}^{m \times 6}$. It can be noticed that (3.26) is just a special case of (3.15).

To overcome the drawbacks related to VKCs, Aertbeliën and De Schutter (2014) have more recently proposed a new constraint-based task formulation within the eTaSL/eTC framework, which hides the closure equations contained in the natural constraints. The definition of the VKC is therefore avoided, while a suitable number of feature coordinates is only used when beneficial to the simplification of the expression of the artificial constraints. In other words, artificial constraints are expressed in the form

$$\boldsymbol{e}(\boldsymbol{q}(t), \boldsymbol{\chi}'(\boldsymbol{q}(t)), t) \leq \boldsymbol{0}, \tag{3.27}$$

where $\boldsymbol{\chi}' \in \mathcal{X}' \subseteq \mathbb{R}^{n_{\chi'}}$, with $n_{\chi'}$ being an arbitrary number of feature coordinates suitably chosen to simplify the expression of (3.27), without being necessarily related to a VKC. Differentiating (3.27) yields

$$\boldsymbol{E}_q \dot{\boldsymbol{q}} + \boldsymbol{E}_{\chi'} \dot{\boldsymbol{\chi}}' + \boldsymbol{e}_t \leq \boldsymbol{0}, \tag{3.28}$$

where $\boldsymbol{E}_{\chi'} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\chi}'} \in \mathbb{R}^{m \times n_{\chi'}}$. Equation (3.28) is then rewritten in matrix form as

$$\boldsymbol{E}_\xi \dot{\boldsymbol{\xi}} \leq \overline{\boldsymbol{r}}_{v_t}, \tag{3.29}$$

by imposing $\boldsymbol{E}_\xi = [\boldsymbol{E}_q \quad \boldsymbol{E}_{\chi'}]$, $\boldsymbol{\xi} = [\boldsymbol{q}^T \quad \boldsymbol{\chi}'^T]^T$ and $\overline{\boldsymbol{r}}_{v_c} = -\boldsymbol{e}_t$. Finally, (3.29) is solved by setting up the optimization problem

$$\begin{aligned} \min_{\dot{\boldsymbol{\xi}}} \ & \frac{1}{2} \dot{\boldsymbol{\xi}}^T \boldsymbol{H} \dot{\boldsymbol{\xi}}, \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t. } & \boldsymbol{E}_\xi \dot{\boldsymbol{\xi}} \leq \overline{\boldsymbol{r}}_{v_t} \end{aligned}. \tag{3.30}$$

It can be noticed that, although avoiding the employment of VKCs while preserving the benefits of feature coordinates, the result of this modeling procedure is in an optimization problem that minimizes a weighted combination of joint and feature coordinates. Additionally, a third set of (slack) variables are added in the work by Aertbeliën and De Schutter (2014) in order to handle constraints with lower priority. Such a complex combination of variables will most likely present mixed units, degrading the physical meaning of the cost function (Lachner, Schettino, et al. 2020), and making a thorough tuning of the weights in $\boldsymbol{H}$ necessary. Such tuning should at the same time preserve the positiveness of $\boldsymbol{H}$, while sufficiently regularizing the optimization problem. Moreover, the possibility of generating robot joint motion that respects the underlying physics of the robotic system is totally excluded by the optimization problem (3.30). Indeed, the generation of dynamically consistent robot motion requires the selection of a cost function with specific decision variables and weight matrix (Osorio and Allmendinger 2022), which is not compatible with the cost definition in (3.30). The proposed modeling procedure from Sect. 3.3.1, on the other hand, results in an optimization problem minimizing a weighted combination of the joint variables only. The possibility of having mixed units is therefore significantly reduced. Furthermore, if the robot inertia matrix is available, it is always possible to ensure dynamical consistency of the generated robot motion by a suitable definition of the matrix $\boldsymbol{H}$, as also shown in Chap. 4.

3.3.3 Extension to multiple robots and PoCs

Re-elaborating the steps in Sect. 3.3.1 in a more general context, it is possible to consider the case in which a set of artificial constraints involves multiple PoCs. In the most general case, these could be fixed to the same robot body or to different bodies. Furthermore, different robots might be also involved. This case is of particular interest in the context of collaborative multi-robot applications. Thus, this section aims at generalizing the constraint-based task formulation from Sect. 3.3.1 to the case of multiple robots having multiple PoCs.

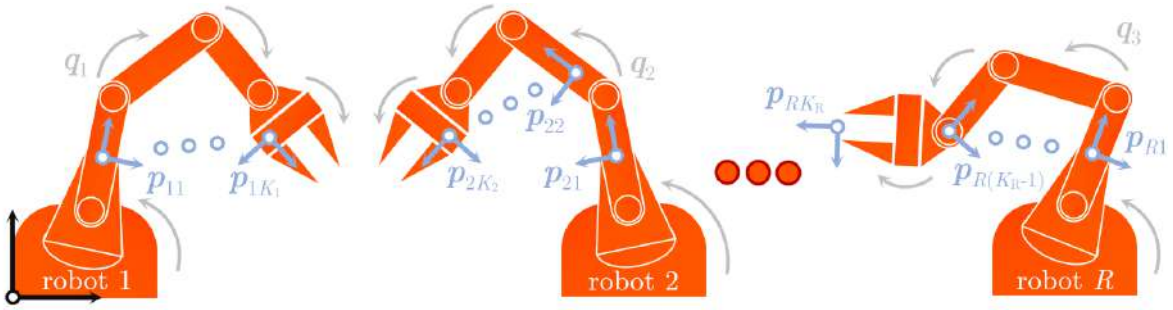


Figure 3.3: Multiple robots with multiple PoCs.

Let R be the number of robots involved in a certain set of artificial constraints, and $\boldsymbol{q}_r$ be the joint coordinate vector of the rth robot, with $r = 1, \ldots, R$. Furthermore, let K_r be the number of PoCs belonging to the rth robot, and $\boldsymbol{p}_{rk}$ be the coordinate vector describing the pose of the kth PoC on the rth robot, with $k = 1, ..., K_r$ (Fig. 3.3). Without loss of generality, let consider the case in which the set of artificial constraints is composed merely by equality constraints. Thus, the system of natural and artificial constraints can be written as

$$\begin{cases} \boldsymbol{f}_{11}(\boldsymbol{q}_1) - \boldsymbol{p}_{11} = \boldsymbol{0} & \text{natural constraints } \textit{Robot 1 – PoC 1} \\ \quad\vdots & \\ \boldsymbol{f}_{rk}(\boldsymbol{q}_r) - \boldsymbol{p}_{rk} = \boldsymbol{0} & \text{natural constraints } \textit{Robot } r \textit{ – PoC } k \\ \quad\vdots & \\ \boldsymbol{f}_{RK_R}(\boldsymbol{q}_R) - \boldsymbol{p}_{RK_R} = \boldsymbol{0} & \text{natural constr. } \textit{Robot } R \textit{ – PoC } K_R \\ \boldsymbol{e}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_R, \boldsymbol{p}_{11}, \ldots, \boldsymbol{p}_{RK_R}, t) = \boldsymbol{0} & \text{artificial constraints} \end{cases},$$

where $\boldsymbol{f}_{rk}(\cdot)$ is the forward kinematics function mapping the joint coordinates $\boldsymbol{q}_r$ to the PoC coordinates $\boldsymbol{p}_{rk}$. Consequently, the set of first-order differential constraints is

$$\begin{cases} \boldsymbol{J}_{11}\dot{\boldsymbol{q}}_1 - \boldsymbol{v}_{11} = \boldsymbol{0} \\ \quad\vdots \\ \boldsymbol{J}_{rk}\dot{\boldsymbol{q}}_r - \boldsymbol{v}_{rk} = \boldsymbol{0} \\ \quad\vdots \\ \boldsymbol{J}_{RK_r}\dot{\boldsymbol{q}}_R - \boldsymbol{v}_{RKr} = \boldsymbol{0} \\ \boldsymbol{E}_{q_1}\dot{\boldsymbol{q}}_1 + \cdots + \boldsymbol{E}_{q_R}\dot{\boldsymbol{q}}_R + \boldsymbol{E}_{p_{11}}\boldsymbol{v}_{11} + \cdots + \boldsymbol{E}_{p_{RK_R}}\boldsymbol{v}_{RK_R} = -\boldsymbol{e}_t \end{cases}, \tag{3.31}$$

where $\boldsymbol{J}_{rk}$ is the Jacobian matrix of the rth robot related to the PoC $\boldsymbol{p}_{rk}$ mapping the joint velocity $\dot{\boldsymbol{q}}_r$ to the PoC velocity $\boldsymbol{v}_{rk}$. The system of equations (3.31) can be expressed in matrix form as

$$\boldsymbol{M}_c' \begin{bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ -\boldsymbol{e}_t \end{bmatrix},$$

where $\dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{q}}_1^T & \dots & \dot{\boldsymbol{q}}_r^T & \dots & \dot{\boldsymbol{q}}_R^T \end{bmatrix}^T$, $\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_{11}^T & \dots & \boldsymbol{v}_{rk}^T & \dots & \boldsymbol{v}_{RK_R}^T \end{bmatrix}^T$, and

$$\boldsymbol{M}_c' = \begin{bmatrix} \begin{bmatrix} \boldsymbol{J}_{11} \\ \vdots \\ \boldsymbol{J}_{1K_1} \end{bmatrix} & \boldsymbol{0} & \boldsymbol{0} & \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} & \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \begin{bmatrix} \boldsymbol{J}_{R1} \\ \vdots \\ \boldsymbol{J}_{RK_R} \end{bmatrix} & \boldsymbol{0} & \boldsymbol{0} & \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} \\ \boldsymbol{E}_{q_1} & \dots & \boldsymbol{E}_{q_R} & \boldsymbol{E}_{p_{11}} & \dots & \boldsymbol{E}_{p_{RK_R}} \end{bmatrix}. \tag{3.32}$$

Analyzing (3.32), it is possible to recognize that the constraint matrix $\boldsymbol{M}_c'$ can be brought to the form (3.9) by setting

$$\boldsymbol{J} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{J}_{11} \\ \vdots \\ \boldsymbol{J}_{1K_1} \end{bmatrix} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \begin{bmatrix} \boldsymbol{J}_{R1} \\ \vdots \\ \boldsymbol{J}_{RK_R} \end{bmatrix} \end{bmatrix}, \qquad \begin{aligned} \boldsymbol{E}_q &= \begin{bmatrix} \boldsymbol{E}_{q_1} & \dots & \boldsymbol{E}_{q_R} \end{bmatrix} \\ \boldsymbol{E}_p &= \begin{bmatrix} \boldsymbol{E}_{p_{11}} & \dots & \boldsymbol{E}_{p_{RK_R}} \end{bmatrix}. \end{aligned} \tag{3.33}$$

Moreover, it can be noticed that every robot involved in the artificial constraints extends the matrix $\boldsymbol{J}$ by introducing an element on its (block) diagonal. Additionally, an element is concatenated column-wise to the matrix $\boldsymbol{E}_q$. On the other hand, the rth block on the diagonal of $\boldsymbol{J}$ consists of the concatenation of the K_r Jacobian matrices associated to the rth robot. Every PoC involved in the artificial constraints additionally adds an element to the matrix $\boldsymbol{E}_p$. From these considerations, basic rules can be derived to construct the matrices in (3.33) for a given task. Once this operation is completed, it is possible to compute the constraint Jacobian matrix as seen in Sect. 3.3.1 for the case of a single robot with one PoC, namely

$$\boldsymbol{J}_c = (\boldsymbol{E}_q + \boldsymbol{E}_p \boldsymbol{J}).$$

The constraint reference velocity (or the velocity upper bound, in the case of inequality constraints) can also be computed as in Sect. 3.3.1. This procedure allows to handle task specifications involving an arbitrary number of robots and PoCs in a unified manner.

3.3.4 Acceleration-level formulations

The previous sections have presented a systematic procedure to generate quadratic optimization problems out of constraint-based task descriptions involving an arbitrary number of robots and PoCs. The method relies on the first-order differential relations describing natural and artificial constraints of the given robot(s). As a consequence, the resulting optimization is performed at velocity level, meaning that the final output consists of joint velocities. Although this methodology may seem attractive in terms of effectiveness and mathematical simplicity, considering a second order task formulation might present some advantages. In fact, performing the optimization at acceleration level may generate smoother robot motions. Moreover, an evolution of the task function as a second order system might be of interest.

An acceleration-level specification composed by equality constraints simply requires a further time differentiation of (3.11). This operation yields

$$\boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} = \boldsymbol{r}_{a_c}, \tag{3.34}$$

where the *constraint reference acceleration*, $\boldsymbol{r}_{a_c} = -\frac{\partial \boldsymbol{e}_t}{\partial t}$, can be modified to impose a desired evolution of the task function. For example, the choice (De Luca, Oriolo, et al. 1992)

$$\boldsymbol{r}_{a_c} = -\frac{\partial \boldsymbol{e}_t}{\partial t} - \boldsymbol{D}\dot{\boldsymbol{e}} - \boldsymbol{K}\boldsymbol{e}, \quad \boldsymbol{D}, \boldsymbol{K} \geq \boldsymbol{0}$$

imposes an evolution of the task function as a second-order linear system characterized by stiffness $\boldsymbol{K}$ and damping $\boldsymbol{D}$. Analogously, for inequality constraints (3.16) becomes

$$\boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{a_c}, \tag{3.35}$$

with the *constraint acceleration upper bound*, $\overline{\boldsymbol{r}}_{a_c} = -\frac{\partial \boldsymbol{e}_t}{\partial t}$, that can be modified to impose a specific behavior to the task function. Considering the constraints (3.34) and (3.35), the joint motion that makes the robot fulfill the given task can be computed in terms of joint acceleration. More specifically, the optimization problem (3.19) can be rewritten at acceleration level as

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}}} \ & \frac{1}{2}\ddot{\boldsymbol{q}}^T\boldsymbol{H}\ddot{\boldsymbol{q}}, \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t.} \ & \boldsymbol{J}_{eq,c}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}} = \boldsymbol{r}_{a_c}, \\ & \underline{\boldsymbol{r}}_{a_c} \leq \boldsymbol{J}_{iq,c}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{a_c} \end{aligned} \tag{3.36}$$

In case of torque-controlled robots, another option often of interest is to solve (3.34) and (3.35) at torque-level, i.e., to generate robot motion in terms of generalized joint torques. This is achieved by recalling the joint space dynamic model of a robot

$$\boldsymbol{M}(\boldsymbol{q}(t))\ddot{\boldsymbol{q}}(t) + \boldsymbol{c}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) + \boldsymbol{g}(\boldsymbol{q}(t)) = \boldsymbol{\tau}(t), \tag{3.37}$$

where $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is the positive definite robot inertia matrix, $\boldsymbol{c} \in \mathbb{R}^n$ is the vector of joint Coriolis and centrifugal torques, $\boldsymbol{g} \in \mathbb{R}^n$ is the vector of the joint gravity torques, and $\boldsymbol{\tau}(t)$ is the vector of joint driving torques. Using (3.37), it is possible to express the

robot joint acceleration as $\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{c} - \boldsymbol{g})$. Thus, defining $\boldsymbol{\tau}' = \boldsymbol{\tau} - \boldsymbol{c} - \boldsymbol{g}$, the constraints (3.34) and (3.35) can be rewritten as

$$\begin{aligned} \boldsymbol{J}_c\boldsymbol{M}^{-1}\boldsymbol{\tau}' + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} &= \boldsymbol{r}_{a_c} \\ \boldsymbol{J}_c\boldsymbol{M}^{-1}\boldsymbol{\tau}' + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} &\leq \overline{\boldsymbol{r}}_{a_c} \end{aligned},$$

and the optimization problem (3.36) can be defined at torque-level as

$$\begin{aligned} \min_{\boldsymbol{\tau}'} \ & \frac{1}{2}\boldsymbol{\tau}'^T\boldsymbol{H}\boldsymbol{\tau}', \quad \boldsymbol{H} \geq \boldsymbol{0} \\ \text{s.t.} \quad & \boldsymbol{J}_{eq,c}\boldsymbol{M}^{-1}\boldsymbol{\tau}' + \dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}} = \boldsymbol{r}_{a_c}, \\ & \underline{\boldsymbol{r}}_{a_c} \leq \boldsymbol{J}_{iq,c}\boldsymbol{M}^{-1}\boldsymbol{\tau}' + \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{a_c} \end{aligned} \tag{3.38}$$

The solution of both the optimization problems (3.36) and (3.38) is thoroughly discussed in Chap. 4, in which a general framework for motion control of redundant robots is presented.

3.4 Simulations

This section presents a set of simulations that numerically supports the validity of the proposed constraint-based programming framework. First, two illustrative examples practically show how the method introduced in Sect. 3.3.1 and Sect. 3.3.3 can be effectively employed to generate a robot motion fulfilling a certain set of constraints. Then, the three case studies from Sect. 2.2 are analyzed. The results are obtained using different kinds of kinematic chains, as well as considering task-descriptions involving multiple robots and multiple PoCs.

3.4.1 Illustrative Examples

A. Single robot, single PoC

The first example involves a KUKA LBR iiwa 7-DoFs manipulator (Fig. 3.4). The task consists in keeping the tip of the robot tool (the orange cone attached to the robot flange) within a certain time-varying distance range from a moving reference point.

The point on the tool tip is indicated by $P(x, y, z)$, whereas $P_d(x_d, y_d, z_d)$ indicates the reference point. Moreover, $d_d \in \mathbb{R}^+$ is the maximum Euclidean distance that is desired between the two points. The position of the point P can be expressed as a function of a an arbitrary PoC frame $\boldsymbol{p} = [x_p \quad y_p \quad z_p \quad \eta \quad \epsilon_x \quad \epsilon_y \quad \epsilon_z]^T$, fixed to the last robot body. Thus, the task can be expressed by the inequality constraint

$$e(\boldsymbol{p}, t) < 0, \tag{3.39}$$

with

$$e(\boldsymbol{p}, t) = \|P(\boldsymbol{p}) - P_d(t)\|_2 - d_d(t). \tag{3.40}$$

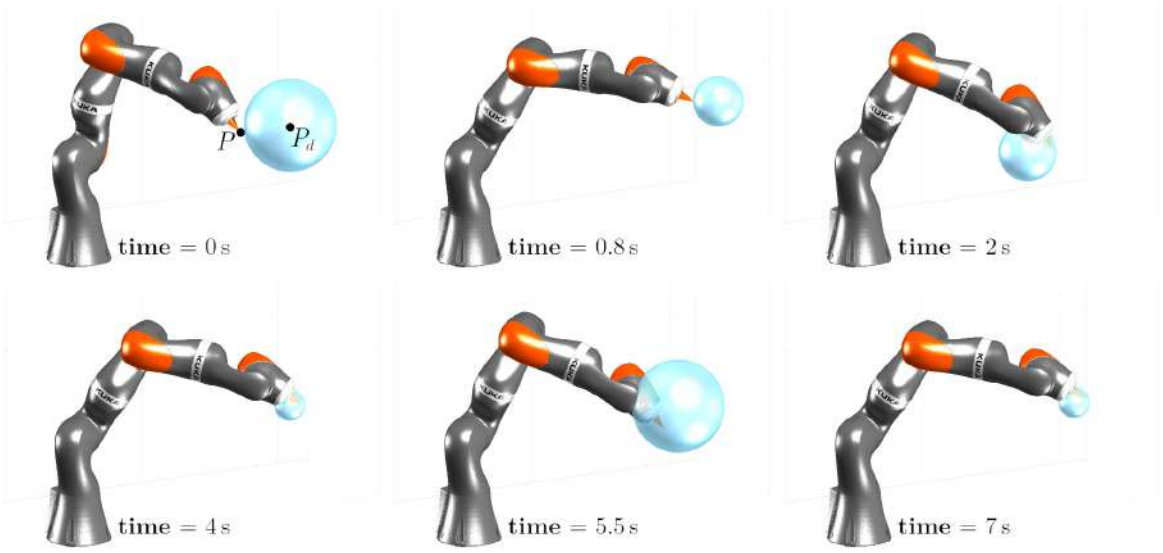


Figure 3.4: Motion sequence of the KUKA LBR iiwa robot in the simulation of Sect. 3.4.1-A.

To simplify the mathematical expressions of this section, $\boldsymbol{p}$ is placed at P (i.e., $x_p = x$, $y_p = y$ and $z_p = z$), with an arbitrary orientation. Thus, (3.40) can be rewritten as

$$e(\boldsymbol{p}, t) = \sqrt{(x_p - x_d(t))^2 + (y_p - y_d(t))^2 + (z_p - z_d(t))^2} - d_d(t),$$

from which it is straightforward to calculate

$$\begin{aligned} \boldsymbol{E}_q &= \boldsymbol{0} \\ \boldsymbol{E}_p &= \begin{bmatrix} \dfrac{x_p - x_d}{\|P - P_d\|_2} & \dfrac{y_p - y_d}{\|P - P_d\|_2} & \dfrac{z_p - z_d}{\|P - P_d\|_2} & 0 & 0 & 0 \end{bmatrix}. \\ e_t &= \frac{x_d - x_p}{\|P - P_d\|_2}\dot{x}_d + \frac{y_d - y_p}{\|P - P_d\|_2}\dot{y}_d + \frac{z_d - z_p}{\|P - P_d\|_2}\dot{z}_d - \dot{d}_d \end{aligned} \tag{3.41}$$

Using (3.41), it is possible to compute the constraint Jacobian matrix and the constraint velocity upper bound. Therefore, an optimization problem of the form (3.18) can be set up to compute the robot motion. In particular, the simulation of this section has been carried out with $\boldsymbol{H} = \boldsymbol{I}$, whereas the velocity upper bound has been defined as in (3.17), with a scalar gain $\boldsymbol{K} = k\boldsymbol{I}$ and $k = 5$. This choice of k is based on the simulation cycle time and the initial value of the task function, and ensures a transient response of the task function that is compatible with the dynamical limits of the robot. The simulation cycle time is 5 ms, whereas the total simulation time is 7 s. The position

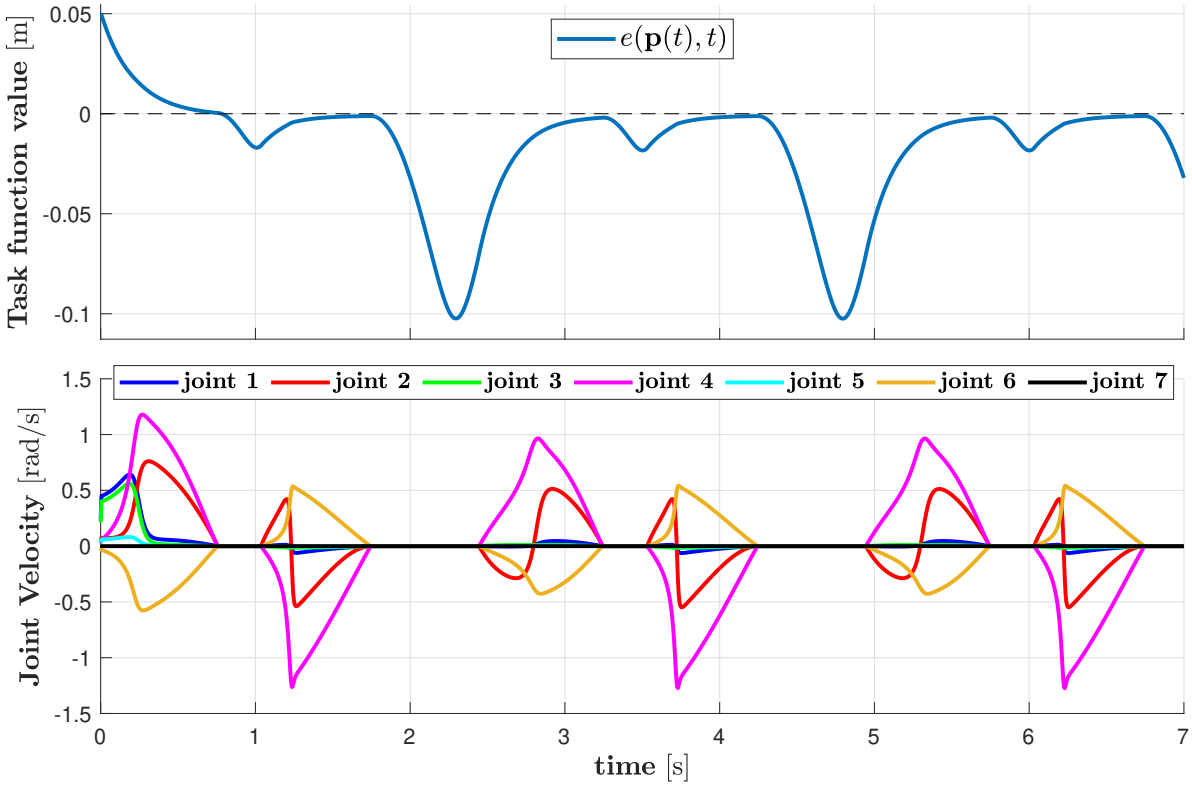


Figure 3.5: Task function value (top) and joint velocity (bottom) obtained in the simulation of Sect. 3.4.1-A.

of the reference point P_d varies on a linear path with sinusoidal velocity profile, namely

$$x_d(t) = x_{d,0} + A_{P_d} \sin\left(\frac{2\pi t}{T_d}\right)$$
$$y_d(t) = y_{d,0} + A_{P_d} \sin\left(\frac{2\pi t}{T_d}\right),$$
$$z_d(t) = z_{d,0} + A_{P_d} \sin\left(\frac{2\pi t}{T_d}\right)$$

with $x_{d,0} = y_{d,0} = z_{d,0} = 0.5\,\text{m}$, $A_{P_d} = 0.1\,\text{m}$ and $T_d = 2.5\,\text{s}$. The maximum distance d_d also varies with a sinusoidal law as

$$d_d(t) = d_{d,0} + A_{d_d} \sin\left(\frac{2\pi t}{T_d} + \frac{\pi}{2}\right),$$

with $d_{d,0} = 0.1\,\text{m}$ and $A_{d_d} = 0.05\,\text{m}$.

Figure 3.4 shows the obtained robot configuration for some instants of time. In each picture, the light blue sphere represents the position of P_d (center of the sphere) and the value of d_d (radius of the sphere) for the considered instant of time. Therefore, for the constraint (3.39) to be satisfied, the tip of the robot tool should always be inside the sphere. However, this condition is not satisfied at the initial time $t = 0$ s. Nevertheless, the specification of the constraint upper velocity bound as in (3.17) allows the robot to

bring the tool tip inside the cone at $t = 0.8$ s. This can also be seen in the top plot of Fig. 3.5. After the initial recovery phase, the constraint remains fulfilled ($e < 0$) for the remainder of the motion. The (exponential) recovery rate depends from the chosen value of k. Figure 3.5 additionally reports the obtained joint velocity commands. It can be noticed that joint motion is produced also when the constraint is satisfied. This happens every time the task function would approach the zero value faster than the provided exponential convergence rate, as a consequence of the choice (3.17).

B. Multi-robot, multi-PoC

The second illustrative example is a simulation involving multiple robots (Fig. 3.6). The first robot (robot 1) is a 17-DoFs mobile dual-arm system, composed of an omnidirectional mobile base and two KUKA LBR iiwa robots. Both arms present a conical tool as the one in Sect. 3.4.1-A. The two points at the tip of the cones are indicated as P_{11} and P_{12}. Their position can be expressed with respect to two PoC frames, $\boldsymbol{p}_{11}$ and $\boldsymbol{p}_{12}$, each fixed to the corresponding tool. As in Sect. 3.4.1-A, the PoC frames are conveniently placed at the tool tip. The second robot (robot 2) consists of a KUKA DKP 400 2-DoFs pan-tilt table. Here the center point of the table is of interest. This is indicated as P_2 and expressed as a function of the PoC frame $\boldsymbol{p}_2$, placed in P_2. The third robot (robot 3) is a KUKA AGILUS 6-DoFs manipulator, also equipped with a conical tool. The point at the tip of the tool is indicated as P_3 and expressed with respect to the PoC frame $\boldsymbol{p}_3$, placed in P_3.

The task is expressed by the following set of equality constraints:

$$\boldsymbol{e}(\boldsymbol{q}_2, \boldsymbol{p}_{11}, \boldsymbol{p}_{12}, \boldsymbol{p}_2, \boldsymbol{p}_3, t) = \begin{bmatrix} e_1(\boldsymbol{q}_2, t) \\ e_2(\boldsymbol{p}_{11}, \boldsymbol{p}_{12}, \boldsymbol{p}_2) \\ e_3(\boldsymbol{p}_{12}, \boldsymbol{p}_3, \boldsymbol{p}_2) \end{bmatrix} = \mathbf{0}, \tag{3.42}$$

with

$$\begin{aligned} e_1 &= q_{2,1} - q_{2,1,d}(t) \\ e_2 &= \|P_{11}(\boldsymbol{p}_{11}) - P_2(\boldsymbol{p}_2)\|_2 - \|P_{12}(\boldsymbol{p}_{12}) - P_2(\boldsymbol{p}_2)\|_2, \\ e_3 &= \|P_{12}(\boldsymbol{p}_{12}) - P_2(\boldsymbol{p}_2)\|_2 - \|P_3(\boldsymbol{p}_3) - P_2(\boldsymbol{p}_2)\|_2 \end{aligned} \tag{3.43}$$

where $q_{2,1}$ is the first element of $\boldsymbol{q}_2$ and $q_{2,1,d}$ a desired trajectory for the corresponding joint. In other words, it is required that the tool points P_{11}, P_{12}, and P_3 have the same distance from the pan-tilt table center point, P_2, whose position varies over time as a result of the desired trajectory assigned to the first joint of robot 2. This is chosen as

$$q_{2,1,d}(t) = \frac{\pi}{4}\cos(t). \tag{3.44}$$

Figure 3.6 gives a complete overview of all the variables involved in the task specification. Differentiating the task function in (3.42) – (3.43) with respect to the joint vector $\boldsymbol{q} =$

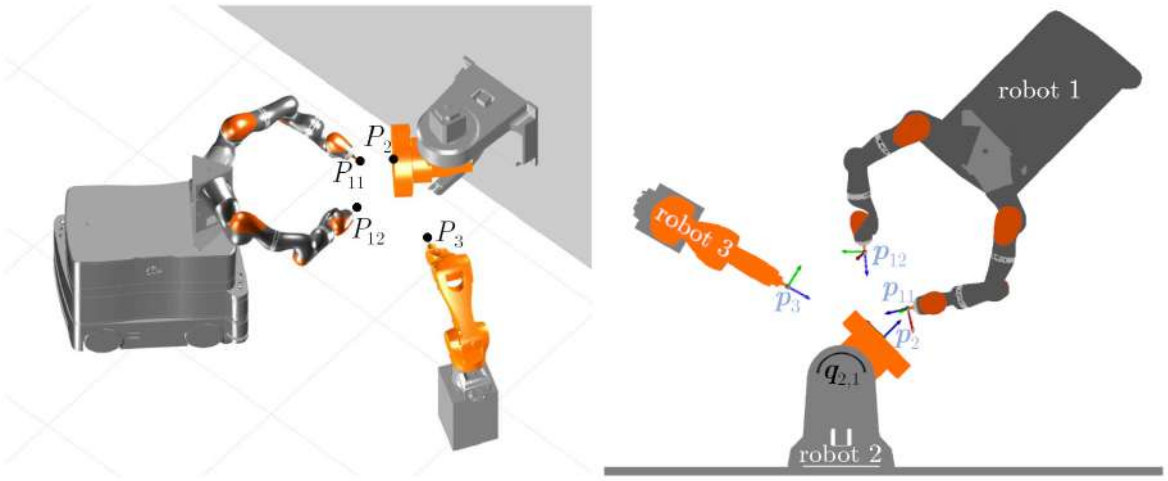


Figure 3.6: Overview of robots, PoC frames and variables involved in the task specification of Sect. 3.4.1-B.

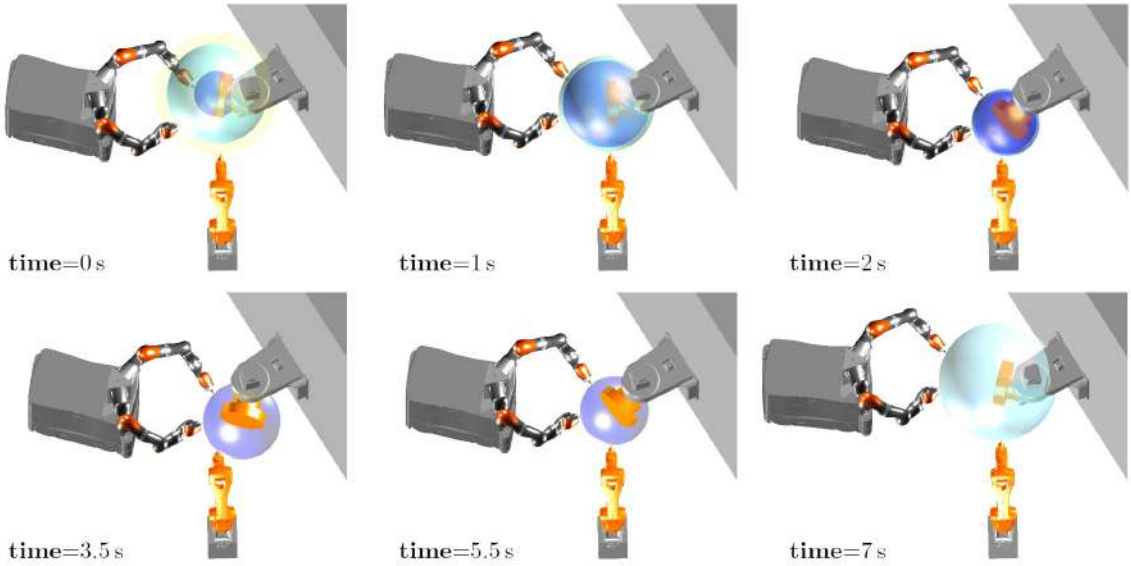


Figure 3.7: Motion sequence of the robots involved in the simulation of Sect. 3.4.1-B.

$\begin{bmatrix} \boldsymbol{q}_1^T & \boldsymbol{q}_2^T & \boldsymbol{q}_3^T \end{bmatrix}^T$ returns the matrix $\boldsymbol{E}_q = \begin{bmatrix} \boldsymbol{E}_{q_1} & \boldsymbol{E}_{q_2} & \boldsymbol{E}_{q_3} \end{bmatrix}$, where

$$\boldsymbol{E}_{q1} = \boldsymbol{0}, \quad \boldsymbol{E}_{q2} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{E}_{q3} = \boldsymbol{0}.$$

The matrix $\boldsymbol{E}_p = \begin{bmatrix} \boldsymbol{E}_{p_{11}} & \boldsymbol{E}_{p_{12}} & \boldsymbol{E}_{p_2} & \boldsymbol{E}_{p_3} \end{bmatrix}$ presents instead all zero elements on the first row, while the remaining rows can be easily computed by differentiating the distance function as in Sect. 3.4.1-A. For the sake of brevity, only the expression of $\boldsymbol{E}_{p_{11}}$ and $\boldsymbol{E}_{p_{12}}$

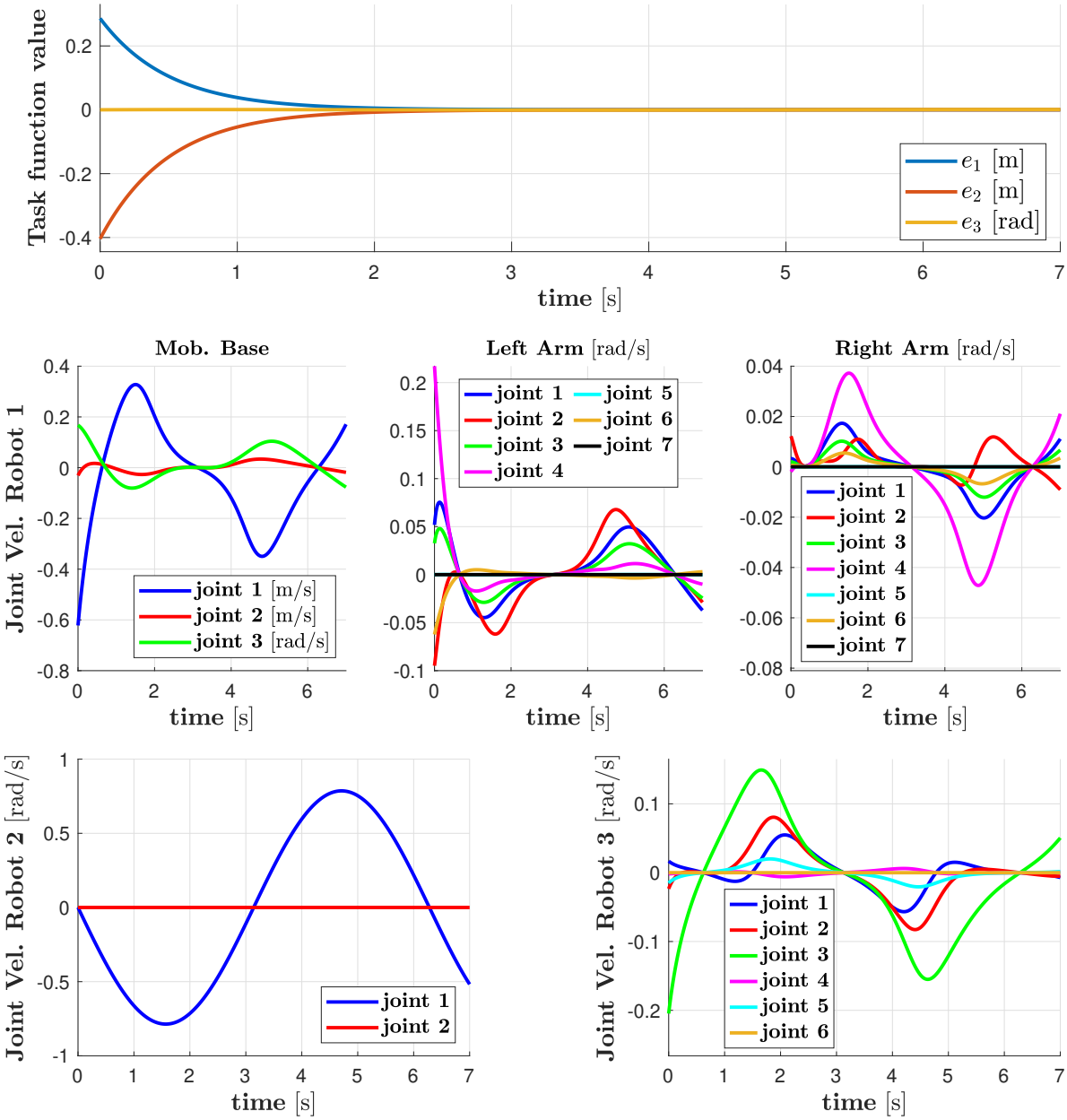


Figure 3.8: Task function value and joint velocities obtained in the simulation of Sect. 3.4.1-B.

are reported

$$\boldsymbol{E}_{p_{11}} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{0} \\ \frac{x_{p_{11}}}{\|P_{11}-P_2\|_2} & \frac{y_{p_{11}}}{\|P_{11}-P_2\|_2} & \frac{z_{p_{11}}}{\|P_{11}-P_2\|_2} & \mathbf{0} \\ 0 & 0 & 0 & \mathbf{0} \end{bmatrix}$$

$$\boldsymbol{E}_{p_{12}} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{0} \\ \frac{-x_{p_{12}}}{\|P_{12}-P_2\|_2} & \frac{-y_{p_{12}}}{\|P_{12}-P_2\|_2} & \frac{-z_{p_{12}}}{\|P_{12}-P_2\|_2} & \mathbf{0} \\ \frac{x_{p_{12}}}{\|P_{12}-P_2\|_2} & \frac{y_{p_{12}}}{\|P_{12}-P_2\|_2} & \frac{z_{p_{12}}}{\|P_{12}-P_2\|_2} & \mathbf{0} \end{bmatrix}.$$

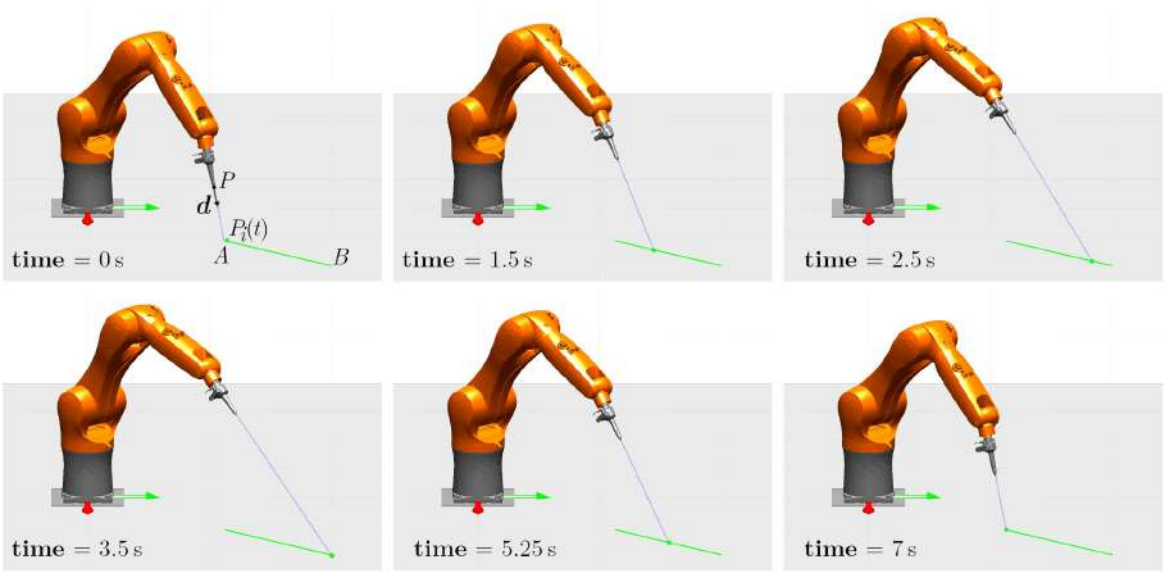


Figure 3.9: Motion sequence of the KUKA AGILUS robot in the first simulation of Sect. 3.4.2-A.

The vector $\boldsymbol{e}_t$ is finally obtained differentiating (3.43) with respect to the time. This operation yields

$$\boldsymbol{e}_t = \begin{bmatrix} \dot{q}_{2,1,d} \\ 0 \\ 0 \end{bmatrix}.$$

The simulation is performed using a matrix $\boldsymbol{H} = \boldsymbol{I}$. The constraint reference velocity is defined as in (3.13) with $\boldsymbol{K} = k\boldsymbol{I}$, $k = 2$. The total simulation time is 7 s, with a cycle time of 5 ms. Figure 3.7 shows the resulting sequence of motion. In each picture, the three spheres represent the distance of the points P_{11}, P_{12}, and P_3 from P_2, which is the center of all the spheres. At the initial time $t = 0$ s, the spheres have different radii, indicating that the constraints (3.42) – (3.43) are not satisfied. However, thanks to the specified constraint reference velocity the robots achieve fulfilment of the given constraints at about $t = 2$ s. This is indicated by the three spheres overlapping and becoming indistinguishable from this time on. At the same time, the first joint of the pan-tilt table moves according to the specification (3.44). The obtained results can be analyzed in Fig. 3.8, which reports the trend of the task function and the computed joint velocities for all the involved robots.

3.4.2 Case Studies

A. Laser tracing

For the laser tracing case study, the application setup is shown in Fig. 3.9. The task involves a KUKA AGILUS 6-DoFs manipulator and consists of tracing the segment $\overline{AB}$

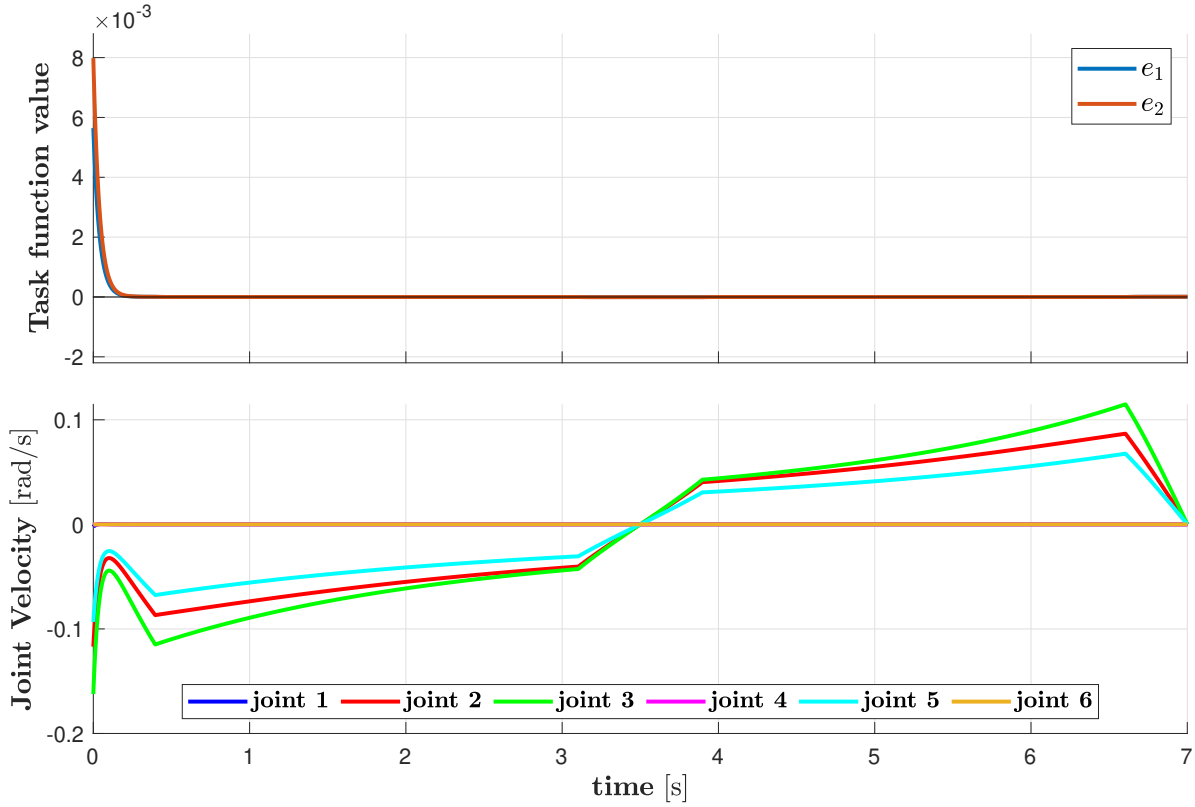


Figure 3.10: Task function value (top) and joint velocity (bottom) of the KUKA AGILUS robot obtained in in the first simulation of Sect. 3.4.2-A.

from A to B and back. The entire tracing should happen in a given time of 7 s, following trapezoidal velocity profiles. As anticipated in Sect. 3.3.2, the task formulation involves one PoC frame $\boldsymbol{p}$ fixed to the laser device, and can be expressed as

$$\boldsymbol{e}(\boldsymbol{p}, t) = \mathbf{0}, \tag{3.45}$$

with

$$\boldsymbol{e}(\boldsymbol{p}, t) = \begin{bmatrix} e_1(\boldsymbol{p}, t) \\ e_2(\boldsymbol{p}, t) \end{bmatrix} = \text{null}(\boldsymbol{d}^T(\boldsymbol{p})) \cdot (P(\boldsymbol{p}) - P_d). \tag{3.46}$$

The simulation is performed using a matrix $\boldsymbol{H} = \boldsymbol{I}$ and a constraint reference velocity defined as in (3.13), with a gain matrix $\boldsymbol{K} = k\boldsymbol{I}$, $k = 25$. The simulation cycle time is 1 ms. The evolution of the task function and the obtained joint velocities can be analyzed in Fig. 3.10.

A second laser tracing simulation, this time involving a KUKA LBR iiwa 7-DoFs robot, is introduced in Fig. 3.11. The task consists of tracing three different segments with three laser devices, all mounted at the robot flange. In this particular scenario, the identification of a suitable sequence of desired Cartesian frames for the end effector would require a huge engineering effort, making this application practically impossible to program with classic methods. Nevertheless, the proposed framework allows (with a single PoC frame fixed to the laser devices) to easily write the constraint set describing

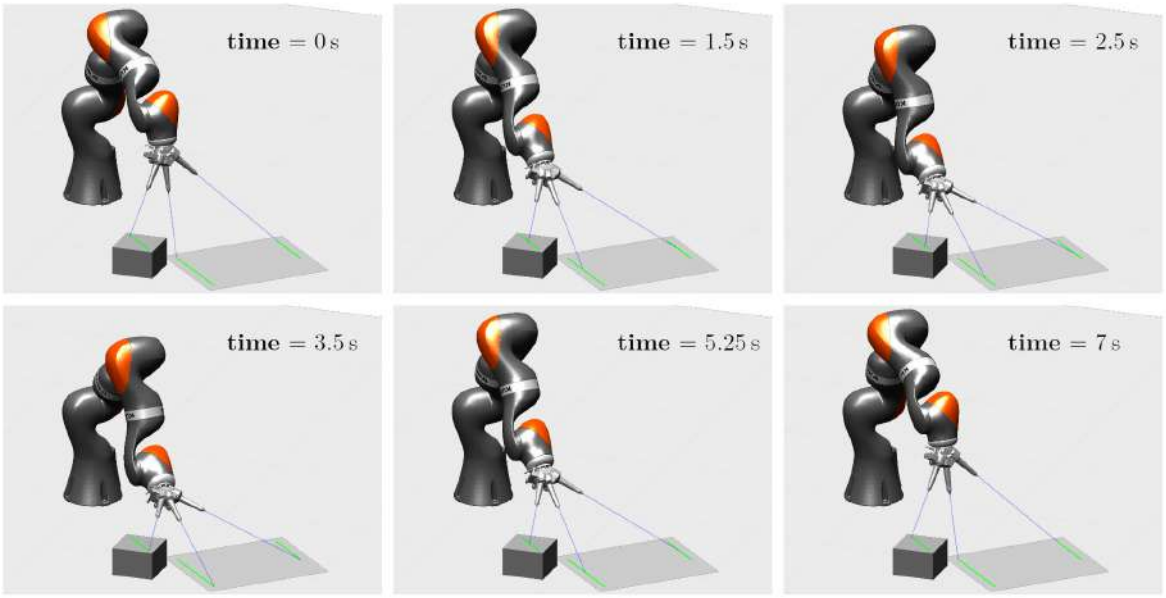


Figure 3.11: Motion sequence of the KUKA LBR iiwa robot in the second simulation of Sect. 3.4.2-A.

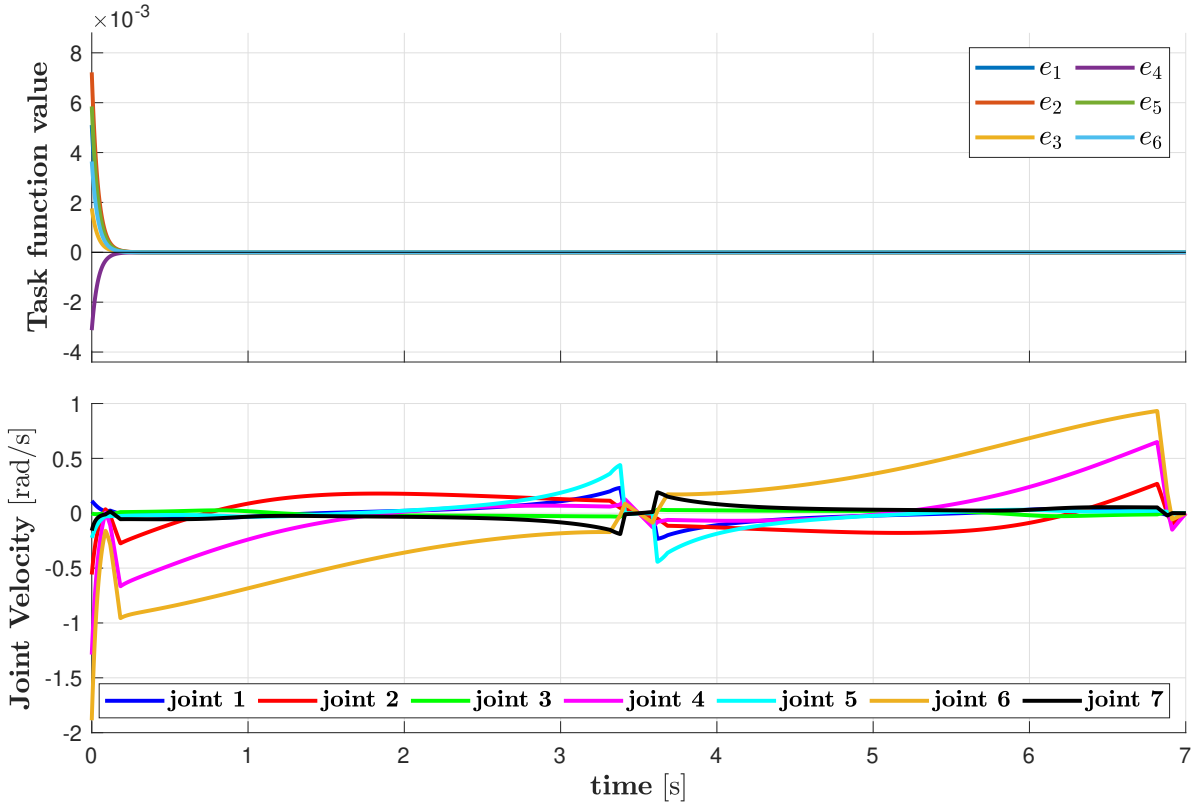


Figure 3.12: Task function value (top) and joint velocity (bottom) of the KUKA LBR iiwa robot in the second simulation of Sect. 3.4.2-A.

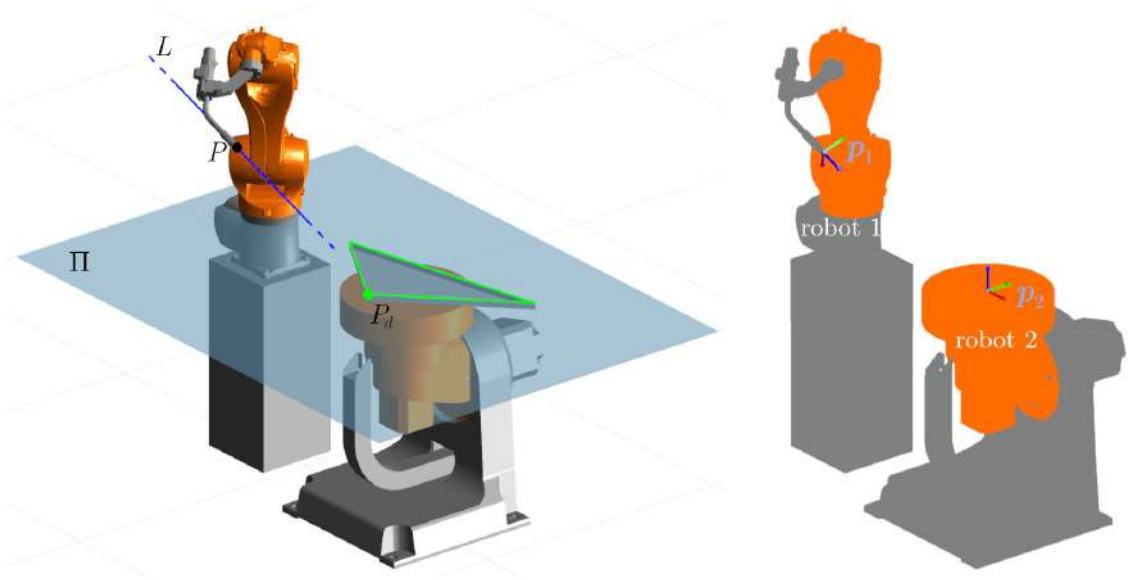


Figure 3.13: Overview of robots, PoC frames and geometric entities involved in the task specification of Sect. 3.4.2-B.

the task. This can be simply derived by extending (3.45)–(3.46). All the parameters remain unchanged from the previous simulation. The trend of the task function value and the obtained joint velocities for this simulation are reported in Fig. 3.12.

B. Multi-robot welding

For the welding case study, a multi-robot application is considered. The welding torch is attached to the flange of a KUKA AGILUS 6-DoFs robot, while the parts to weld are mounted on a KUKA DKP 400 2-DoFs pan-tilt table. Figure 3.13 shows the application setup, as well as an overview of all the geometric entities and PoC frames involved in the task formulation. In this particular case, the tip of the welding torch, indicated by P, is required to track a time-varying target point, P_d, which moves on a sequence of desired paths (green segments) with trapezoidal velocity profiles. As presented in Sect. 2.2.2, an additional constraint on the angle between the welding torch main axis, L, and the working plane, Π, exists. In this case, the working plane is considered as a virtual plane fixed to robot 2, which embeds the target paths. Thus, the task specification (2.4) can be formulated as

$$\begin{cases} \boldsymbol{e}_{eq,1-3}(\boldsymbol{p}_1, \boldsymbol{p}_2) = 0 \\ e_{iq,4}(\boldsymbol{p}_1, \boldsymbol{p}_2) \geq 0 \end{cases},$$

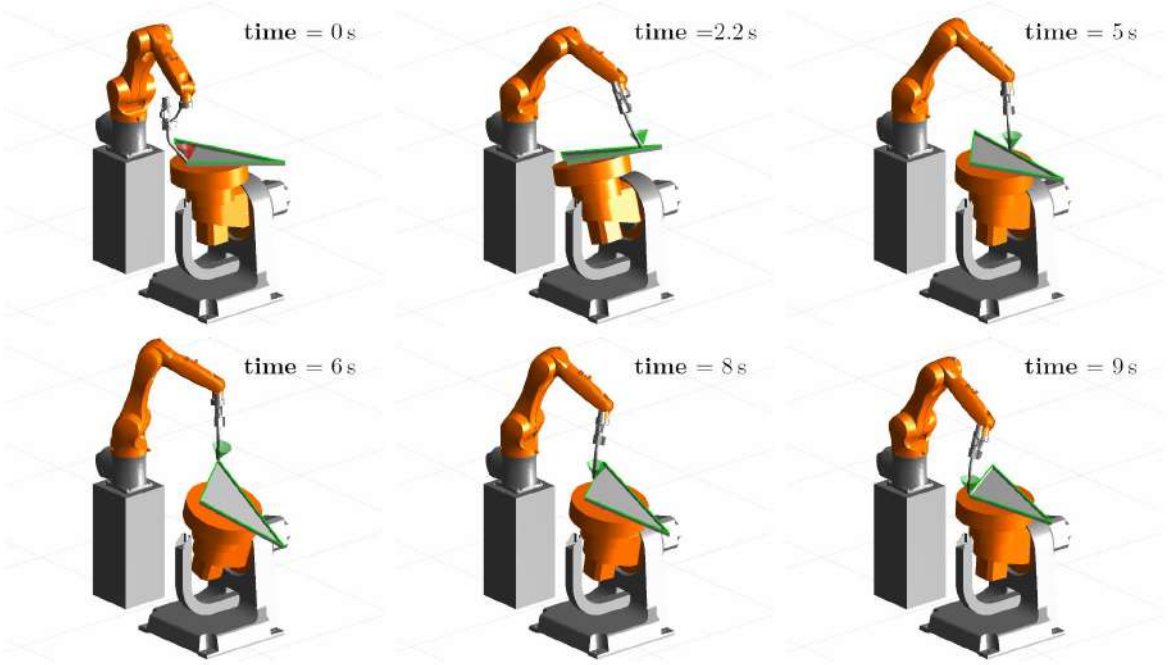


Figure 3.14: Motion sequence of the robots involved in the simulation of Sect. 3.4.2-B.

with

$$\boldsymbol{e}_{eq,1-3} = \begin{bmatrix} e_{eq,1} \\ e_{eq,2} \\ e_{eq,3} \end{bmatrix} = P - P_d \quad ,$$
$$e_{iq,4} = \text{asin}(\boldsymbol{d}_L \cdot \boldsymbol{n}_\Pi) - \alpha_{min}$$

where $\boldsymbol{d}_L$ and $\boldsymbol{n}_\Pi$ are defined as in Sect. 2.2.2.

For the simulation of this section, it is $\alpha_{min} = \pi/3\,\text{rad}$. Furthermore, constraint reference velocities are generated according to (3.13), with a gain $k_{eq} = 5$. Similarly, the velocity upper bound for the inequality constraint on $e_{iq,4}$ are generated according to (3.17), with a gain $k_{iq} = 3$. The total simulation time is 9 s, with a cycle time of 5 ms. Finally, the optimization problem uses $\boldsymbol{H} = \boldsymbol{I}$.

Figure 3.14 reports the obtained sequence of motion for the two robots. In each picture, the cone with apex in P_d and apex angle $2(\pi/2 - \alpha_{min})$ helps visualizing whether the constraint on the angle between L and Π is fulfilled. As an additional visual support, the cone is colored in green when the condition on the angle is met, in red otherwise. At the initial time $t = 0$, the constraint is not satisfied. However, thanks to the definition of a constraint velocity upper bound as in (3.17), the robots are able to recover the initial gap. Indeed, they start fulfilling the angle constraint at about 2.2 s and keep respecting it throughout the rest of the motion. This can also be seen in the top plot of Fig. 3.15. After an initial recovery phase, all the constraint remains fulfilled for the remainder of the motion. Figure 3.15 additionally reports the obtained joint velocity for the two robots.

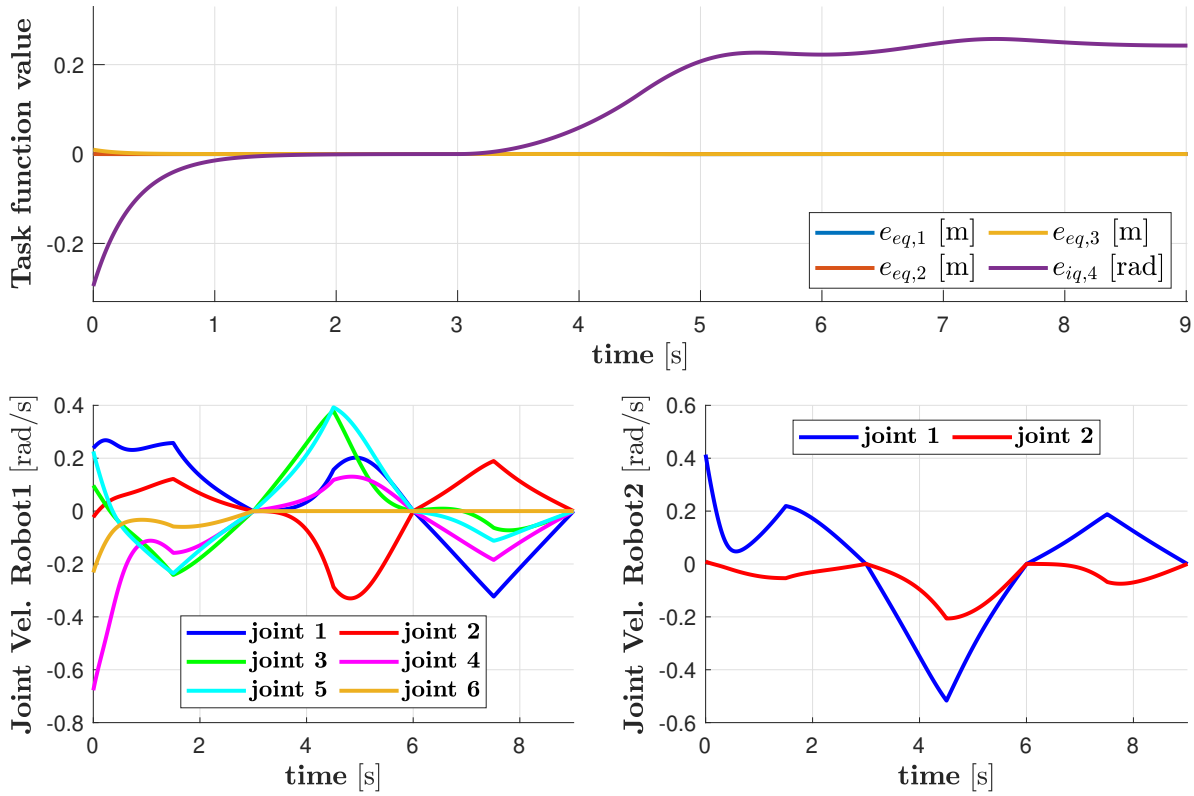


Figure 3.15: Task function value (top) and joint velocities (bottom) obtained in the simulation of Sect. 3.4.2-B.

C. Deburring

For the third case study, the deburring tool from Sect. 2.2.3 is attached to the flange of a KUKA LBR iiwa 7-DoFs robot, while the workpiece is fixed to the environment. Figure 3.16 shows the application setup, as well as an overview of the main geometric entities and PoC frames involved in the task formulation according to the specification (2.6). This can be expressed by the following set of constraints:

$$\begin{cases} \boldsymbol{e}_{eq,1-2} = 0 \\ e_{eq,3} = 0 \\ e_{eq,4} = 0 \\ e_{iq,1} \geq 0 \\ e_{iq,2} \leq 0 \end{cases}$$

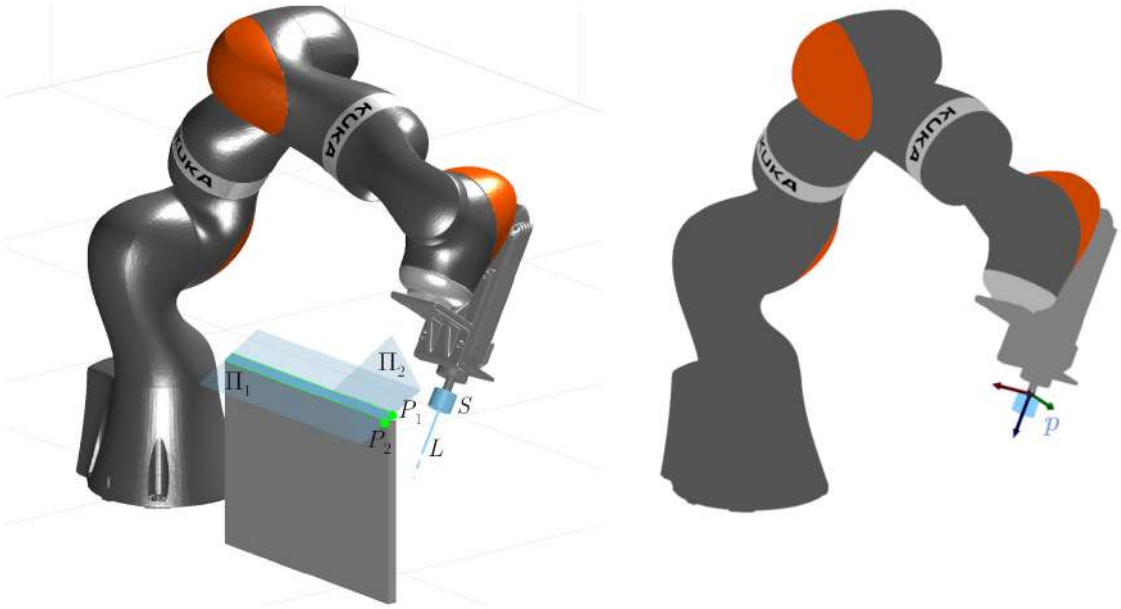


Figure 3.16: Overview of robots, PoC frames and geometric entities involved in the task specification of Sect. 3.4.2-C.

with

$$
\begin{aligned}
\boldsymbol{e}_{eq,1-2} &= \begin{bmatrix} e_{eq,1} \\ e_{eq,2} \end{bmatrix} = \mathrm{null}(\boldsymbol{d}_L^T(\boldsymbol{p})) \cdot \boldsymbol{d}_{\overline{P_1 P_2}} \\
e_{eq,3} &= \boldsymbol{n}_{\Pi_1} \cdot (P(\boldsymbol{p}) - P_{\Pi_1}) - r \\
e_{eq,4} &= \boldsymbol{n}_{\Pi_2} \cdot (P(\boldsymbol{p}) - P_{\Pi_2}) \\
e_{iq,1} &= \boldsymbol{n}_{\Pi_B}(\boldsymbol{p}) \cdot (P_m - P_{\Pi_B}((\boldsymbol{p}))) - \frac{l}{2} \\
e_{iq,2} &= \boldsymbol{n}_{\Pi_B}(\boldsymbol{p}) \cdot (P_m - P_{\Pi_B}((\boldsymbol{p}))) - h + \frac{l}{2}
\end{aligned}
\quad . \tag{3.47}
$$

All the quantities in (3.47) are defined as in Sect. 2.2.3.

The simulation is carried out with $\boldsymbol{H} = \boldsymbol{I}$, whereas the total simulation time is 7 s, with a cycle time of 5 ms. Constraint reference velocities and velocity lower/upper bounds are again defined in order to ensure exponential behavior of the task function, with scalar gains $k_{eq} = k_{iq} = 2$.

Figure 3.17 shows the generated robot motion. Once again, the specified constraints are not fulfilled at the initial time $t = 0$, but the robot accomplishes an (exponential) recovery of the initial error. This is also visible in the top plots of Fig. 3.18, where the evolution of all the task function components is reported, along with the generated joint velocities.

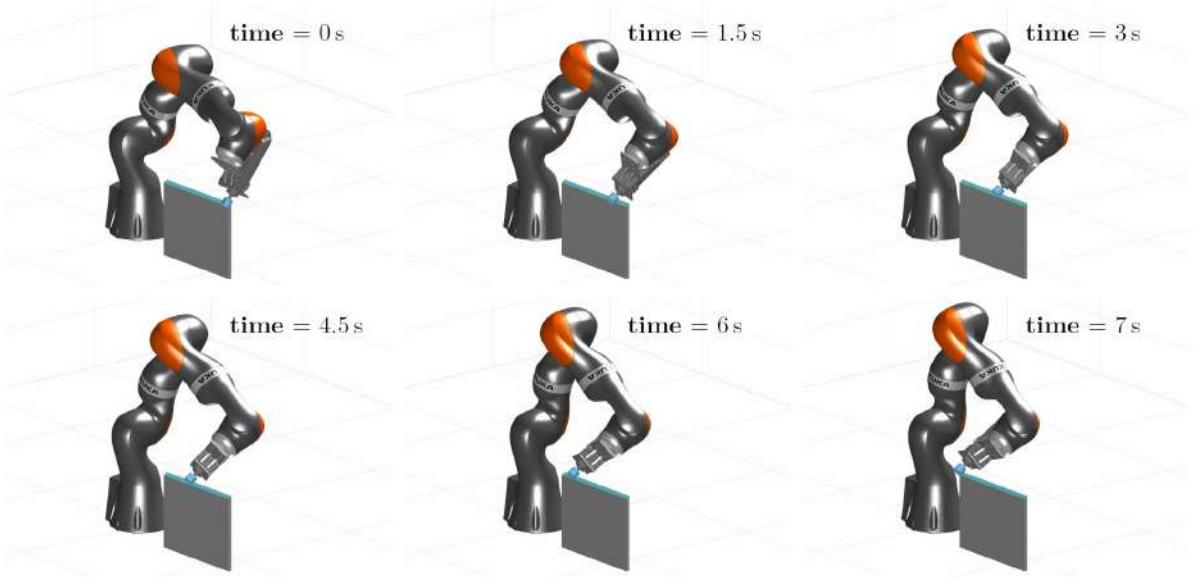


Figure 3.17: Motion sequence of the KUKA LBR iiwa robot in the simulation of Sect. 3.4.2-C.

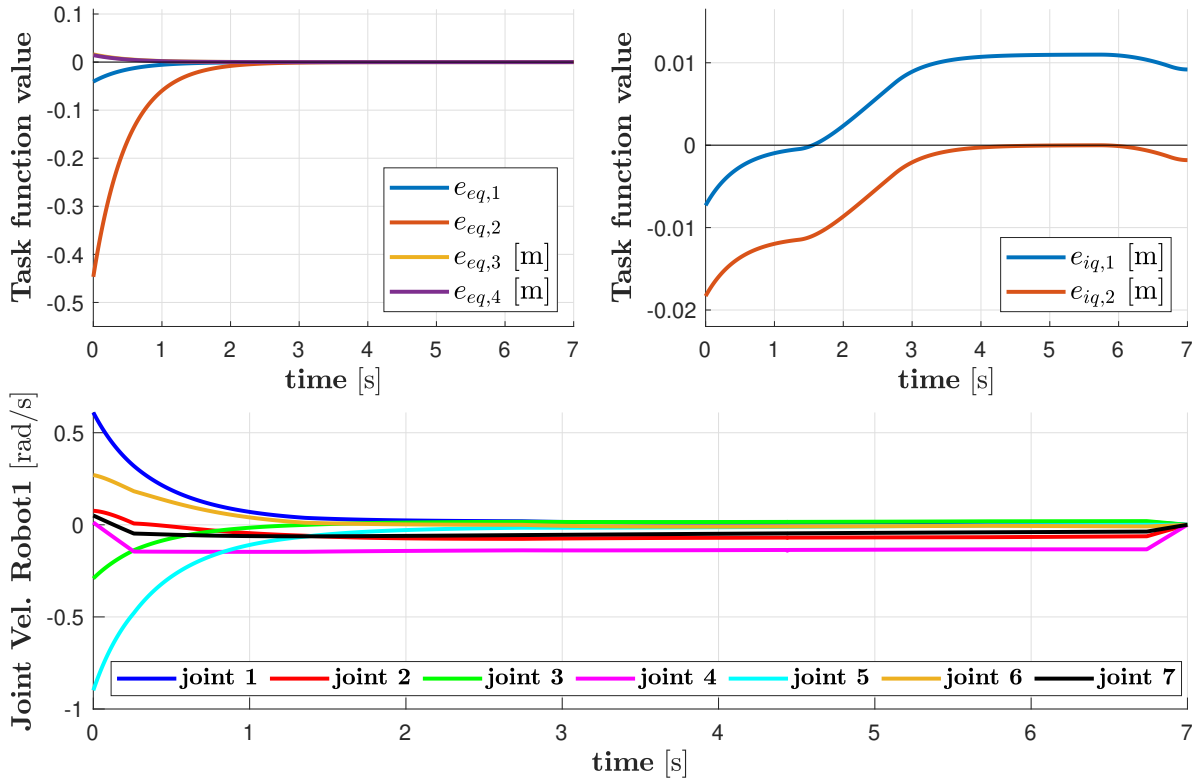


Figure 3.18: Task function value (top) and joint velocities (bottom) obtained in the simulation of Sect. 3.4.2-C.

3.5 Discussion

This chapter has presented a general framework for the constraint-based programming of redundant robots. The proposed formulation supports the intuitive task specification introduced in Chap. 2 and presents a significant versatility. Indeed, it can handle single and multi-robot applications in a unified fashion, as well as handling multiples PoCs for each robot. The generality of the framework is enhanced by the fact that classic programming based on Cartesian frames can be obtained as special case, as it is for joint motion commands. Furthermore, the proposed methodology has also be shown to overcome some limitations of existing constraint-based programming frameworks.

The effectiveness of the framework has been proved through a set of simulations, in which complex tasks are executed with very limited programming efforts, additionally optimizing redundant DoFs. Particularly significant examples are the multi-laser tracing application from Sect. 3.4.2-A and the multi-robot welding from Sect. 3.4.2-B. Programming such tasks with Cartesian frames would require a huge engineering effort, and eventually lead to suboptimal solutions.

Additional work on the framework could focus on how to properly include force constraints in the proposed formalism for task description. The explicit consideration of uncertainties coming from the environment and/or sensor data is also a missing feature in the current formulation. In any case, sensitivity to possible measurement noise affecting task variables can be controlled by acting on the gain in (3.13) and (3.17). Finally, effective handling of possible conflicting constraints or strategies to avoid that such conflicts arise represent other open research fields.

Chapter 4

Redundancy Resolution

4.1 Introduction

Redundancy resolution is defined as the problem of selecting a specific robot posture among a set of (likely infinite) possible solutions accomplishing a given task. This problem, extensively studied in the last decades, becomes especially relevant when dealing with robots with a large number of DoFs, or in the case of collaborative multi-robot applications, as also remarked in Chap. 3.

Redundancy allows for more versatility in performing an assigned task and for the simultaneous specification of multiple objectives. However, proper strategies are required to compute suitable joint motions and achieve effective exploitation of the redundant DoFs.

As seen in the previous chapters, tasks can consist of equality or inequality constraints. Additionally, they might also present different priority based on their respective relevance. For example, safety-related tasks, such as avoiding collision with the environment, may be considered of utmost importance and therefore assigned the highest priority. Furthermore, the robots might be required to autonomously operate in unstructured environments. For this reason, redundancy resolution algorithms are often designed to find solutions online and to include robust and predictable behavior in case the assigned tasks become unfeasible altogether due to some change in the environment. Finally, when considering robots with a high number of DoFs, also the computational efficiency becomes considerably important.

In the previous chapter, the formulation of tasks for redundant robots has led to the derivation of constrained optimization problems, which should be solved to obtain an optimal robot posture at each instant of time. All the optimization problems presented in Sect. 3.3.1 and 3.3.4 share a similar structure, consisting of a quadratic cost function and linear (equality and inequality) constraints. Thus, this chapter focuses on general methods to solve such class of constrained optimization problems in the context of robot motion control. Additionally, some critical aspects originating from the discrete-time implementation of the solver are analyzed.

4.2 Related Work

Robots with a large number of DoFs have seen an increasing demand in industrial applications and service robotics in recent years. From industrial manipulators (Scheurer, Fiore, et al. 2016) up to humanoid robots (Escande, Mansard, et al. 2014) and unmanned vehicles (Antonelli 2014; Baizid, Giglio, et al. 2015), robots are more and more often designed to be highly redundant, i.e., to have significantly more DoFs than are needed to perform a given task.

Redundancy resolution methods have been extensively studied for decades. Using the first-order differential kinematic model of the robot, Whitney (1969) first introduced a method for solving a single-task problem based on a minimum-norm solution. A task priority approach exploiting null space projection was then presented by Maciejewski and Klein (1985) for two tasks and later extended to a generic number of tasks by Siciliano and J.-J. Slotine (1991). Similar approaches have been developed at acceleration level, using the second-order differential kinematic model (De Luca, Oriolo, et al. 1992). The exploitation of the robot dynamical model has instead led to the Operation Space Formulation (Khatib 1983) and its extension to prioritized tasks (Dietrich and Ott 2019; Ott, Dietrich, et al. 2015; Sentis and Khatib 2004).

In all the above-mentioned contributions, the tasks assigned to the robotic system consist of a set of equality constraints. However, several tasks may be naturally described as inequality constraints. Handling of inequality constraints has been tackled in different ways. Most methods are based on pseudoinversion of task Jacobian matrices and null space projections, and here referred to as *analytical*. An early approach in this class of methods is represented by the Gradient-Based Projection (GBP) method (Liegeois et al. 1977), in which inequalities are converted into a cost function. Robot redundancy is then exploited to minimize such cost, in the attempt of keeping the task variables within their allowed range. However, the optimization is performed at a lower priority level, meaning that fulfillment of the inequality constraints is not guaranteed. Another simple, yet effective, approach is to perform task scaling (Antonelli, Chiaverini, et al. 2003; Chiacchio and Chiaverini 1995; Hollerbach 1983), i.e., to reduce the speed (or acceleration) required by equality constraint task commands to recover feasibility with respect to one or more violated inequality constraints. One benefit of the task scaling method is that the directions of the equality constraints task commands are preserved. The execution of such commands is only extended in time, which is in many cases acceptable. Furthermore, task scaling can easily be applied to the case of prioritized tasks and provides the robot with a predictable behavior in case of conflict between equality and inequality constraints. On the other hand, the mere use of this technique implies slower task execution every time an inequality constraint would be violated. Before resorting to such a drastic measure, one should first verify that no alternative joint motions exist, which can ensure the satisfaction of both equality and inequality constraints.

Another widely adopted approach is to convert inequality constraints into equivalent equality constraints. To avoid overconstrained motions, these additional equality constraints can only be added to the set of prioritized tasks (often referred to as *Stack of Tasks (SoT)*) with the lowest priority. Thus, the satisfaction of the original inequalities

is not guaranteed (Khatib 1986). Alternatively, the satisfaction of inequality constraints can be monitored, and the equivalent equality constraints inserted at the desired priority level only when task inequality bounds are approached. A classic method in torque-based redundancy resolution is to resort to artificial potential fields (Sentis and Khatib 2005), in which virtual forces are used to push task variables away from the bounds of their allowed range. The effectiveness of this method, however, strongly depends on the parameters that regulate the activation and the intensity of the repulsive virtual force (Muñoz Osorio, Fiore, et al. 2018). Using a similar approach, methods for handling inequality constraints in velocity-based schemes were developed by Mansard, Khatib, et al. (2009) and Moe, Antonelli, et al. (2016). Also in this case, special parameters are introduced to define activation functions and thresholds, as well as to shape the functions that force task variables to stay in a *safe* range. Furthermore, handling the continuous activation/deactivation of the additional equality constraints may require parallel computation of several possible solutions (Di Lillo, Pierri, et al. 2021; Moe, Antonelli, et al. 2016). Additionally, online modifications to the SoT typically produce discontinuities in the solution (Sentis and Khatib 2005). Besides requiring more parameterization, methods to ensure smooth insertion to/removal from the SoT typically fail at respecting the strict priority among the tasks during transitions (Mansard, Khatib, et al. 2009) or imply a significant increase of the computational cost (J. Lee, Mansard, et al. 2012; Liu, Tan, et al. 2016).

Given the above-mentioned limitations of analytical approaches in dealing with inequality constraints, numerical methods based on Hierarchical Quadratic Programming (HQP) have been more intensively investigated in recent years (Aertbeliën and De Schutter 2014; Escande, Mansard, et al. 2014; Hoffman, Laurenzi, et al. 2018; Kanoun, Lamiraux, et al. 2011; Liu, Tan, et al. 2016; Quiroz-Omaña and Adorno 2019). The main idea, first introduced by Kanoun, Lamiraux, et al. (2011), is to solve a cascade of Quadratic Programming (QP) problems, one for each level of priority. Both equality and inequality constraints can be easily specified in the context of QP at each priority level. Although effective in fulfilling the given constraints, the technique suffered from a high computational cost, when compared to analytical solutions. This effect has been later mitigated by formalizing the multiple QP problems in a single one and using complete orthogonal decomposition to obtain the null spaces of the prioritized tasks (Escande, Mansard, et al. 2014; Liu, Tan, et al. 2016). Another drawback of the mentioned solutions based on HQP is that they do not feature a defined behavior in case a task is not feasible. In fact, unfeasible tasks are normally handled through the introduction of slack variables and/or by relaxing the constraints in a least square sense. However, such strategies do not deliver a predictable robot behavior. On the other hand, in many robot applications (especially in the industrial field) it is acceptable to just extend the execution of the equality constraints over time, while preserving the task directions. In all these cases, task scaling seems a more appropriate solution to recover feasibility.

The Saturation in the Null Space (SNS) algorithm (Flacco, De Luca, and Khatib 2015) for kinematic control provides a promising link between analytical methods and HQP, while featuring a task scaling technique. Task prioritization uses null space projections as in analytical methods. At each level of priority, inequality constraints are monitored and, if necessary, converted into equality constraints in a similar way as in

(active-set) QP problems. Finally, a task scaling strategy is applied in all cases where it was not possible to find a feasible solution. Although special variants of this algorithm tackle additional important aspects like the optimality of the solution and the efficiency of the computation, several points still remain unsolved. First, only joint space inequality constraints are handled, which are always treated with the highest priority. Thus, operational inequality constraints, defined in task space and/or having lower priority are not directly addressed. Moreover, the optimization process presents a simple cost function, based on the idea of minimizing the pure joint velocity norm. Similarly to the work by Escande, Mansard, et al. (2014), also the machinery used to speed-up the computation (Flacco and De Luca 2013a) strongly relies on the consideration of such cost function. However, joint velocity minimization has been shown to have drawbacks for robots whose joint coordinates present mixed units, which is often the case when considering robots with a large number of DoFs, e.g., mobile manipulators (Lachner, Schettino, et al. 2020). This aspect is also essential in the context of energy-aware motion generation (Lachner, Allmendinger, et al. 2021). Therefore, a proper redundancy resolution algorithm should provide the possibility to select more appropriate metrics.

4.3 General Hierarchical Framework

Most of the solutions mentioned in Sect. 4.2 utilize velocity-based redundancy resolution, mainly because of its mathematical simplicity. However, moving to second-order algorithms, i.e., acceleration or torque level, offers some advantages, e.g., enabling the inequality constraints to include maximum/minimum accelerations, and also improving the noise, vibration and harshness (NVH) behavior of the robot. Torque-based approaches additionally offer compliance, allowing the robot to safely handle physical contact. On the other hand, these methods suffer from dynamic model uncertainties as pointed out by Di Lillo, Antonelli, et al. (2021) and evidenced by some of the experimental results presented in Sect. 4.3.5.

Velocity and acceleration-based models describe the robot using a kinematic equation at velocity and acceleration level, respectively; the output of the control algorithm is then a vector of joint velocities or accelerations, which are typically integrated to compute joint positions that are then given as input to underlying joint position controllers. On the other hand, a torque-based approach uses the dynamic model of the robot, and outputs a vector of joint torques. These are then directly sent to joint actuators and an actual control loop is closed using measured joint position and velocity.

This section presents a general framework for hierarchical redundancy resolution under arbitrary (equality and inequality) constraints. A general formulation of the redundancy resolution problem is proposed, which allows to express all of the aforementioned redundancy resolution schemes (velocity-, acceleration- and torque-based) in a unified form. The proposed generalized redundancy resolution problem is further developed to specifically include arbitrary equality and inequality constraints on every priority level. Furthermore, an additional input is considered, which allows for the specification of optimization criteria to better manage possible residual redundant DoFs. Having a unified description of the redundancy resolution problem also allows for the design of a sin-

gle solver. Here, a novel algorithm is introduced, named *extended SNS (eSNS)*, which builds on the SNS methodology extending it in many significant aspects. Being capable of solving the proposed generalized redundancy resolution problem, the algorithm can indistinctly handle velocity-, acceleration- or torque-based schemes. Moreover, arbitrary inequality constraints can be managed at each level of priority. Finally, arbitrary metrics and additional inputs are considered in the optimization process when computing a solution.

An analysis of the computational efficiency is also carried out, leading to a special variant of the algorithm, named *Fast-eSNS*. Another variant (*Opt-eSNS*) can be obtained as a result of the analysis on the optimality of the solution and is also introduced in this section. Finally, a novel shaping of the inequality constraint bounds is proposed, which allows treating position, velocity and acceleration limitations in a unified way. The parameterization of such shaping is consistent with the one typically used to define task commands in equality constraints. Thus, it is based on the desired behavior of dynamic systems and embeds a strong physical meaning. It is also shown that such shaping represents an extension of the solutions proposed by Flacco, De Luca, and Khatib (2012, 2015) and by Osorio, Allmendinger, et al. (2019).

4.3.1 Mathematical Background

To introduce the formalism and the mathematical background on redundancy resolution, some relevant findings from Chap. 3 needs to be briefly recalled.

In Sect. 3.3.1, the formulation of equality constraints ($\boldsymbol{e} = \boldsymbol{0}$) in the constraint-based programming framework has led to the following relation

$$\boldsymbol{J}_c \dot{\boldsymbol{q}} = \boldsymbol{r}_{v_c}, \tag{4.1}$$

expressing the equality constraint at the first-order differential level. In (4.1), $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ is the vector of joint velocities, $\boldsymbol{J}_c = (\boldsymbol{E}_q + \boldsymbol{E}_p \boldsymbol{J}) \in \mathbb{R}^{m \times n}$ indicates the constraint Jacobian matrix, and $\boldsymbol{r}_{v_c} = -\boldsymbol{e}_t \in \mathbb{R}^m$ is the constraint reference velocity, with $n \geq m$. Similar constraints have been obtained at acceleration-level, namely

$$\boldsymbol{J}_c \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c \dot{\boldsymbol{q}} = \boldsymbol{r}_{a_c}, \tag{4.2}$$

where $\boldsymbol{r}_{a_c} = -\frac{\partial \boldsymbol{e}_t}{\partial t}$ represents the constraint reference acceleration. Finally, for torque-controlled robots, the constraints (4.2) can be rewritten as

$$\boldsymbol{J}_c \boldsymbol{M}^{-1} \boldsymbol{\tau}' + \dot{\boldsymbol{J}}_c \dot{\boldsymbol{q}} = \boldsymbol{r}_{a_c}, \tag{4.3}$$

with $\boldsymbol{M}$ and $\boldsymbol{\tau}'$ defined as in Sect. 3.3.4). In addition to (4.3), constraints on the maximum joint driving torque could also be of interest for this class of robots. Such constraints can be expressed as:

$$\underline{\boldsymbol{\tau}}(t) \leq \boldsymbol{\tau}(t) \leq \overline{\boldsymbol{\tau}}(t). \tag{4.4}$$

Section 3.3.2 has also discussed how the proposed formalism naturally embeds classic inverse kinematics problems. In this case, it is $\boldsymbol{E}_q = \boldsymbol{0}$ and $\boldsymbol{E}_p = \boldsymbol{I} \in \mathbb{R}^6$. Thus, a

constraint Jacobian matrix $\boldsymbol{J}_c = \boldsymbol{J}$ is automatically returned. Analogously, a task could also consist of pure joint space commands, in which case it is $\boldsymbol{J}_c = \boldsymbol{E}_q = \boldsymbol{I} \in \mathbb{R}^n$. This special case is often considered to track joint space references or to handle joint limitations, i.e., to ensure that all joint positions (as well as velocities and accelerations) stay within the allowed mechanical ranges.

Another important aspect in Chap. 3 is how $\boldsymbol{r}_{v_c}$ and $\boldsymbol{r}_{a_c}$ can be modified to impose a desired evolution of the task function $\boldsymbol{e}$. In particular, it has been remarked how the choice

$$\boldsymbol{r}_{v_c} = -\boldsymbol{e}_t - \boldsymbol{K}\boldsymbol{e}, \quad \boldsymbol{K} > \boldsymbol{0} \tag{4.5}$$

imposes an evolution of the task function as a first-order linear system

$$\dot{\boldsymbol{e}} + \boldsymbol{K}\boldsymbol{e} = \boldsymbol{0},$$

with a convergence rate depending on the eigenvalues of $\boldsymbol{K}$. Analogously, choosing

$$\boldsymbol{r}_{a_c} = -\frac{\partial \boldsymbol{e}_t}{\partial t} - \boldsymbol{D}\dot{\boldsymbol{e}} - \boldsymbol{K}\boldsymbol{e}, \quad \boldsymbol{D}, \boldsymbol{K} \geq \boldsymbol{0} \tag{4.6}$$

imposes an evolution of the task function as a second-order linear system characterized by stiffness $\boldsymbol{K}$ and damping $\boldsymbol{D}$. Similar choices to (4.5) and (4.6) could also be used for the specification of constraint velocity and acceleration lower/upper bounds, in order to provide limit convergence rate in case of violation of inequality constraints. However, more sophisticated shaping of the bounds might be of interest, as discussed in the next section.

4.3.2 Shaping of constraint velocity and acceleration bounds

In the case of inequality constraints ($\boldsymbol{e} \leq \boldsymbol{0}$ or $\boldsymbol{e} \geq \boldsymbol{0}$), the constraint-based formulation from Sect. 3.3.1 leads to the following differential relation

$$\underline{\boldsymbol{r}}_{v_c} \leq \boldsymbol{J}_c\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{v_c}, \tag{4.7}$$

with $\underline{\boldsymbol{r}}_{v_c}$ and $\overline{\boldsymbol{r}}_{v_c}$ being the constraint velocity lower and upper bound, respectively. Without loss of generality, let focus on the shaping of the constraint velocity upper bound. The same consideration will anyway apply also for $\underline{\boldsymbol{r}}_{v_c}$. As anticipated in the previous section, the choice (4.5) can be applied also to inequality constraints. In particular, choosing

$$\overline{\boldsymbol{r}}_{v_c}(t) = -\boldsymbol{e}_t(t) - \boldsymbol{K}\boldsymbol{e}(t), \quad \boldsymbol{K} > \boldsymbol{0} \tag{4.8}$$

imposes an evolution of the task function that is no faster than a first-order linear system with a convergence rate depending on the eigenvalues of $\boldsymbol{K}$. However, maximum values of the constraint velocity, $\boldsymbol{v} = \boldsymbol{J}_c\dot{\boldsymbol{q}}$, might also be of interest. These can be easily integrated in (4.8). Let f^i denote the ith component of a vector $\boldsymbol{f}$, and let ${\boldsymbol{f}^i}^T$ indicate the row vector containing the ith row of a matrix $\boldsymbol{F}$. Then, indicating with $\overline{\boldsymbol{v}}(t)$ the vector of the maximum values for the constraint velocity, the ith component of $\overline{\boldsymbol{r}}_{v_c}$ can be chosen as

$$\overline{r}^i_{v_c} = \min\left\{-e^i_t(t) - {\boldsymbol{k}^i}^T\boldsymbol{e}(t),\ \overline{v}^i(t)\right\}. \tag{4.9}$$

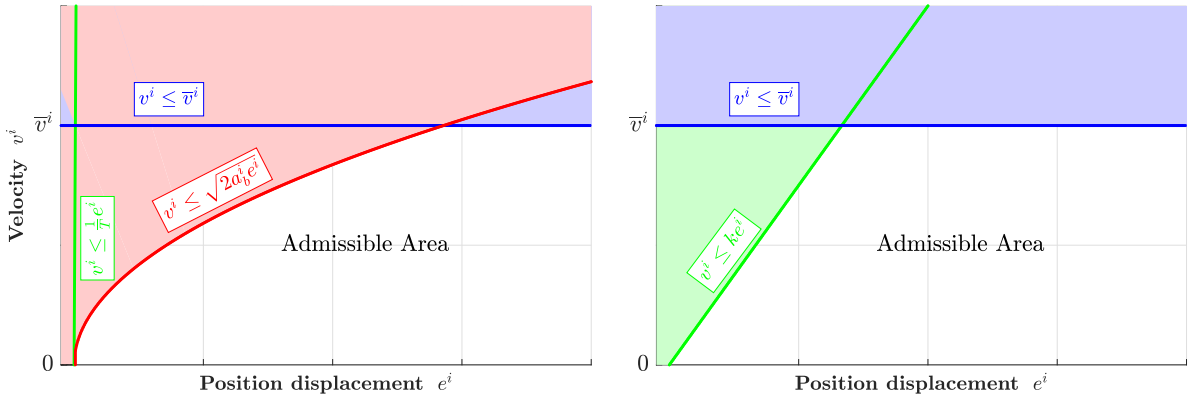


Figure 4.1: Shaping of the upper velocity bound in case of $e_t^i(t) = 0$ and $\overline{v}^i(t) = \overline{v}^i$: the shaping obtained through the formulation in (4.10) is reported on the left, whereas the one obtained with the proposed shaping (4.9) is reported on the right; the white areas represent the set of admissible pairs $(e^i(t), v^i(t))$ according to the corresponding bound shaping. The figures have been generated using $T = 0.005\,\text{s}$, $\overline{v}^i = 2\,\text{m/s}$, $a_b^i = 0.75\,\text{m/s}^2$ and $\boldsymbol{K} = k\boldsymbol{I}$, with $k = 1.5\,\text{s}^{-1}$.

It should be noticed that the proposed velocity bound (4.9) improves the shaping by Flacco, De Luca, and Khatib (2015). This can be written as

$$\overline{r}_v^i(t) = \min\left\{\frac{e^i(t)}{T},\ \overline{v}^i, \sqrt{2a_b^i e^i(t)}\right\}, \tag{4.10}$$

with T being the control cycle time and $a_b^i > 0$ a parameter to adjust the shaping when the value of e^i is approaching 0. Ignoring for a moment the third term on the right-hand side of (4.10), it can be easily recognized that the bound proposed in (4.9) generalizes the one in (4.10). Indeed, the latter is obtained from (4.9) assuming $e_t^i = 0$, $\overline{v}^i$ to be constant over time, and $\boldsymbol{k}^i$ to be a vector having $1/T$ in the ith position and zero in all the remaining ones. Furthermore, the typically small value of T (usually $1 \div 5$ milliseconds in nowadays controllers) allows the task function to approach zero with high speed, leading to sudden deceleration when the limit is actually hit. This typically undesired effect is avoided by the introduction of the third term on the right-hand side of (4.10), whose action can be regulated via the braking parameter a_b^i. However, this makes the contribution of the first term on the right-hand side of (4.10) irrelevant in practice, as it always results in a less restricting bound, no matter the value of e^i. On the other hand, the proposed bound (4.9) allows, through the setting of $\boldsymbol{k}^{iT}$, to adjust the maximum rate of convergence (over time) of $\boldsymbol{e}$ and, thus, also to regulate the maximum allowed speed in the proximity of the zero value. The above considerations can be better understood when looking at Fig. 4.1, in which a practical case of shaping of the constraint velocity upper bound according to (4.9) and (4.10) is presented.

Similarly to the velocity case, the formulation from Sect. 3.3.4 leads to the following second-order differential constraint

$$\underline{\boldsymbol{r}}_{a_c} \leq \boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{a_c}, \tag{4.11}$$

which, by recalling that $\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}\boldsymbol{\tau}'$, is equivalently written for torque-controlled robots as

$$\underline{\boldsymbol{r}}_{a_c} \leq \boldsymbol{J}_c\boldsymbol{M}^{-1}\boldsymbol{\tau}' + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{r}}_{a_c}. \tag{4.12}$$

The terms $\underline{\boldsymbol{r}}_{a_c}$ and $\overline{\boldsymbol{r}}_{a_c}$ in (4.11)–(4.12) represent the lower and upper bounds for the constraint acceleration $\dot{\boldsymbol{v}} = \boldsymbol{J}_c\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_c\dot{\boldsymbol{q}}$.

Without loss of generality, let focus again on the shaping of the upper bound. The choice

$$\overline{\boldsymbol{r}}_{a_c}(t) = -\frac{\partial \boldsymbol{e}_t}{\partial t}(t) - \boldsymbol{D}\dot{\boldsymbol{e}}(t) - \boldsymbol{K}\boldsymbol{e}(t), \quad \boldsymbol{D}, \boldsymbol{K} \geq \boldsymbol{0} \tag{4.13}$$

imposes an evolution of the task function that is no faster than a second-order linear system characterized by stiffness $\boldsymbol{K}$ and damping $\boldsymbol{D}$. However, it might be desired, through a more sophisticated shaping of $\overline{\boldsymbol{r}}_{a_c}$, to also limit the constraint velocity $\boldsymbol{v} = \boldsymbol{J}_c\dot{\boldsymbol{q}}$. Recalling from (3.10) that $\dot{\boldsymbol{e}} = \boldsymbol{v} + \boldsymbol{e}_t$, and introducing $\boldsymbol{K}_1 = \boldsymbol{D}^{-1}\boldsymbol{K}$, the upper bound (4.13) can be rewritten as

$$\overline{\boldsymbol{r}}_{a_c}(t) = -\frac{\partial \boldsymbol{e}_t}{\partial t}(t) - \boldsymbol{D}\left(\boldsymbol{v}(t) - (-\boldsymbol{e}_t(t) - \boldsymbol{K}_1\boldsymbol{e}(t))\right). \tag{4.14}$$

Limitations on the constraint velocity can then be integrated in (4.14) by replacing the term $-\boldsymbol{e}_t(t) - \boldsymbol{K}_1\boldsymbol{e}(t)$ with the corresponding velocity bound computed as in (4.9)

$$\overline{\boldsymbol{r}}_{a_c}(t) = -\frac{\partial \boldsymbol{e}_t}{\partial t}(t) - \boldsymbol{D}(\boldsymbol{v}(t) - \overline{\boldsymbol{r}}_{v_c}(t)). \tag{4.15}$$

Finally, limitations on the constraint acceleration, $\dot{\boldsymbol{v}}$, can be directly integrated into (4.15). Indicating with $\overline{\boldsymbol{a}}$ the vector of the maximum allowed accelerations, the ith component of $\overline{\boldsymbol{r}}_{a_c}$ can be chosen as

$$\overline{r}^i_{a_c}(t) = \min\left\{-e^i_{t_t}(t) - \boldsymbol{d}^{i\,T}(\boldsymbol{v}(t) - \overline{\boldsymbol{r}}_{v_c}),\ \overline{a}^i(t)\right\}, \tag{4.16}$$

where the symbol $-e^i_{t_t}$ has been used to indicate the ith component of $\frac{\partial \boldsymbol{e}_t}{\partial t}$. The proposed acceleration bound (4.16) generalizes the shaping proposed by Flacco, De Luca, and Khatib (2012) and Osorio, Allmendinger, et al. (2019). These can be obtained by assuming $\boldsymbol{e}_t = \boldsymbol{0}$, the maximum velocity $\overline{\boldsymbol{v}}$ to be constant over time, and by choosing $\boldsymbol{d}^i$ as a vector having $1/T$ in the ith position and zero in all remaining ones. Furthermore, the first term of the bounds by Flacco, De Luca, and Khatib (2012) and Osorio, Allmendinger, et al. (2019) can be neglected for the same reasons discussed for the bound (4.10).

4.3.3 Generalized control problem

Consider the case in which a given task specification is composed by both equality and inequality constraints. Analyzing the constraints in (4.1)–(4.3), it can be easily noticed that it is possible to bring them all into the form

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}. \tag{4.17}$$

	velocity	**acceleration**	**torque**
$\boldsymbol{A}$	$\boldsymbol{J}_{eq,c}$	$\boldsymbol{J}_{eq,c}$	$\boldsymbol{J}_{eq,c}\boldsymbol{M}^{-1}$
$\boldsymbol{u}$	$\dot{\boldsymbol{q}}$	$\ddot{\boldsymbol{q}}$	$\boldsymbol{\tau}' = \boldsymbol{\tau} - \boldsymbol{c} - \boldsymbol{g}$
$\boldsymbol{b}$	$\boldsymbol{r}_{v_c}$	$\boldsymbol{r}_{a_c} - \dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}}$	$\boldsymbol{r}_{a_c} - \dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}}$
$\boldsymbol{C}$	$\boldsymbol{J}_{iq,c}$	$\boldsymbol{J}_{iq,c}$	$\boldsymbol{J}_{iq,c}\boldsymbol{M}^{-1}$
$\underline{\boldsymbol{d}}$	$\underline{\boldsymbol{r}}_{v_c}$	$\underline{\boldsymbol{r}}_{a_c} - \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}}$	$\underline{\boldsymbol{r}}_{a,c} - \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}}$
$\overline{\boldsymbol{d}}$	$\overline{\boldsymbol{r}}_{v_c}$	$\overline{\boldsymbol{r}}_{a_c} - \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}}$	$\overline{\boldsymbol{r}}_{a,c} - \dot{\boldsymbol{J}}_{iq,c}\dot{\boldsymbol{q}}$

Table 4.1: Definitions of the variables in (4.17)–(4.18) for velocity-, acceleration- and torque-based schemes.

Similarly, the constraints in (4.7), (4.11)–(4.12) can take the form

$$\underline{\boldsymbol{d}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{d}}. \tag{4.18}$$

The definition of all the variables in (4.17) and (4.18) for the different control schemes can be found in Tab. 4.1, where the subscripts eq and iq are used to indicate the Jacobian matrices that are related to the task functions involved in equality and inequality constraints, respectively. Although not directly included in Tab. 4.1, it should be noticed that the constraints (4.4) can also be brought to the form (4.18) for torque-based schemes by setting $\boldsymbol{C} = \boldsymbol{I}$, $\underline{\boldsymbol{d}} = \underline{\boldsymbol{\tau}} - \boldsymbol{c} - \boldsymbol{g}$, and $\overline{\boldsymbol{d}} = \overline{\boldsymbol{\tau}} - \boldsymbol{c} - \boldsymbol{g}$.

The problem of controlling a robot can therefore be reformulated as the problem of finding, for each instant of time, a suitable vector $\boldsymbol{u}$ such that a certain number of constraints given in the form (4.17)–(4.18) are satisfied. In case of redundant robots, infinite solutions exist to this problem. Thus, a solution can be found by solving a constrained optimization problem. In the context of redundancy resolution, the control effort is typically chosen as a cost, i.e.,

$$\begin{aligned} \min_{\boldsymbol{u}} \ & \frac{1}{2}\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{u} \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}, \quad \underline{\boldsymbol{d}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{d}}, \end{aligned} \tag{4.19}$$

where $\boldsymbol{H} \in \mathbb{R}^{n\times n}$ is an arbitrary, positive (semi-) definite weighting matrix. However, it is often desired to control the residual redundant DoFs of the robot by introducing a reference vector $\boldsymbol{u}_r$ and solving the problem

$$\begin{aligned} \min_{\boldsymbol{u}} \ & \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}_r)^T\boldsymbol{H}(\boldsymbol{u} - \boldsymbol{u}_r) \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}, \quad \underline{\boldsymbol{d}} \leq \boldsymbol{C}\boldsymbol{u} \leq \overline{\boldsymbol{d}}. \end{aligned} \tag{4.20}$$

This is of particular interest in acceleration and torque-based schemes, where $\boldsymbol{u}_r$ is typically modeled to damp internal motion of the manipulator (De Luca, Oriolo, et al. 1992). Another common practice is to use $\boldsymbol{u}_r$ to include an auxiliary optimization criteria (e.g.,

manipulability) via the GBP method by Liegeois et al. (1977). The choice of $\boldsymbol{H}$ in (4.19) and (4.20) is left to the user. However, specific metrics exist, which ensure dynamical consistency of the generated robot motion (Peters, Mistry, et al. 2008; Schettino, Fiore, et al. 2020). Such metrics are used in Section 4.3.5-A.

Omitting the inequality constraints, the solution for (4.20) is given by (Liegeois et al. 1977)

$$\boldsymbol{u} = \boldsymbol{A}^{\dagger H}\boldsymbol{b} + \boldsymbol{P}\boldsymbol{u}_r, \tag{4.21}$$

where $\boldsymbol{A}^{\dagger H} = \boldsymbol{H}^{-1}\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{H}^{-1}\boldsymbol{A}^T\right)^{-1}$ is the right pseudoinverse of $\boldsymbol{A}$, weighted through $\boldsymbol{H}$, and $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{A}^{\dagger H}\boldsymbol{A}$ is a projector in the null space of $\boldsymbol{A}$. In (4.21), matrix $\boldsymbol{A}$ is assumed to be full row-rank. Such hypothesis will be held also for matrix $\boldsymbol{C}$. If this is not the case, singularity-robust techniques should be used in order to perform pseudoinversions (see the work by Chiaverini (1997) and Di Vito, Natale, et al. (2017) for an overview).

The optimization problem in (4.19) can be further extended to minimize

$$\min_{\boldsymbol{u}} \ \frac{1}{2}(\boldsymbol{u} - \boldsymbol{u}_r)^T\boldsymbol{H}(\boldsymbol{u} - \boldsymbol{u}_r) \tag{4.22}$$

while attempting to satisfying a set of multiple tasks (imposing equality and inequality constraints) specified with different levels of priorities

$$\begin{aligned}
1^{\text{st}}\text{ level} &: \boldsymbol{A}_1\boldsymbol{u} = \boldsymbol{b}_1, \quad \underline{\boldsymbol{d}}_1 \leq \boldsymbol{C}_1\boldsymbol{u} \leq \overline{\boldsymbol{d}}_1 \\
&\vdots \quad , \quad \vdots \\
\text{k}^{\text{th}}\text{ level} &: \boldsymbol{A}_k\boldsymbol{u} = \boldsymbol{b}_k, \quad \underline{\boldsymbol{d}}_k \leq \boldsymbol{C}_k\boldsymbol{u} \leq \overline{\boldsymbol{d}}_k \\
&\vdots \quad , \quad \vdots \\
\text{N}^{\text{th}}\text{ level} &: \boldsymbol{A}_N\boldsymbol{u} = \boldsymbol{b}_N, \quad \underline{\boldsymbol{d}}_N \leq \boldsymbol{C}_N\boldsymbol{u} \leq \overline{\boldsymbol{d}}_N,
\end{aligned} \tag{4.23}$$

with $\boldsymbol{A}_k$, $\boldsymbol{b}_k$, $\boldsymbol{C}_k$, $\underline{\boldsymbol{d}}_k$ and $\overline{\boldsymbol{d}}_k$ describing the equality and inequality constraints of the kth priority level (lower k denoting higher priority). Omitting the inequality constraints, the solution to (4.22)–(4.23) can be computed recursively as (Siciliano and J.-J. Slotine 1991)

$$\begin{cases}
\boldsymbol{u}_0 = \boldsymbol{0} \\
\boldsymbol{u}_k = \boldsymbol{u}_{k-1} + (\boldsymbol{A}_k\boldsymbol{P}_{k-1})^{\dagger H}\left(\boldsymbol{b}_k - \boldsymbol{A}_k\boldsymbol{u}_{k-1}\right) \\
\boldsymbol{u} = \boldsymbol{u}_N + \boldsymbol{P}_N\boldsymbol{u}_r
\end{cases}, \tag{4.24}$$

with $k = 1, ..., N$. The matrix $\boldsymbol{P}_k \in \mathbb{R}^{n\times n}$ in (4.24) is the well-known augmented projector (Siciliano and J.-J. Slotine 1991), which can also be recursively computed as (Greville 1960)

$$\begin{cases}
\boldsymbol{P}_0 = \boldsymbol{I} \\
\boldsymbol{P}_k = (\boldsymbol{I} - (\boldsymbol{A}_k\boldsymbol{P}_{k-1})^{\dagger H}\boldsymbol{A}_k)\boldsymbol{P}_{k-1}
\end{cases}. \tag{4.25}$$

Finding a solution that fulfills also the inequality constraints is a more complex challenge, and it is the subject of the following section.

	velocity	acceleration	torque
$\boldsymbol{b}'$	$\boldsymbol{r}_{v_c}$	$\boldsymbol{r}_{a_c}$	$\boldsymbol{r}_{a_c}$
$\boldsymbol{b}''$	$\mathbf{0}$	$\dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}}$	$\dot{\boldsymbol{J}}_{eq,c}\dot{\boldsymbol{q}}$

Table 4.2: Definition of $\boldsymbol{b}'$ and $\boldsymbol{b}''$ in (4.26) for velocity, acceleration and torque-based schemes

4.3.4 Extended Saturation in the Null Space Method

As remarked in Sect. 4.2, the SNS algorithm by Flacco, De Luca, and Khatib (2015) is a powerful tool for the (kinematic) control of redundant robots under hard joint constraints. In multiple iterations, it computes which joint requires the most scaling to be within its limits, and subsequently adds the inequality constraint of that limit to the equality constraints of the task. Moreover, a new scaling factor is computed for the task velocity reference, which would ensure all joints to be within their limits at the cost of a reduced speed in task execution. This cycle is repeated until either a solution that does not require task scaling is found, or all redundant DoFs of the robot are exhausted. In the latter case, the solution that requires the least scaling of the task (among the ones obtained in all the iterations) is returned.

This section presents the proposed extension to the SNS algorithm, named *extended SNS* (*eSNS*), and its integration into the unified control framework introduced in the previous section. Thanks to this framework, the eSNS algorithm can solve redundancy resolution problems defined at velocity, acceleration or torque level, indistinctly. Furthermore, compared to the original SNS, the proposed algorithm can handle arbitrary inequality constraints in the form (4.18) on multiple priority levels. Finally, compared to pure joint velocity norm minimization, the eSNS attempts at minimizing the cost defined in (4.20), which includes the weighting matrix $\boldsymbol{H}$ and the additional input vector $\boldsymbol{u}_r$.

Basic eSNS

With reference to Algorithm 4.1, this section shows the steps of the eSNS in its basic version.

First, the term $\boldsymbol{b}_k$ from (4.24) can be decomposed as $\boldsymbol{b}_k = \boldsymbol{b}'_k - \boldsymbol{b}''_k$. The definitions for $\boldsymbol{b}'_k, \boldsymbol{b}''_k \in \mathbb{R}^n$ can be found in Tab. 4.2 (where dependency on k is omitted for brevity). Thus, the solution (4.24) for the kth level of priority considering only the equality constraints can be rewritten as

$$\boldsymbol{u}_k = \boldsymbol{u}_{k-1} + (\boldsymbol{A}_k \boldsymbol{P}_{k-1})^{\dagger H} (\boldsymbol{b}'_k - \boldsymbol{b}''_k - \boldsymbol{A}_k \boldsymbol{u}_{k-1}) + \boldsymbol{P}_k \boldsymbol{u}_{r,k}, \tag{4.26}$$

with

$$\boldsymbol{u}_{r,k} = \begin{cases} \boldsymbol{u}_r, & \text{if } k = N \\ \mathbf{0}, & \text{otherwise} \end{cases}.$$

To additionally ensure satisfaction of inequality constraints, the solution (4.26) is ex-

Algorithm 4.1 Basic Extended SNS algorithm for multiple priority levels

1: $\boldsymbol{P}_0 = \boldsymbol{I}$, $\boldsymbol{u}_0 = \boldsymbol{0}$
2: **for** $k = 1 \rightarrow N$ **do**
3: $\quad \tilde{\boldsymbol{A}}_k = \boldsymbol{A}_k \boldsymbol{P}_{k-1}, \quad \boldsymbol{P}_k = \left(\boldsymbol{I} - \tilde{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{A}_k\right) \boldsymbol{P}_{k-1}$
4: $\quad \boldsymbol{u}_{sat} = \boldsymbol{0},\ \boldsymbol{u}' = \tilde{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{b}_k',\ \boldsymbol{u}'' = \tilde{\boldsymbol{A}}_k^{\dagger H} \left(-\boldsymbol{b}_k'' - \boldsymbol{A}_k \boldsymbol{u}_{k-1}\right),\ \boldsymbol{u}''' = \boldsymbol{P}_k \boldsymbol{u}_{r,k}$
5: $\quad s_k^* = 0, \quad \boldsymbol{C}_{1\rightarrow k}^{sat} = \text{null}, \quad \boldsymbol{d}_{1\rightarrow k}^{sat} = \text{null}$
6: $\quad$ **repeat**
7: $\quad\quad$ limits_violated = FALSE
8: $\quad\quad \boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat} + \boldsymbol{u}' + \boldsymbol{u}'' + \boldsymbol{u}'''$
9: $\quad\quad \boldsymbol{\rho} = \boldsymbol{C}_{1\rightarrow k} \boldsymbol{u}', \quad \boldsymbol{\phi} = \boldsymbol{C}_{1\rightarrow k} \boldsymbol{u}_k, \quad \boldsymbol{\sigma} = \boldsymbol{\phi} - \boldsymbol{\rho}$
10: $\quad\quad$ **if** $\exists i \in [1 : l_k] : \left(\phi^i < \underline{d}_{1\rightarrow k}^i\right) \vee \left(\phi^i > \overline{d}_{1\rightarrow k}\right)$ **then**
11: $\quad\quad\quad$ limits_violated = TRUE
12: $\quad\quad\quad$ get_task_scaling_factor($\boldsymbol{\rho}$,$\boldsymbol{\sigma}$) ▷ call alg. 4.2
13: $\quad\quad\quad$ **if** task scaling factor $> s_k^*$ **then**
14: $\quad\quad\quad\quad s_k^* =$ task scaling factor
15: $\quad\quad\quad\quad \boldsymbol{u}_{sat}^* = \boldsymbol{u}_{sat}, \quad \boldsymbol{u}'^* = \boldsymbol{u}', \quad \boldsymbol{u}''^* = \boldsymbol{u}'', \quad \boldsymbol{u}'''^* = \boldsymbol{u}'''$
16: $\quad\quad\quad$ **end if**
17: $\quad\quad\quad j =$ most critical constraint
18: $\quad\quad\quad \boldsymbol{C}_{1\rightarrow k}^{sat} \leftarrow \text{concatenate}(\boldsymbol{C}_{1\rightarrow k}^{sat}, \boldsymbol{c}_{1\rightarrow k}^j)$
19: $\quad\quad\quad \hat{\boldsymbol{P}}_{k-1} = \left(\boldsymbol{I} - \left(\boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{P}_{k-1}\right)^{\dagger H} \boldsymbol{C}_{1\rightarrow k}^{sat}\right) \boldsymbol{P}_{k-1}$
20: $\quad\quad\quad \hat{\boldsymbol{A}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}$
21: $\quad\quad\quad$ **if** $\text{rank}(\hat{\boldsymbol{A}}_k) \geq m_k$ **then**
22: $\quad\quad\quad\quad d_{1\rightarrow k}^j = \begin{cases} \underline{d}_{1\rightarrow k}^j, & \phi^j < \underline{d}_{1\rightarrow k}^j \\ \overline{d}_{1\rightarrow k}^j, & \phi^j > \overline{d}_{1\rightarrow k}^j \end{cases}$
23: $\quad\quad\quad\quad \boldsymbol{d}_{1\rightarrow k}^{sat} \leftarrow \text{concatenate}(\boldsymbol{d}_{1\rightarrow k}^{sat}, d_{1\rightarrow k}^j)$
24: $\quad\quad\quad\quad \hat{\boldsymbol{C}}_k = \boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{P}_{k-1}, \quad \hat{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \hat{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{A}_k\right) \hat{\boldsymbol{P}}_{k-1}$
25: $\quad\quad\quad\quad \boldsymbol{u}_{sat} = \hat{\boldsymbol{C}}_k^{\dagger H} \left(\boldsymbol{d}_{1\rightarrow k}^{sat} - \boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{u}_{k-1}\right), \quad \boldsymbol{u}' = \hat{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{b}_k'$
26: $\quad\quad\quad\quad \boldsymbol{u}'' = \hat{\boldsymbol{A}}_k^{\dagger H} \left(-\boldsymbol{b}_k'' - \boldsymbol{A}_k \left(\boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat}\right)\right), \quad \boldsymbol{u}''' = \hat{\boldsymbol{P}}_k \boldsymbol{u}_{r,k}$
27: $\quad\quad\quad$ **else**
28: $\quad\quad\quad\quad \boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat}^* + s_k^* \boldsymbol{u}'^* + \boldsymbol{u}''^* + \boldsymbol{u}'''^*$
29: $\quad\quad\quad\quad$ limits_violated = FALSE
30: $\quad\quad\quad$ **end if**
31: $\quad\quad$ **end if**
32: $\quad$ **until** limits_violated = FALSE
33: **end for**
34: $\boldsymbol{u} = \boldsymbol{u}_N$

panded to the form (line 8)[1]

$$\boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat} + \boldsymbol{u}' + \boldsymbol{u}'' + \boldsymbol{u}''', \tag{4.27}$$

[1] All references to line numbers in this section refer to Algorithm 1.

with the following initialization (line 4):

$$\begin{aligned}\boldsymbol{u}_{sat} &= \boldsymbol{0}\\ \boldsymbol{u}' &= (\boldsymbol{A}_k\boldsymbol{P}_{k-1})^{\dagger H}\,\boldsymbol{b}'_k\\ \boldsymbol{u}'' &= (\boldsymbol{A}_k\boldsymbol{P}_{k-1})^{\dagger H}\,(-\boldsymbol{b}''_k - \boldsymbol{A}_k(\boldsymbol{u}_{k-1}+\boldsymbol{u}_{sat}))\\ \boldsymbol{u}''' &= \boldsymbol{P}_k\boldsymbol{u}_{r,k}.\end{aligned}$$

The vector $\boldsymbol{u}_k$ is considered an admissible solution if it satisfies all the inequality constraints defined in the first k levels of priority, namely

$$\underline{\boldsymbol{d}}_{1\to k} \leq \boldsymbol{C}_{1\to k}\boldsymbol{u}_k \leq \overline{\boldsymbol{d}}_{1\to k}, \tag{4.28}$$

with

$$\begin{aligned}\underline{\boldsymbol{d}}_{1\to k} &= \begin{bmatrix}\underline{\boldsymbol{d}}_1^T & \dots & \underline{\boldsymbol{d}}_k^T\end{bmatrix}^T \in \mathbb{R}^{l_k}\\ \overline{\boldsymbol{d}}_{1\to k} &= \begin{bmatrix}\overline{\boldsymbol{d}}_1^T & \dots & \overline{\boldsymbol{d}}_k^T\end{bmatrix}^T \in \mathbb{R}^{l_k}\\ \boldsymbol{C}_{1\to k} &= \begin{bmatrix}\boldsymbol{C}_1^T & \dots & \boldsymbol{C}_k^T\end{bmatrix}^T \in \mathbb{R}^{l_k\times n}.\end{aligned}$$

If a solution $\boldsymbol{u}_k$ is admissible (line 10), the algorithm moves to the next priority level. If, instead, some of the inequality constraints in (4.28) are violated, the eSNS starts/keeps iterating (line 6) to find a valid solution, eventually scaling the task velocity/acceleration in $\boldsymbol{b}'_k$ by a factor $s_k \in [0,1]$ (line 28), if needed. At each iteration, the task scaling factor s_k is computed using Algorithm 4.2 (line 12). The input arguments $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$ are given as (line 9):

$$\begin{aligned}\boldsymbol{\rho} &= \boldsymbol{C}_{1\to k}\boldsymbol{u}'\\ \boldsymbol{\sigma} &= \boldsymbol{C}_{1\to k}\boldsymbol{u}_k - \boldsymbol{\rho}.\end{aligned} \tag{4.29}$$

Although a new scaling factor is computed at every eSNS iteration, it is only applied when all redundant DoFs of the robot have been exhausted without finding a valid solution (line 28). In all other cases, it is only checked whether the current solution allows larger task scaling. In such a case, the solution (and the corresponding task scaling factor) is stored as the best found (lines 14–15).

Algorithm 4.2 also returns the most critical constraint, i.e., the violated inequality constraint that requires the smallest scaling factor s_k to be satisfied. This constraint is used in the next phase of the algorithm, which is the saturation phase (lines 17–26). First, the most critical constraint is converted into an equality constraint

$$(\boldsymbol{c}^j_{1\to k})^T\boldsymbol{u}_k = d^j_{1\to k},$$

with j being the index of the most critical constraint and

$$d^j_{1\to k} = \begin{cases}\underline{d}^j_{1\to k}, & \text{if } (\boldsymbol{c}^j_{1\to k})^T\boldsymbol{u}_k < \underline{d}^j_{1\to k}\\ \overline{d}^j_{1\to k}, & \text{if } (\boldsymbol{c}^j_{1\to k})^T\boldsymbol{u}_k > \overline{d}^j_{1\to k}\end{cases}.$$

Algorithm 4.2 Task scaling factor and most critical constraint computation

1: **function** GET_TASK_SCALING_FACTOR($\boldsymbol{\rho}$,$\boldsymbol{\sigma}$)
2: **for** $i = 1 \rightarrow l_k$ **do**
3: $\underline{s}^i = \left(\underline{d}^i_{1\rightarrow k} - \sigma^i\right)/\rho^i$
4: $\overline{s}^i = \left(\overline{d}^i_{1\rightarrow k} - \sigma^i\right)/\rho^i$
5: **if** $\underline{s}^i > \overline{s}^i$ **then**
6: switch($\underline{s}^i, \overline{s}^i$)
7: **end if**
8: **end for**
9: $s_{\max} = \min_i \{\overline{\boldsymbol{s}}\}$
10: $s_{\min} = \max_i \{\underline{\boldsymbol{s}}\}$
11: most critical constraint = $\operatorname{argmin}_i \{\overline{\boldsymbol{s}}\}$
12: **if** $s_{\min} > s_{\max} \vee s_{\max} < 0 \vee s_{\min} > 1$ **then**
13: task scaling factor = 0
14: **else**
15: task scaling factor = $\min\{s_{\max}, 1\}$
16: **end if**
17: **end function**

Then, it is added to the *saturation set*, which is the set of all converted inequalities, forming a system of equations that can be expressed as

$$\boldsymbol{C}^{sat}_{1\rightarrow k}\boldsymbol{u}_k = \boldsymbol{d}^{sat}_{1\rightarrow k}. \tag{4.30}$$

Finally, the saturation commands (4.30) are enforced in the solution (4.27), updating its terms as follows (lines 25–26):

$$\begin{aligned}
\boldsymbol{u}_{sat} &= \left(\boldsymbol{C}^{sat}_{1\rightarrow k}\boldsymbol{P}_{k-1}\right)^{\dagger H}\left(\boldsymbol{d}^{sat}_{1\rightarrow k} - \boldsymbol{C}^{sat}_{1\rightarrow k}\boldsymbol{u}_{k-1}\right)\\
\boldsymbol{u}' &= \left(\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\right)^{\dagger H}\boldsymbol{b}'_k\\
\boldsymbol{u}'' &= \left(\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\right)^{\dagger H}\left(-\boldsymbol{b}''_k - \boldsymbol{A}_k(\boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat})\right)\\
\boldsymbol{u}''' &= \hat{\boldsymbol{P}}_k\boldsymbol{u}_{r,k},
\end{aligned} \tag{4.31}$$

with

$$\begin{aligned}
\hat{\boldsymbol{P}}_{k-1} &= \left(\boldsymbol{I} - \left(\boldsymbol{C}^{sat}_{1\rightarrow k}\boldsymbol{P}_{k-1}\right)^{\dagger H}\boldsymbol{C}^{sat}_{1\rightarrow k}\right)\boldsymbol{P}_{k-1}\\
\hat{\boldsymbol{P}}_k &= \left(\boldsymbol{I} - \left(\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\right)^{\dagger H}\boldsymbol{A}_k\right)\hat{\boldsymbol{P}}_{k-1}
\end{aligned}.$$

It should be noticed that enforcing a new saturation only makes sense if there are enough redundant DoFs left in the system to handle it. From an algebraic point of view, this condition is satisfied if $\operatorname{rank}(\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}) \geq m_k$ (line 21), with $\boldsymbol{A}_k \in \mathbb{R}^{m_k\times n}$ ($m_k \leq n$).

Relation to the original SNS algorithm

Analyzing Algorithm 4.1, a similar structure as the original SNS algorithm (Flacco, De Luca, and Khatib 2015) can be recognized: after computing a first guess that satisfies the equality constraints, the solution is updated as long as one or more limits are found to be violated. Insertions to the saturation set happen using a policy based on the identification of the most critical constraint. Finally, task scaling is applied when all redundant DoFs have been used without finding a valid solution. It can also be noticed from (4.31) that, as in the original SNS algorithm, an intermediate level of priority is imposed between the saturation commands $\boldsymbol{u}_{sat}$ and the other terms introduced in (4.27). This is guaranteed by the null space projectors $\hat{\boldsymbol{P}}_{k-1}$ and $\hat{\boldsymbol{P}}_k$.

On the other hand, thanks to the special structure of $\hat{\boldsymbol{C}}_k$ and $\hat{\boldsymbol{P}}_{k-1}$, Algorithm 4.1 can perform saturation in any task space and, thus, handle arbitrary inequality constraints in the form (4.18). This is a key extension compared to the work by Flacco, De Luca, and Khatib (2015), where only joint space inequality constraints have been handled. Moreover, the general cost (4.20) is considered in Algorithm 4.1 through the use of general weighted pseudoinverses and the inclusion of the $\boldsymbol{u}'''$ in (4.31). In fact, specializing Algorithm 4.1 for velocity-based redundancy resolution, setting $\boldsymbol{H} = \boldsymbol{I} \in \mathbb{R}^{n\times n}$, $\boldsymbol{u}_r = \boldsymbol{0}$ and considering only joint space constraint with the highest priority ($\boldsymbol{C}_{1\to k} = \boldsymbol{I} \in \mathbb{R}^{n\times n}\, \forall k = 1,\dots,N$) would return the multi-task SNS algorithm by Flacco, De Luca, and Khatib (2015).

To further highlight the relation between eSNS and SNS, the saturation commands $\boldsymbol{u}_{sat}$ in (4.31) can be rewritten in the form

$$\boldsymbol{u}_{sat} = \left((\boldsymbol{I} - \boldsymbol{W}_k)\, \boldsymbol{C}_{1\to k}\boldsymbol{P}_{k-1}\right)^{\dagger H} \boldsymbol{d}_{sat}, \tag{4.32}$$

where

$$d^j_{sat} = \begin{cases} \underline{d}^j_{1\to k} - (\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}_{k-1}, & \text{if } (\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}_k < \underline{d}^j_{1\to k} \\ \overline{d}^j_{1\to k} - (\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}_{k-1}, & \text{if } (\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}_k > \overline{d}^j_{1\to k} \\ 0, & \text{otherwise} \end{cases},$$

from which it is easier to recognize the similarity with the joint velocity saturation of the SNS. The diagonal matrix $\boldsymbol{W}_k \in \mathbb{R}^{l_k\times l_k}$, whose diagonal has 0 elements in the entries corresponding to the constraints in the saturation set and 1 in all other entries, operates as a constraint selection matrix in (4.32) as in the original SNS. Also the projector $\hat{\boldsymbol{P}}_{k-1}$ can be brought in a form that recalls the one introduced by Flacco, De Luca, and Khatib (2015)

$$\hat{\boldsymbol{P}}_{k-1} = \left(\boldsymbol{I} - \left((\boldsymbol{I} - \boldsymbol{W}_k)\, \boldsymbol{C}_{1\to k}\boldsymbol{P}_{k-1}\right)^{\dagger H} \boldsymbol{C}_{1\to k}\right) \boldsymbol{P}_{k-1}. \tag{4.33}$$

Fast-eSNS

This section analyzes the numerical performance of Algorithm 4.1. The goal is to obtain a more efficient algorithm, which will be named *Fast-eSNS*. The procedure will follow a similar reasoning as the work by Flacco and De Luca (2013a). However, not all the machinery can be easily transferred to the general case considered in this work, since it exploits some specific properties of the standard Moore-Penrose pseudoinverse.

At each level of priority, the first operation that is critical for the computation time is the pseudoinversion of the matrix $\tilde{\boldsymbol{A}}_k = \boldsymbol{A}_k \boldsymbol{P}_{k-1}$ (line 3 of Algorithm 4.1). Different numerical methods exist for computing the pseudoinverse of a matrix and the programmer must decide on a trade-off between speed and robustness. A common choice is to resort to the singular value decomposition, which enables the analysis of singularities and the implementation of damped pseudoinversion methods. However, a faster method based on QR-Decomposition exists, as extensively pointed out by Flacco and De Luca (2013a) and Ziese, Fiore, et al. (2020). Consider the QR-Decomposition of the product $\tilde{\boldsymbol{A}}_k \boldsymbol{H}^{-1} \tilde{\boldsymbol{A}}_k^T$

$$\tilde{\boldsymbol{A}}_k \boldsymbol{H}^{-1} \tilde{\boldsymbol{A}}_k^T = \boldsymbol{Q}_k \begin{bmatrix} \boldsymbol{R}_k \\ \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_k & \boldsymbol{Z}_k \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_k \\ \boldsymbol{0} \end{bmatrix},$$

with $\boldsymbol{Y}_k \in \mathbb{R}^{n \times m_k}$ and $\boldsymbol{Z_k} \in \mathbb{R}^{n \times (n-m_k)}$ orthogonal matrices composing $\boldsymbol{Q}_k$, and $\boldsymbol{R}_k \in \mathbb{R}^{m_k \times m_k}$ being upper triangular. Then, the weighted pseudoinverse of $\tilde{\boldsymbol{A}}_k$ can be computed as

$$\tilde{\boldsymbol{A}}_k^{\dagger H} = \boldsymbol{H}^{-1} \tilde{\boldsymbol{A}}_k^T \boldsymbol{Y}_k \boldsymbol{R}_k^{-T},$$

which requires the inversion of an $m_k \times m_k$ upper triangular matrix only.

Proceeding with the analysis of Algorithm 4.1, it can be easily recognized that the second expensive operation is the update of the solution $\boldsymbol{u}_k$ (lines 25–26). This requires, at every iteration, the computation of the pseudoinverse of $\hat{\boldsymbol{C}}_k = \boldsymbol{C}_{1\to k}^{sat} \boldsymbol{P}_{k-1}$ and $\hat{\boldsymbol{A}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}$. However, the new solution can be computed from the previous one by a rank-one update of the pseudoinverse computations. The first step is the rewriting of the general solution $\boldsymbol{u}_k$ by using task augmentation (Chiacchio, Chiaverini, et al. 1991) as

$$\begin{aligned} \boldsymbol{u}_k &= \boldsymbol{u}_{k-1} + \left(\boldsymbol{A}_{TA,k} \boldsymbol{P}_{k-1}\right)^{\dagger H} \boldsymbol{b}_{TA,k} \\ &= \boldsymbol{u}_{k-1} + \left[\begin{pmatrix} \boldsymbol{A}_k \\ \boldsymbol{C}_{1\to k}^{sat} \end{pmatrix} \boldsymbol{P}_{k-1} \right]^{\dagger H} \begin{bmatrix} \boldsymbol{b}_k' - \boldsymbol{b}_k'' - \boldsymbol{A}_k \boldsymbol{u}_{k-1} \\ \boldsymbol{d}_{1\to k}^{sat} - \boldsymbol{C}_{1\to k}^{sat} \boldsymbol{u}_{k-1} \end{bmatrix}. \end{aligned} \tag{4.34}$$

It is possible to show that the result of (4.34) coincides with the one obtained from (4.31), only if a feasible solution exists. However, the case of unfeasible solutions is already excluded in Algorithm 4.1 by the condition $\operatorname{rank}(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}) \geq m_k$. At each eSNS iteration, only one new saturation command is imposed, corresponding to the most critical constraint j. Thus, (4.34) can be rewritten as

$$\begin{aligned} \boldsymbol{u}_k = \boldsymbol{u}_{k-1} \\ + \left[\begin{pmatrix} \boldsymbol{A}_{TA,k} \\ (\boldsymbol{c}_{1\to k}^j)^T \end{pmatrix} \boldsymbol{P}_{k-1} \right]^{\dagger H} \begin{bmatrix} \boldsymbol{b}_k' - \boldsymbol{b}_k'' - \boldsymbol{A}_k \boldsymbol{u}_{k-1} \\ \boldsymbol{d}_{1\to k}^{sat} - \boldsymbol{C}_{1\to k}^{sat} \boldsymbol{u}_{k-1} \\ d_{1\to k}^j - (\boldsymbol{c}_{1\to k}^j)^T \boldsymbol{u}_{k-1} \end{bmatrix}, \end{aligned}$$

with $\boldsymbol{A}_{TA,k}$ initialized to $\boldsymbol{A}_k$ at the first iteration and redefined at the end of every cycle

as:

$$\boldsymbol{A}_{TA,k} \leftarrow \begin{bmatrix} \boldsymbol{A}_{TA,k} \\ (\boldsymbol{c}^j_{1\to k})^T \end{bmatrix}. \tag{4.35}$$

It can be easily noticed that, at each new iteration, the matrix to pseudoinvert differs from the one at the previous iteration only by the appended row $(\boldsymbol{c}^j_{1\to k})^T \boldsymbol{P}_{k-1}$. Therefore, a rank-one update strategy could be used. In particular, an update method for arbitrary weighting matrix has been derived from the algorithm by Guo-rong and Yong-lin (1986) for appending a column. This yields

$$\begin{aligned} \boldsymbol{u}_k = \boldsymbol{u}_{k-1} + \begin{bmatrix} \left(\boldsymbol{I} - \boldsymbol{\chi}(\boldsymbol{c}^j_{1\to k})^T\right) \left(\boldsymbol{A}_{TA,k}\boldsymbol{P}_{k-1}\right)^{\dagger H} & \boldsymbol{\chi} \end{bmatrix} \\ \times \begin{bmatrix} \boldsymbol{b}'_k - \boldsymbol{b}''_k - \boldsymbol{A}_k \boldsymbol{u}_{k-1} \\ \boldsymbol{d}^{sat}_{1\to k} - \boldsymbol{C}^{sat}_{1\to k}\boldsymbol{u}_{k-1} \\ d^j_{1\to k} - (\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}_{k-1} \end{bmatrix}, \end{aligned} \tag{4.36}$$

with $\boldsymbol{\chi} = \left((\boldsymbol{c}^j_{1\to k})^T \hat{\boldsymbol{P}}_k\right)^{\dagger H} \in \mathbb{R}^n$ defined as *update vector*. At each new iteration the solution can then be updated as:

$$\begin{aligned} \hat{\boldsymbol{P}}_k &\leftarrow \left(\boldsymbol{I} - \boldsymbol{\chi}(\boldsymbol{c}^j_{1\to k})^T\right) \hat{\boldsymbol{P}}_k \\ \boldsymbol{u}_{sat} &\leftarrow \boldsymbol{u}_{sat} + \boldsymbol{\chi}\left(d^j_{1\to k} - (\boldsymbol{c}^j_{1\to k})^T \left(\boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat}\right)\right) \\ \boldsymbol{u}' &\leftarrow \boldsymbol{u}' - \boldsymbol{\chi}(\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}' \\ \boldsymbol{u}'' &\leftarrow \boldsymbol{u}'' - \boldsymbol{\chi}(\boldsymbol{c}^j_{1\to k})^T \boldsymbol{u}'' \\ \boldsymbol{u}''' &\leftarrow \hat{\boldsymbol{P}}_k \boldsymbol{u}_{r,k}. \end{aligned} \tag{4.37}$$

It should be noticed that only the weighted pseudoinverse of a vector is required to compute $\boldsymbol{\chi}$ and, thus, to update the solution according to (4.37). This allows to significantly speed up the update of the eSNS solution.

Optimality of the eSNS solution

This section focuses on the optimality properties of the eSNS. The analysis will refer to the Algorithm 4.1, but the same considerations would apply for the Fast-eSNS. With reference to the optimization problem (4.23), it is easy to see that Algorithm 4.1 provides an optimal solution in case no saturation commands are required. In fact, the entire algorithm reduces to (4.24) in this trivial case. In case some inequality constraints are violated, however, the eSNS can enforce saturation commands at each level of priority following a specific policy, which is based on the iterative identification of the most critical constraint. Moreover, if at the kth level of priority it was not possible to find a feasible solution $\boldsymbol{u_k}$, the eSNS applies a task scaling strategy. It should be noticed that the final scaling factor will also depend on the specific sequence with which the constraints were added to the saturation set. Thus, it can be concluded that, in general, there is no guarantee that the final saturation set and task scaling factor will produce

an optimal solution. In other words, other admissible solutions might exist that return a smaller value of the cost function in (4.23) and/or higher scale factors.

To check whether a solution is optimal in some sense, a suitable optimization criterion must be defined. To this purpose, the following optimization problem is introduced to compute a solution at the kth level of priority

$$\begin{aligned} \min_{\boldsymbol{u},s_k} \ & \frac{1}{2}(\boldsymbol{u}-\boldsymbol{u}_{r,k})^T\boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_{r,k}) + \frac{1}{2}M(1-s_k)^2 \\ \text{s.t. } & \boldsymbol{A}_k\boldsymbol{u} = s_k\boldsymbol{b}'_k - \boldsymbol{b}''_k \\ & \boldsymbol{A}_{1\to k-1}\boldsymbol{u} = \boldsymbol{b}'^*_{1\to k-1} - \boldsymbol{b}''_{1\to k-1} \\ & \underline{\boldsymbol{d}}_{1\to k} \le \boldsymbol{C}_{1\to k}\boldsymbol{u} \le \overline{\boldsymbol{d}}_{1\to k} \\ & 0 \le s_k \le 1 \end{aligned} \quad , \tag{4.38}$$

where $\boldsymbol{b}'^*_{1\to k-1} = \begin{bmatrix} s_1\boldsymbol{b}'^T_1 & \dots & s_{k-1}\boldsymbol{b}'^T_{k-1} \end{bmatrix}^T$ and $M \gg 1$ is a new (scalar) parameter used to weight the maximization of the scale factor s_k with respect to the minimization of $(\boldsymbol{u}-\boldsymbol{u}_{r,k})^T\boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_{r,k})$. The problem (4.38) can be rewritten as a standard quadratic problem with linear (equality and inequality) constraints

$$\begin{aligned} \min_{\boldsymbol{\xi}} \ & \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{\Theta}\boldsymbol{\xi} \\ \text{s.t. } & \boldsymbol{\Lambda}\boldsymbol{\xi} = \boldsymbol{\beta}, \quad \boldsymbol{\Gamma}\boldsymbol{\xi} \le \boldsymbol{\delta}, \end{aligned} \tag{4.39}$$

with

$$\begin{aligned} \boldsymbol{\xi} &= \begin{bmatrix} \boldsymbol{u}-\boldsymbol{u}_{r,k} \\ 1-s_k \end{bmatrix}, \boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{H} & \boldsymbol{0} \\ \boldsymbol{0} & M \end{bmatrix}, \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{b}'_k \\ \boldsymbol{A}_{1\to k-1} & \boldsymbol{0} \end{bmatrix}, \\ \boldsymbol{\beta} &= \begin{bmatrix} \boldsymbol{b}'_k - \boldsymbol{b}''_k - \boldsymbol{A}_k\boldsymbol{u}_{r,k} \\ \boldsymbol{b}'^*_{1\to k-1} - \boldsymbol{b}''_{1\to k-1} - \boldsymbol{A}_{1\to k-1}\boldsymbol{u}_{r,k} \end{bmatrix}, \\ \boldsymbol{\Gamma} &= \begin{bmatrix} -\boldsymbol{C}_{1\to k} & \boldsymbol{0} \\ \boldsymbol{C}_{1\to k} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \\ \boldsymbol{0} & -1 \end{bmatrix}, \quad \boldsymbol{\delta} = \begin{bmatrix} -\underline{\boldsymbol{d}}_{1\to k} + \boldsymbol{C}_{1\to k}\boldsymbol{u}_{r,k} \\ \overline{\boldsymbol{d}}_{1\to k} - \boldsymbol{C}_{1\to k}\boldsymbol{u}_{r,k} \\ 1 \\ 0 \end{bmatrix}. \end{aligned}$$

Necessary and sufficient optimality conditions for the problem (4.39) are given by the well-known Karush-Kuhn-Tucker (KKT) criteria (Kuhn and Tucker 1951):

$$\boldsymbol{\Theta}\boldsymbol{\xi} + \boldsymbol{\Lambda}^T\boldsymbol{\lambda} + \boldsymbol{\Gamma}^T\boldsymbol{\mu} = \boldsymbol{0} \tag{4.40a}$$

$$\boldsymbol{\mu}^T(\boldsymbol{\Gamma}\boldsymbol{\xi} - \boldsymbol{\delta}) = 0 \tag{4.40b}$$

$$\boldsymbol{\Lambda}\boldsymbol{\xi} = \boldsymbol{\beta} \tag{4.40c}$$

$$\boldsymbol{\Gamma}\boldsymbol{\xi} \le \boldsymbol{\delta} \tag{4.40d}$$

$$\boldsymbol{\mu} \ge \boldsymbol{0}, \tag{4.40e}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{m_1+\ldots+m_k}$ and $\boldsymbol{\mu} \in \mathbf{R}^{2l_k+2}$ are the Lagrange multipliers associated with the equality and inequality constraints, respectively.

In order to analyze the optimality properties of the eSNS, start considering condition (4.40c). This imposes the satisfaction of the kth equality constraint, possibly obtained through task scaling. Furthermore, it enforces that the $(k-1)$ higher priority equality constraints are satisfied, preserving the task scaling factors obtained by the eSNS up to the $(k-1)$th level of priority. Reviewing the structure of the eSNS, it can be easily stated that the satisfaction of the kth equality constraint is guaranteed at each iteration by construction of the algorithm. Moreover, the use of the projector $\hat{\boldsymbol{P}}_{k-1}$ in the solution ensures that the satisfaction of higher priority equality constraints (each considered with its computed task scaling factor) is preserved. In the light of the considerations above, it is possible to extract from (4.40a) an expression related only to the constraints of the kth priority level. This is achieved via left multiplication of (4.40a) by the $(n+1) \times (n+1)$ matrix

$$\boldsymbol{\Psi}_{k-1} = \begin{bmatrix} \boldsymbol{P}_{k-1}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}.$$

Recalling that $\boldsymbol{A}_{1\to k-1}\boldsymbol{P}_{k-1} = \mathbf{0}$, the above-mentioned multiplication yields

$$\tilde{\boldsymbol{\Theta}}\boldsymbol{\xi} + \tilde{\boldsymbol{\Lambda}}^T\tilde{\boldsymbol{\lambda}} + \tilde{\boldsymbol{\Gamma}}^T\boldsymbol{\mu} = \mathbf{0}, \tag{4.41}$$

where $\tilde{\boldsymbol{\Theta}} = \boldsymbol{\Psi}_{k-1}\boldsymbol{\Theta}$, $\tilde{\boldsymbol{\Lambda}} = \begin{bmatrix} \boldsymbol{A}_k\boldsymbol{P}_{k-1} & \boldsymbol{b}_k' \end{bmatrix}$, $\tilde{\boldsymbol{\Gamma}} = \boldsymbol{\Gamma}\boldsymbol{\Psi}_{k-1}^T$, and $\tilde{\boldsymbol{\lambda}} \in \mathbb{R}^{m_k}$ contains the first m_k elements of $\boldsymbol{\lambda}$.

Another condition that is automatically satisfied inside the eSNS is (4.40d). Indeed, task-related inequality constraints are checked at every eSNS iteration. Moreover, the constraints on s_k are satisfied by construction of Algorithm 4.2, which already outputs task scaling factors in the range $[0, 1]$.

It can be noticed that, for the constraints belonging to the saturation set, also condition (4.40b) is automatically satisfied inside the eSNS. Indeed, for a generic constraint j in the set, it is $\boldsymbol{\gamma}^j\boldsymbol{\xi} - \delta^j = 0$, regardless of the value of μ^j, which is imposed to be positive by (4.40e). Satisfying (4.40b) for those constraints that are not in the saturation set requires that the corresponding elements of $\boldsymbol{\mu}$ are null. Defining the $(n+1) \times (n+1)$ matrix

$$\hat{\boldsymbol{\Psi}}_{k-1} = \begin{bmatrix} \hat{\boldsymbol{P}}_{k-1}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$$

allows extracting from (4.40a) an expression related only to constraints of the kth priority level that are not in the saturation set. This is achieved via left multiplication by $\hat{\boldsymbol{\Psi}}_{k-1}$. Recalling that $\boldsymbol{A}_{1\to k-1}\hat{\boldsymbol{P}}_{k-1} = \mathbf{0}$ and that the components of $\boldsymbol{\mu}$ associated to non-saturated constraints are null, the multiplication yields

$$\hat{\boldsymbol{\Theta}}\boldsymbol{\xi} + \hat{\boldsymbol{\Lambda}}^T\tilde{\boldsymbol{\lambda}} = \mathbf{0}, \tag{4.42}$$

where $\hat{\boldsymbol{\Theta}} = \hat{\boldsymbol{\Psi}}_{k-1}\boldsymbol{\Theta}$ and $\hat{\boldsymbol{\Lambda}} = \begin{bmatrix} \boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1} & \boldsymbol{b}_k' \end{bmatrix}$. Since the possible rank-deficiency of $\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}$ is already checked at each eSNS iteration, (4.42) admits always a solution $\tilde{\boldsymbol{\lambda}}$,

which can be computed as

$$\tilde{\boldsymbol{\lambda}} = -\left(\hat{\boldsymbol{\Lambda}}^{\dagger\Theta}\right)^T \hat{\boldsymbol{\Theta}}\boldsymbol{\xi}. \tag{4.43}$$

Note that the computation of $\tilde{\boldsymbol{\lambda}}$ from (4.43) is not expensive. Indeed, given the structure of $\hat{\boldsymbol{\Lambda}}$, the matrix $\hat{\boldsymbol{\Lambda}}^{\dagger\Theta}$ can be obtained as rank-one update of $(\boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1})^{\dagger\Theta}$, which is already computed at every eSNS iteration. Substituting (4.43) in (4.41) yields

$$\tilde{\boldsymbol{\Gamma}}^T\boldsymbol{\mu} = -\left(\tilde{\boldsymbol{\Theta}} - \tilde{\boldsymbol{\Lambda}}^T\left(\hat{\boldsymbol{\Lambda}}^{\dagger\Theta}\right)^T\hat{\boldsymbol{\Theta}}\right)\boldsymbol{\xi}. \tag{4.44}$$

As already mentioned, condition (4.40b) imposes that the elements of $\boldsymbol{\mu}$ associated with the constraints that are not in the saturation set are null. Therefore, the vector $\boldsymbol{\mu}$ can be partitioned as $\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}^{satT} & \mathbf{0}\end{bmatrix}^T$, where $\boldsymbol{\mu}^{sat} \in \mathbb{R}^{l_{sat}}$ collects the Lagrange multipliers associated with the saturated constraints. Thus, (4.44) can be rewritten as

$$\left(\tilde{\boldsymbol{\Gamma}}^{sat}\right)^T\boldsymbol{\mu}^{sat} = -\left(\tilde{\boldsymbol{\Theta}} - \tilde{\boldsymbol{\Lambda}}^T\left(\hat{\boldsymbol{\Lambda}}^{\dagger\Theta}\right)^T\hat{\boldsymbol{\Theta}}\right)\boldsymbol{\xi},$$

where $\tilde{\boldsymbol{\Gamma}}^{sat} \in \mathbb{R}^{l_{sat}\times(n+1)}$ is the matrix obtained by extracting the rows of $\tilde{\boldsymbol{\Gamma}}$ associated with the saturated constraints. Given the linear independence of the rows of $\boldsymbol{C}_{1\to k}$ (as assumed in Sect. 4.3.3), the matrix $\tilde{\boldsymbol{\Gamma}}^{sat}$ can be considered as full row-rank. Thus, $\boldsymbol{\mu}^{sat}$ can be computed as

$$\boldsymbol{\mu}^{sat} = -\left(\left(\tilde{\boldsymbol{\Theta}}^T - \hat{\boldsymbol{\Theta}}^T\hat{\boldsymbol{\Lambda}}^{\dagger\Theta}\tilde{\boldsymbol{\Lambda}}\right)\left(\tilde{\boldsymbol{\Gamma}}^{sat}\right)^{\dagger\Theta}\right)^T\boldsymbol{\xi}. \tag{4.45}$$

Therefore, it can be concluded that the eSNS provides an optimal solution to the problem (4.38) if and only if the components of $\boldsymbol{\mu}^{sat}$ in (4.45) are non-negative. Note that the evaluation of (4.45) increases the computational burden of the algorithm. Indeed, an additional pseudoinverse operation is required for computing $(\tilde{\boldsymbol{\Gamma}}^{sat})^{\dagger\Theta}$. Nevertheless, the optimality check can be significantly simplified by following similar reasoning as the work by Flacco and De Luca (2013b), i.e., by considering that the eSNS is already able to output a task scaling factor close to 1 (its maximum). Thus, the optimality request on the scaling factor can be removed and the following simplified QP problem can be considered:

$$\begin{aligned}\min_{\boldsymbol{u}}\ & \frac{1}{2}(\boldsymbol{u}-\boldsymbol{u}_{r,k})^T\boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_{r,k}) \\ \text{s.t. }& \boldsymbol{A}_k\boldsymbol{u} = s_k\boldsymbol{b}'_k - \boldsymbol{b}''_k \\ & \boldsymbol{A}_{1\to k-1}\boldsymbol{u} = \boldsymbol{b}'^{*}_{1\to k-1} - \boldsymbol{b}''_{1\to k-1} \\ & \underline{\boldsymbol{d}}_{1\to k} \le \boldsymbol{C}_{1\to k}\boldsymbol{u} \le \overline{\boldsymbol{d}}_{1\to k}\end{aligned}, \tag{4.46}$$

where s_k is the scaling factor obtained from the eSNS. By setting

$$\boldsymbol{\xi} = \boldsymbol{u} - \boldsymbol{u}_{r,k}, \quad \boldsymbol{\Theta} = \boldsymbol{H}, \quad \boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{A}_k \\ \boldsymbol{A}_{1\to k-1} \end{bmatrix},$$
$$\boldsymbol{\beta} = \begin{bmatrix} s_k \boldsymbol{b}'_k - \boldsymbol{b}''_k - \boldsymbol{A}_k \boldsymbol{u}_{r,k} \\ \boldsymbol{b}'^{*}_{1\to k-1} - \boldsymbol{b}''_{1\to k-1} - \boldsymbol{A}_{1\to k-1}\boldsymbol{u}_{r,k} \end{bmatrix},$$
$$\boldsymbol{\Gamma} = \begin{bmatrix} -\boldsymbol{C}_{1\to k} \\ \boldsymbol{C}_{1\to k} \end{bmatrix}, \boldsymbol{\delta} = \begin{bmatrix} -\underline{\boldsymbol{d}}_{1\to k} + \boldsymbol{C}_{1\to k}\boldsymbol{u}_{r,k} \\ \overline{\boldsymbol{d}}_{1\to k} - \boldsymbol{C}_{1\to k}\boldsymbol{u}_{r,k} \end{bmatrix},$$

problem (4.46) can be brought to the standard form (4.39) and the same analysis based on the KKT conditions can be conducted as in the previous case. Again, the only condition that needs to be checked is the non-negativeness of the components of the multipliers $\boldsymbol{\mu}^{sat}$. These can be computed by following the same steps shown in the previous case. As a result, (4.45) becomes

$$\boldsymbol{\mu}^{sat} = -\Big(\boldsymbol{H}\Big(\boldsymbol{I} - \big(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}\big)^{\dagger H} \boldsymbol{A}_k \Big) \times \big(\boldsymbol{\Gamma}^{sat} \boldsymbol{P}_{k-1}\big)^{\dagger H} \Big)^T (\boldsymbol{u} - \boldsymbol{u}_{r,k}).$$

Moreover, by considering the particular structure of $\boldsymbol{\Gamma}$ and defining the auxiliary vector $\tilde{\boldsymbol{\mu}}^{sat}$ as

$$\tilde{\boldsymbol{\mu}}^{sat} = -\Big(\boldsymbol{H}\Big(\boldsymbol{I} - \big(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}\big)^{\dagger H} \boldsymbol{A}_k \Big) \times \big(\boldsymbol{C}^{sat}_{1\to k} \boldsymbol{P}_{k-1}\big)^{\dagger H} \Big)^T (\boldsymbol{u} - \boldsymbol{u}_{r,k}), \tag{4.47}$$

it is possible to compute the components of $\boldsymbol{\mu}^{sat}$ as

$$\mu^{sat,j} = \begin{cases} -\tilde{\mu}^{sat,j}, & \text{if } (\boldsymbol{c}^{sat,j}_{1\to k})^T \boldsymbol{u}_k = \underline{d}^{sat,j}_{1\to k} \\ \tilde{\mu}^{sat,j}, & \text{if } (\boldsymbol{c}^{sat,j}_{1\to k})^T \boldsymbol{u}_k = \overline{d}^{sat,j}_{1\to k} \end{cases},$$

with $j = 1, \ldots, l_{sat}$. Note that the evaluation of (4.47) requires very limited computational effort, since both $(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1})^{\dagger H}$ and $(\boldsymbol{C}^{sat}_{1\to k} \boldsymbol{P}_{k-1})^{\dagger H}$ are already computed inside the eSNS. Analogously, reintroducing the constraint selection matrix $\boldsymbol{W}_k$, it is also possible to define the auxiliary vector $\tilde{\boldsymbol{\mu}}$ as

$$\tilde{\boldsymbol{\mu}} = -\Big(\boldsymbol{H}\Big(\boldsymbol{I} - \big(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}\big)^{\dagger H} \boldsymbol{A}_k \Big) \times \Big(\big(\boldsymbol{I} - \boldsymbol{W}_k\big) \boldsymbol{C}_{1\to k} \boldsymbol{P}_{k-1} \Big)^{\dagger H} \Big)^T (\boldsymbol{u} - \boldsymbol{u}_{r,k}), \tag{4.48}$$

and then compute all the components of $\boldsymbol{\mu}$ as follows:

$$\begin{aligned}
&\mu^j = \mu^{j+l_k} = \tilde{\mu}^j = 0, \text{ if } w_k^{j,j} = 1 \\
&\mu^j = -\tilde{\mu}^j, \mu^{j+l_k} = 0, \text{ if } w_k^{j,j} = 0 \text{ AND } (\boldsymbol{c}_{1\to k}^j)^T \boldsymbol{u}_k = \underline{d}_{1\to k}^j \\
&\mu^j = 0, \mu^{j+l_k} = \tilde{\mu}^j, \quad \text{if } w_k^{j,j} = 0 \text{ AND } (\boldsymbol{c}_{1\to k}^j)^T \boldsymbol{u}_k = \overline{d}_{1\to k}^j,
\end{aligned} \tag{4.49}$$

with $j = 1, \ldots, l_k$ and $w_k^{j,j}$ being the jth element of the diagonal of $\boldsymbol{W}_k$. From (4.48) and (4.49), it is easier to see that the results of this section extend the work by Flacco and De Luca (2013b) and Flacco, De Luca, and Khatib (2015). Indeed, in the special case in which $\boldsymbol{H} = \boldsymbol{C}_{1\to k} = \boldsymbol{I} \in \mathbb{R}^{n\times n}, \boldsymbol{u}_{r,k} = \boldsymbol{0}\ \forall k = 1, \ldots, N$, (4.48) reduces to

$$\tilde{\boldsymbol{\mu}} = -\left(\left(\boldsymbol{I} - \left(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}\right)^{\dagger I} \boldsymbol{A}_k\right) \left((\boldsymbol{I} - \boldsymbol{W}_k)\, \boldsymbol{P}_{k-1}\right)^{\dagger I}\right)^T \boldsymbol{u},$$

which, imposing $\boldsymbol{A}_k = \boldsymbol{J}_k$ (velocity-based redundancy resolution), returns the auxiliary vector introduced by Flacco and De Luca (2013b).

Opt-eSNS

Following the analysis from the previous section, it is possible to develop a new variant of the eSNS algorithm, named *Opt-eSNS*, which can always guarantee optimality of the solution. The Opt-eSNS is derived from the basic one by simply adding the optimality check on $\boldsymbol{\mu}^{sat}$ after every iteration (see Algorithm 4.3). This allows constraints to also be removed from the saturation set, when the associated Lagrange multipliers are negative. In view of (4.47), the optimality check can be performed directly on the components of $\tilde{\boldsymbol{\mu}}^{sat}$. Furthermore, the possibility of removing constraints from the saturation set allows the initialization of $\boldsymbol{C}_{1\to k}^{sat}$ and $\boldsymbol{d}_{1\to k}^{sat}$ based on the result of the previous execution of the algorithm (*warm start*). Considering that the commands $\boldsymbol{b}_k$, $\underline{\boldsymbol{d}}_{1\to k}$, $\overline{\boldsymbol{d}}_{1\to k}$ are typically smooth, it is reasonable to assume that two consecutive solutions will have similar saturation sets at each level of priority. Thus, indicating with $\{sat\}_k^-$ the set of indices associated with the saturated commands of the kth priority level at the previous execution of the algorithm, it is possible to initialize $\boldsymbol{C}_{1\to k}^{sat}$ and $\boldsymbol{d}_{1\to k}^{sat}$ by extracting from $\boldsymbol{C}_{1\to k}$, $\underline{\boldsymbol{d}}_{1\to k}$ and $\overline{\boldsymbol{d}}_{1\to k}$ the rows and the components according to the indices in $\{sat\}_k^-$. This operation is carried out at lines 3–4 of Algorithm 4.3. Then, the set $\{sat\}_k^-$ is updated at line 26, after the solution for the kth priority level has been computed. This kind of initialization of the saturation set typically reduces the total number of iterations inside the algorithm, therefore resulting in a faster computation (see the results of Sect. 4.3.5-C). Algorithm 4.3 iterates until an optimal solution is found. Therefore, a more suitable name for the boolean variable used at lines 22 and 25 would be *non_optimal_solution*.

It is finally worth considering whether the adoption of the Opt-eSNS is preferable over the employment of state-of-the-art QP solvers, which can directly solve the problem (4.38) for each level of priority. First, it should be noticed that the dimension of the set of constraints significantly increases as the dimension of $\boldsymbol{A}_{1\to k-1}$ increases in (4.38) at each new priority level. This makes state-of-the-art QP solvers become slower as k increases. On the other hand, the use of null space projectors in the Opt-eSNS limits the dimension

Algorithm 4.3 Opt-eSNS algorithm for multiple priority levels

$\boldsymbol{P}_0 = \boldsymbol{I}$, $\boldsymbol{u}_0 = \boldsymbol{0}$
for $k = 1 \rightarrow N$ **do**
  $\boldsymbol{C}_{1\rightarrow k}^{sat} \leftarrow \boldsymbol{C}_{1\rightarrow k}^{\{sat\}_k^-}$
  $\boldsymbol{d}_{1\rightarrow k}^{sat} \leftarrow \boldsymbol{d}_{1\rightarrow k}^{\{sat\}_k^-}$
  $\hat{\boldsymbol{A}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}$
  $\hat{\boldsymbol{C}}_k = \boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{P}_{k-1}$
  $\boldsymbol{P}_k = \left(\boldsymbol{I} - \tilde{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{A}_k\right) \boldsymbol{P}_{k-1}$
  $\hat{\boldsymbol{P}}_{k-1} = \left(\boldsymbol{I} - \left(\boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{P}_{k-1}\right)^{\dagger H} \boldsymbol{C}_{1\rightarrow k}^{sat}\right) \boldsymbol{P}_{k-1}$
  $\hat{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \left(\boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1}\right)^{\dagger H} \boldsymbol{A}_k\right) \hat{\boldsymbol{P}}_{k-1}$
  $\boldsymbol{u}_{sat} = \hat{\boldsymbol{C}}_k^{\dagger H} \left(\boldsymbol{d}_{1\rightarrow k}^{sat} - \boldsymbol{C}_{1\rightarrow k}^{sat} \boldsymbol{u}_{k-1}\right)$
  $\boldsymbol{u}' = \hat{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{b}_k'$
  $\boldsymbol{u}'' = \hat{\boldsymbol{A}}_k^{\dagger H} \left(-\boldsymbol{b}_k'' - \boldsymbol{A}_k \left(\boldsymbol{u}_{k-1} + \boldsymbol{u}_{sat}\right)\right)$
  $\boldsymbol{u}''' = \hat{\boldsymbol{P}}_k \boldsymbol{u}_{r,k}$
  $s_k^* = 0$
  repeat
    ...(same code as Alg. 4.1)
    $\tilde{\boldsymbol{\mu}}^{sat} = -\left(\boldsymbol{H}\left(\boldsymbol{I} - \hat{\boldsymbol{A}}_k^{\dagger H} \boldsymbol{A}\right) \hat{\boldsymbol{C}}_k^{\dagger H}\right)^T \left(\boldsymbol{u} - \boldsymbol{u}_{r,k}\right)$
    for $j = 1 \rightarrow l_{sat}$ **do**
      if $\begin{pmatrix} (\boldsymbol{c}_{1\rightarrow k}^{sat,j})^T \boldsymbol{u}_k = \underline{d}_{1\rightarrow k}^{sat,j} \text{ AND } \tilde{\mu}^{sat,j} > 0 \\ \text{OR} \\ (\boldsymbol{c}_{1\rightarrow k}^{sat,j})^T \boldsymbol{u}_k = \overline{d}_{1\rightarrow k}^{sat,j} \text{ AND } \tilde{\mu}^{sat,j} < 0 \end{pmatrix}$ **then**
        $\{sat\}_k \leftarrow \{sat\}_k \backslash \{j\}$
        limits_violated = TRUE
      end if
    end for
  until limits_violated = FALSE
  $\{sat\}_k^- = \{sat\}_k$
end for
$\boldsymbol{u} = \boldsymbol{u}_N$

of the matrices to be (pseudo-)inverted, typically resulting in faster computation times for numeric results). Furthermore, although it is normally possible to find a good range of values, the choice of M might result not to be trivial and dependent on the specific problem to solve. Values that are not big enough could lead to conservative scale factors. On the contrary, too large values can undermine the numerical stability of the solver and lead to unfeasible problems. Not being dependent on such parameter is then an additional benefit of the proposed Opt-eSNS algorithm.

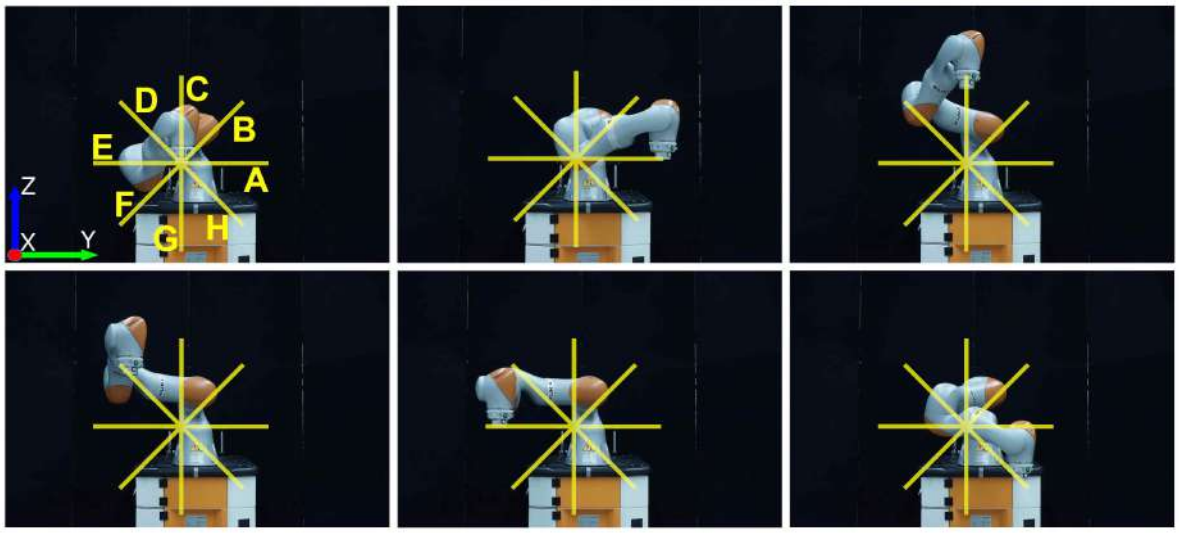


Figure 4.2: LBR iiwa moving on a star-like path during the experiments of Sect. 4.3.5-A. The path is defined on the YZ plane. Each star segment (indicated with a capital letter) has a length of 24 cm.

4.3.5 Results

This section reports experiments and simulations to numerically support the effectiveness of the proposed framework in solving (prioritized) redundancy resolution problems defined at velocity, acceleration or torque level. The benefits of the variants introduced in Sect. 4.3.4 are also highlighted, compared to basic eSNS algorithm. The presented results are obtained through experiments carried out on a KUKA LBR iiwa 7-DoFs robot (Fig. 4.2), as well as a highly-redundant mobile dual-arm system (Fig. 4.13).

A. First set of experiments with the LBR iiwa robot

In the first set of experiments the LBR iiwa performs repeatedly the same tasks with the basic eSNS solver working at velocity, acceleration and torque level. In the first two cases the output of the solver is integrated to obtain a joint position reference, which is then provided to a low-level position controller; in the last case the output of the algorithm is directly fed to the joint actuators and an actual control loop is implemented. All the algorithms run as real-time module in a VxWorks® (32 bit) environment on one of the four cores of an Intel Core™i5-45705 (2.89 GHz) CPU with 432 MB of dedicated RAM. The control cycle time is 1 ms.

The first task is to track a given desired Cartesian trajectory with the center point of the robot end-effector (equality constraint). Figure 4.2 shows the Cartesian path used for the experiments. The robot is commanded to move on each segment of the star, returning every time to the center point, following a sinusoidal velocity profile. The total planned time is 12 seconds (1.5 seconds per segment). The initial end-effector orientation must be kept along the entire trajectory. Therefore, both Cartesian position and orientation of the robot are controlled and only one redundant DoF is available.

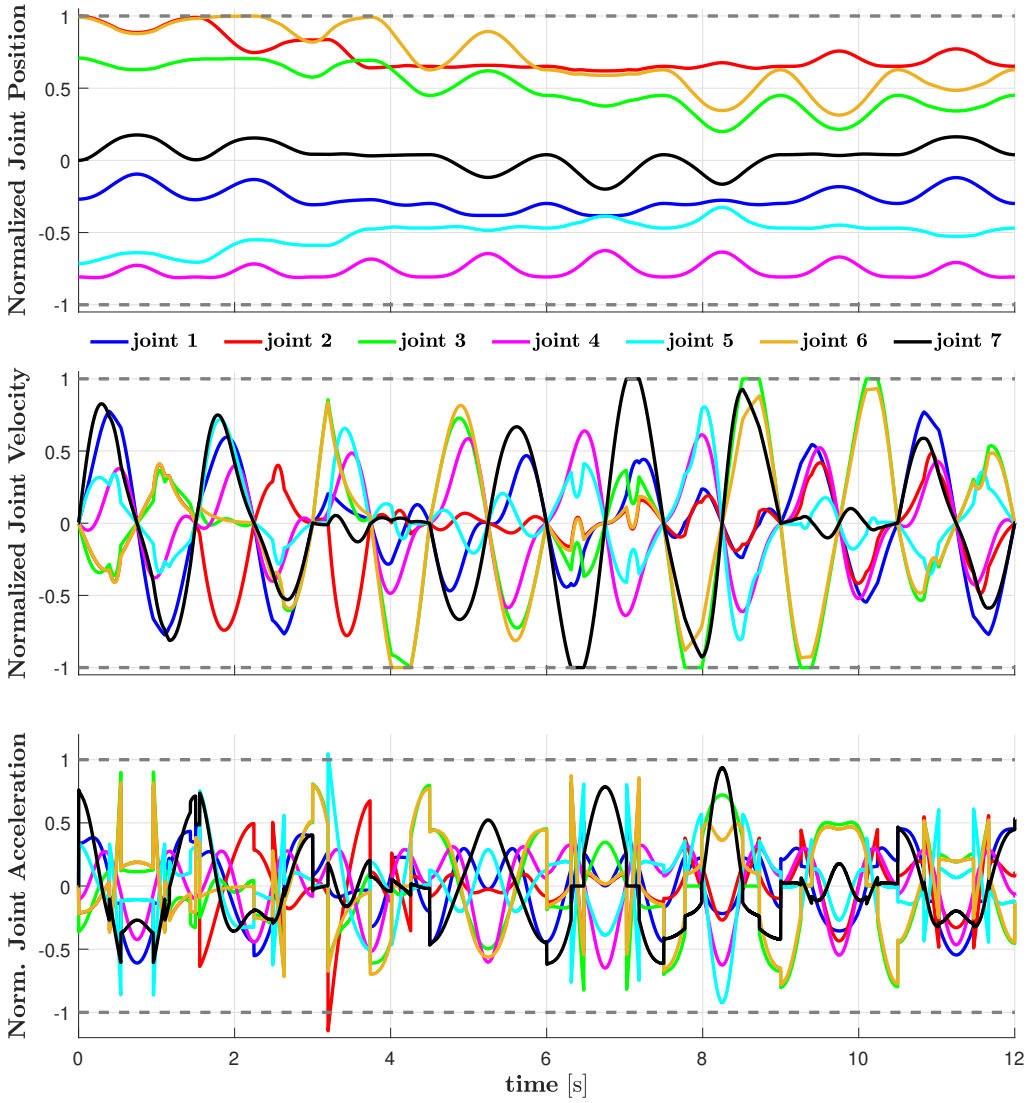


Figure 4.3: Normalized joint position, velocity and acceleration produced by the velocity-based eSNS solver in the experiments of Sect. 4.3.5-A.

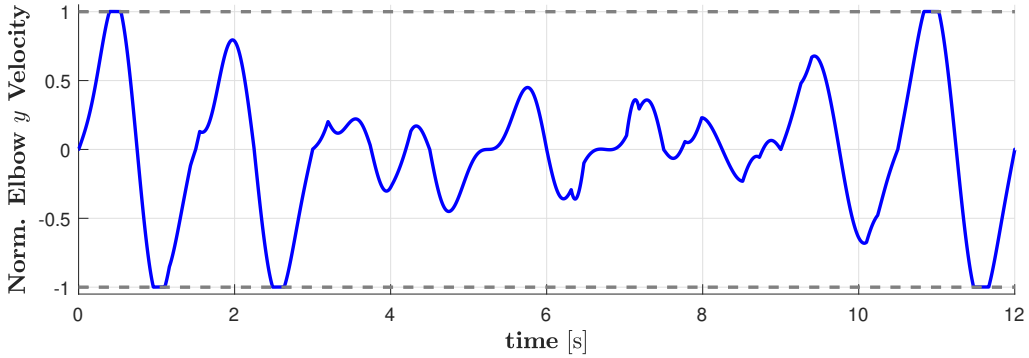


Figure 4.4: Normalized elbow velocity in y-direction produced by the velocity-based eSNS solver in the experiments of Sect. 4.3.5-A.

Joint nr.	Position Lim. [rad]	Velocity Lim. [rad/s]	Acceler. Lim. [rad/s^2]	Initial Pos. [rad]
1	± 2.9234	± 1.45	± 9	-0.7854
2	± 2.0508	± 1.45	± 9	$+2.0502$
3	± 2.9234	± 1.45	± 9	$+2.0721$
4	± 2.0508	± 1.45	± 9	-1.6563
5	± 2.9234	± 1.45	± 9	-2.0893
6	± 2.0508	± 1.45	± 9	$+2.0342$
7	± 3.0107	± 1.45	± 9	0

Table 4.3: Initial configuration and joint limits for the LBR iiwa. The first joint is at the base of the robot.

The second task is to keep the velocity of the center point of the robot elbow along the direction of the y-axis below the limit of $0.35\,\mathrm{m/s}$ (inequality constraint). Additional limitations on joint position, velocity and acceleration are considered, which can be seen in Tab. 4.3. These limitations will originate a set of additional (joint space) inequality constraints for the solver.

The two considered tasks are handled using one level of priority, meaning that the inequalities regarding joint and elbow limitations are stacked and treated as one set of constraints. The initial configuration of the robot, which can be seen in the top-left corner of Fig. 4.2, places the second and the sixth joint very close to their respective upper position limit (see Tab. 4.3).

The velocity-based eSNS solver uses the velocity bounds in (4.9) with $\boldsymbol{K} = 10\boldsymbol{I}$, $\boldsymbol{H} = \boldsymbol{M}$ and $\boldsymbol{u}_r = \boldsymbol{0}$. The task reference velocity is defined as in (4.5), with $\boldsymbol{K} = 50\boldsymbol{I}$. The output joint velocities are shown in Fig. 4.3 and have been integrated and differentiated, so as to obtain the corresponding joint position and acceleration. For the sake of clarity, all the values are reported after a normalization that brings the allowed ranges defined in Tab. 4.3 to the interval $[-1, 1]$. Figure 4.4 shows the resulting elbow velocity along the direction of the y-axis. Also this signal is reported normalized with respect to to its allowed range. The intensive occurrence of saturation can be easily identified in Fig. 4.3 and Fig. 4.4, proving the effectiveness of the proposed algorithm in respecting the hard bounds imposed by the inequality constraints in both joint and task space. Furthermore, the task scaling factor remains constant and equal to 1, indicating that the end-effector task remains feasible during the entire motion (Fig. 4.9). On the other hand, Figure 4.3 also shows violation of the joint acceleration limits. The fulfillment of such limits cannot be guaranteed by the velocity bounds in (4.9).

The acceleration-based eSNS solver uses $\boldsymbol{H} = \boldsymbol{M}$ and the acceleration bounds in (4.16), with $\boldsymbol{D} = 40\boldsymbol{I}$ and $\boldsymbol{K}_1 = 10$. The reference task acceleration is defined as in (4.6), with $\boldsymbol{K} = 400\boldsymbol{I}$ and $\boldsymbol{D} = 40\boldsymbol{I}$. Furthermore, the choice $\boldsymbol{u}_r = -k_{damp}\dot{\boldsymbol{q}}(t)$ (with

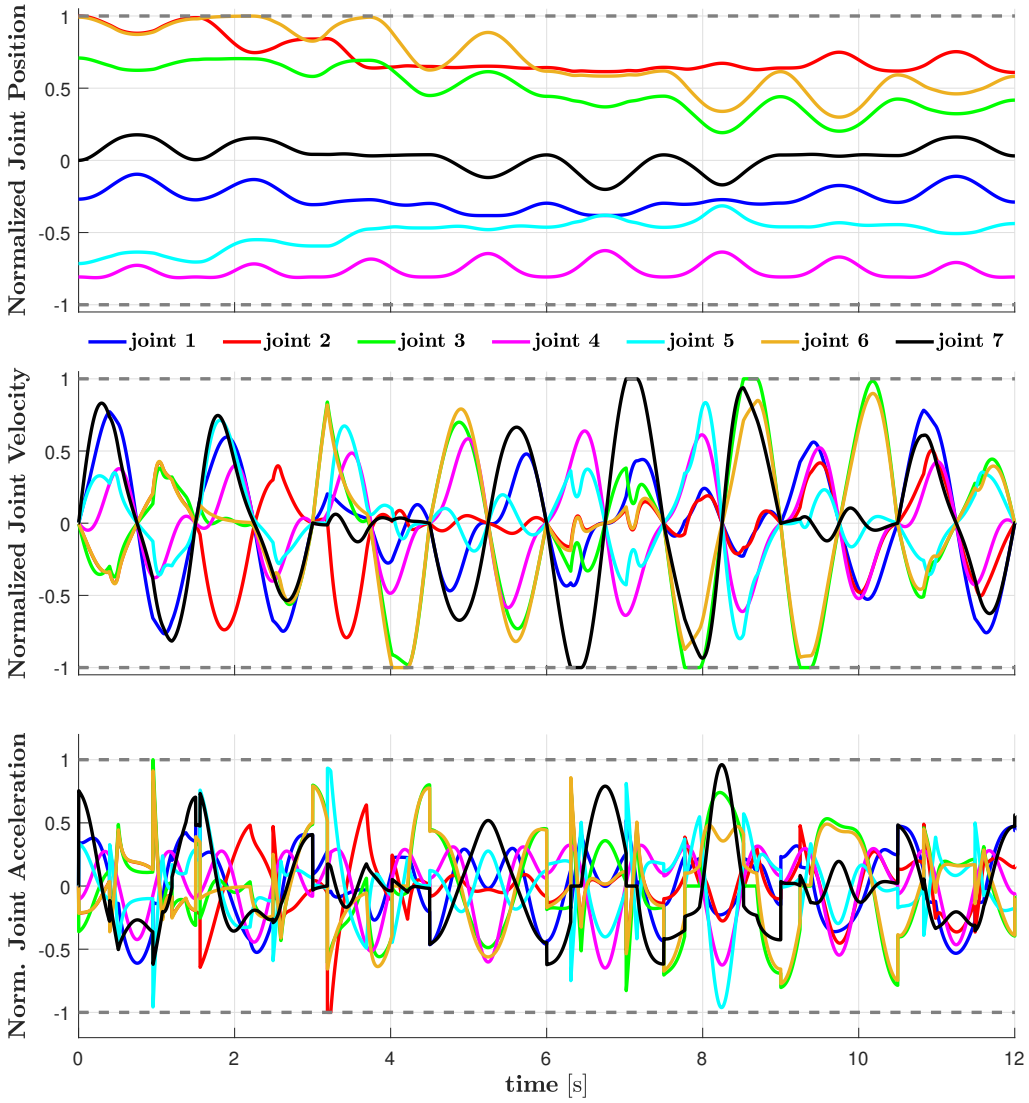


Figure 4.5: Normalized joint position, velocity and acceleration produced by the acceleration-based eSNS solver in the experiments of Sect. 4.3.5-A.

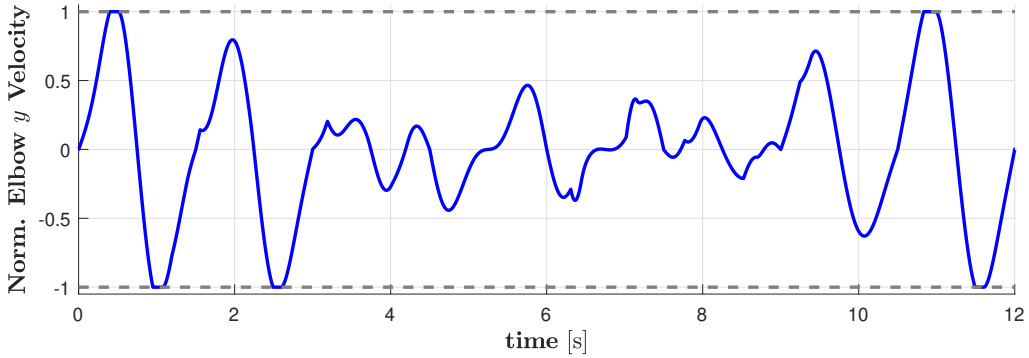


Figure 4.6: Normalized elbow velocity in y-direction produced by the acceleration-based eSNS solver in the experiments of Sect. 4.3.5-A.

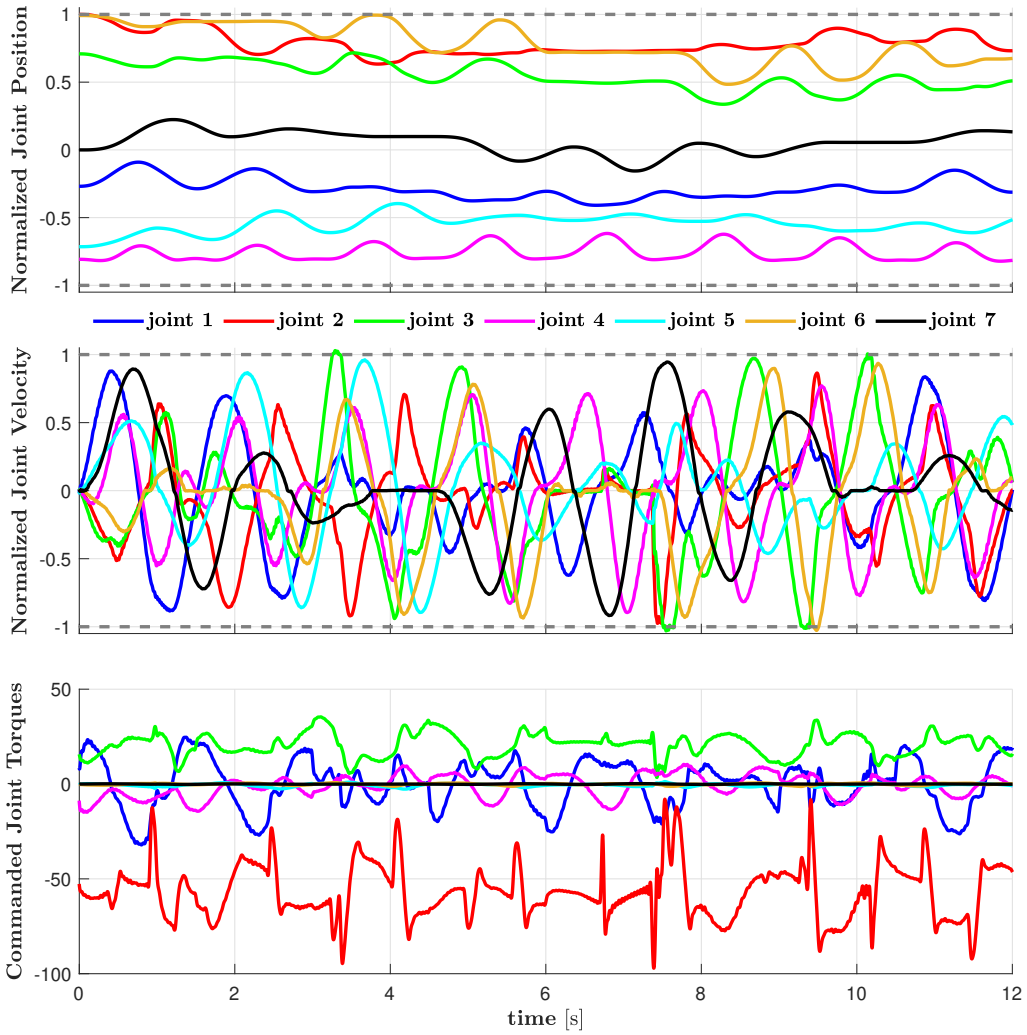


Figure 4.7: Joint motion produced by the torque-based eSNS solver in the experiments of Sect. 4.3.5-A; since a measure of the joint acceleration is not available, only (normalized) joint position and velocity are reported alongside the commanded joint torques, which are the actual output of the solver.

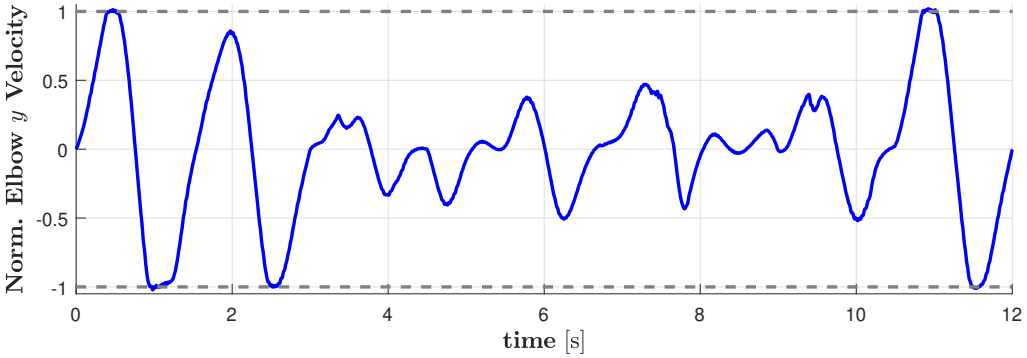


Figure 4.8: Normalized elbow velocity in y-direction produced by the torque-based eSNS solver in the experiments of Sect. 4.3.5-A.

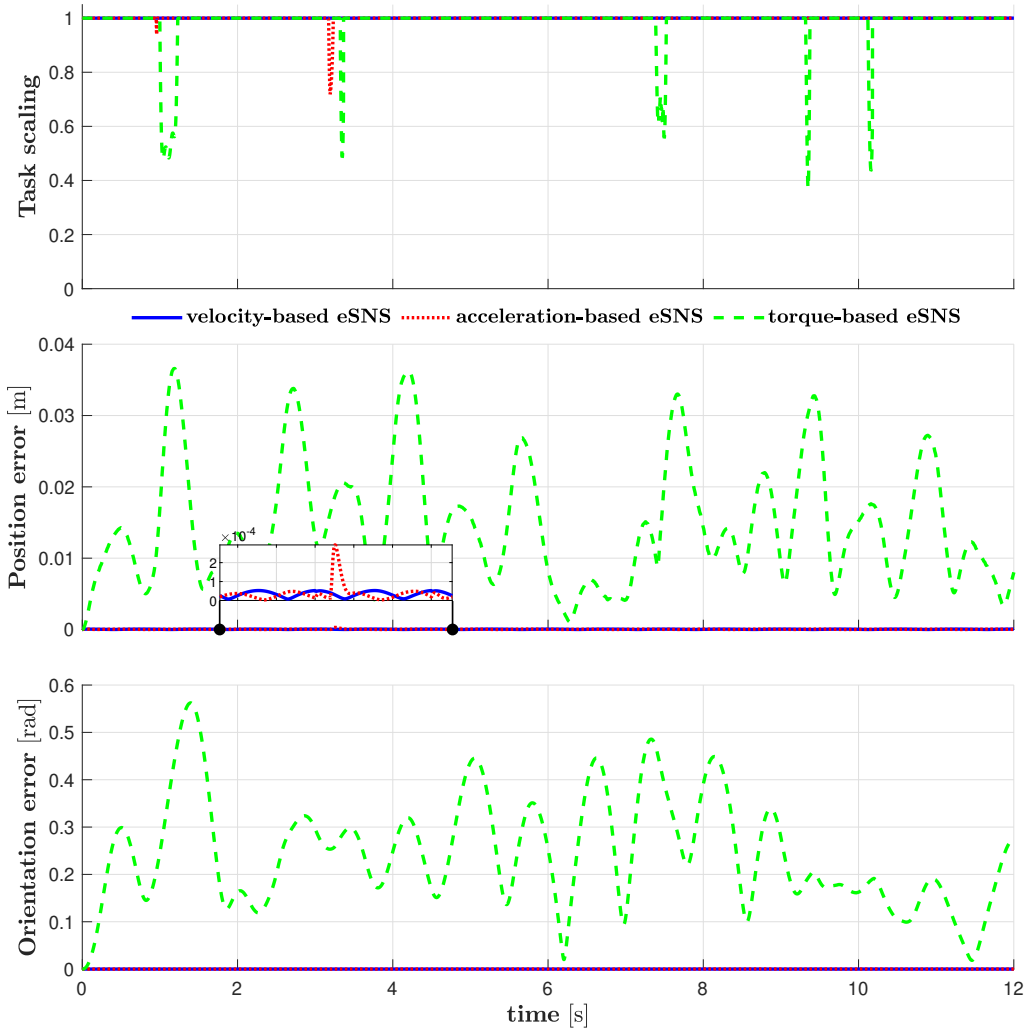


Figure 4.9: Task scaling factor (above) and Cartesian tracking error for the experiments of Sect. 4.3.5-A; position error (in norm) is presented in the center; orientation error, reported as the angle extracted from the quaternion error, is reported below.

k_{damp} a positive gain set to 20) is used to damp null space motions. The generated joint motion is shown in Fig. 4.5. Compared to Fig. 4.3, the constraint on the maximum joint acceleration is now fulfilled. On the other hand, it is more difficult for the algorithm to find a feasible solution throughout the Cartesian end-effector trajectory and a task scaling smaller than one is observed in some cases (Fig. 4.9). The elbow velocity along the y-direction is also reported (Fig. 4.6).

The torque-based eSNS solver uses the same task reference and bounds as the acceleration controller. Damping of null space motion is again included, this time with the choice $\boldsymbol{u}_r = -\boldsymbol{M}(k_{damp}\dot{\boldsymbol{q}}(t))$, with $k_{damp} = 20$. The choice $\boldsymbol{H} = \boldsymbol{M}^{-1}$ should produce the same joint motion as the acceleration-based solver. However, inaccuracy in the estimation of the complete robot dynamics results in a different motion (see Fig. 4.7). For the same reason, small inaccuracies can be noticed on the joint velocity saturation, as well as on the saturation of the elbow velocity along the y-direction (see Fig. 4.8). A

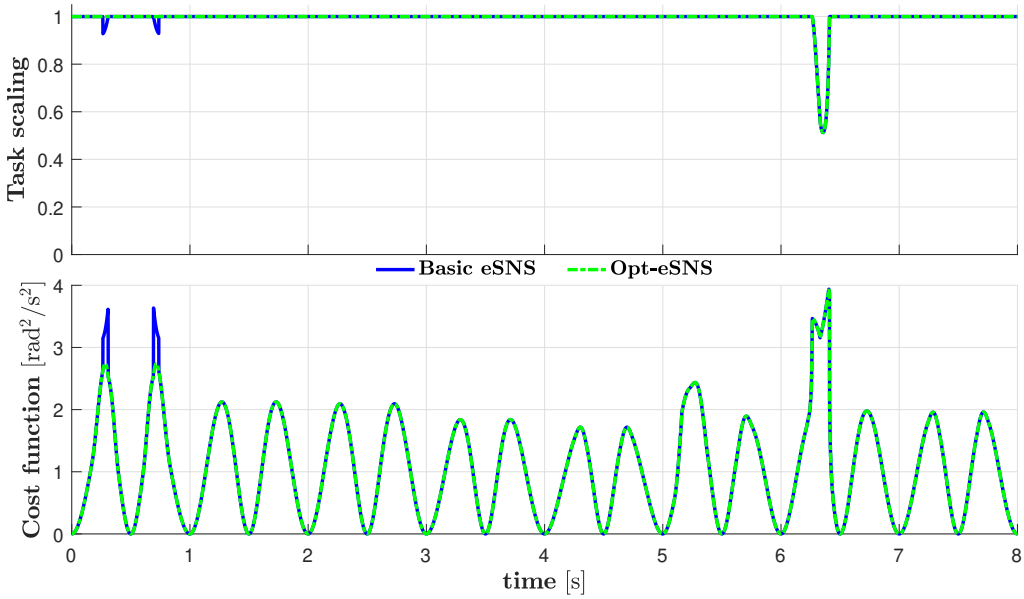


Figure 4.10: Task scaling factors (above) and cost function values (below) produced by the different eSNS variants in the experiments of Sect. 4.3.5-B.

different trend of the task scaling factor is also observed in Fig. 4.9. For the sake of completeness, Fig. 4.7 also reports the trend of the commanded joint torques, which are the actual output of the solver.

The tracking of the Cartesian end-effector trajectory is also poorer when the robot is torque-controlled (Fig. 4.9). In the other cases, the tracking error is considerably smaller and increases only when a task scaling $s < 1$ is observed, as visible in the enlargement of the center plot.

B. Second set of Experiments with the LBR iiwa robot

The main goal of the second set of experiments is to highlight the benefits of the Opt-eSNS, compared to the basic eSNS algorithm. The experimental setup presents the same computing hardware and control cycle time as Sect. 4.3.5-A, while the task definition has been slightly modified. Indeed, the LBR iiwa is commanded again to track the Cartesian trajectory described in Sect. 4.3.5-A with its end-effector center point. However, this time the orientation is not constrained, leaving four redundant DoFs available. Moreover, the total planned time is reduced to 8 seconds (1 second per each star segment) and the initial joint configuration is changed to $\boldsymbol{q}_0 = [-0.9 \;\; 2 \;\; 2.1721 \;\; -1.8055 \;\; -2.0893 \;\; 1.9834 \;\; 0]^T$ rad. The same inequality constraints (elbow velocity along the direction of the y-axis and joint limits) of Sect. 4.3.5-A are instead considered. All the tasks (equality and inequality constraints) are handled using one level of priority and solved using the velocity-based solver with $\boldsymbol{H} = \boldsymbol{I}$ and $\boldsymbol{u}_r = \boldsymbol{0}$. Velocity bounds are computed according to (4.9) using $\boldsymbol{K} = 10\boldsymbol{I}$, whereas task references are computed according to (4.5), with $\boldsymbol{K} = 50\boldsymbol{I}$. This specific setup has been chosen because it clearly shows the non optimality of the solution returned by the basic eSNS. Indeed, the basic eSNS and Opt-eSNS show quite different results when used to perform

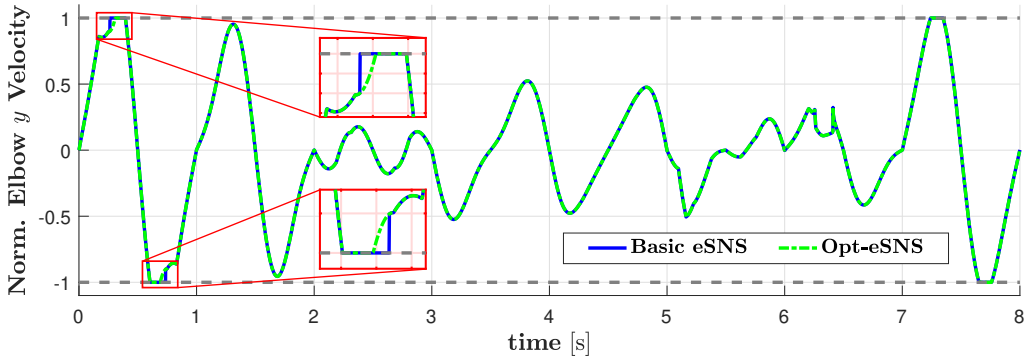


Figure 4.11: Normalized elbow velocity in y-direction produced by the different eSNS variants in the experiments of Sect. 4.3.5-B.

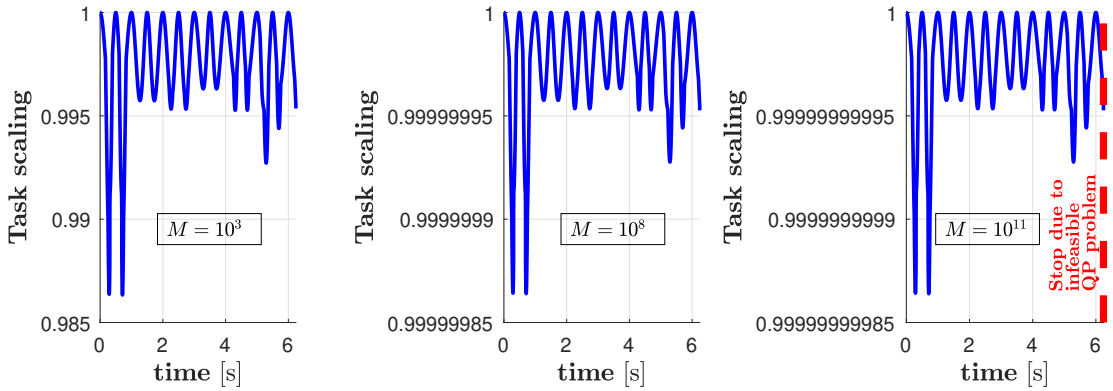


Figure 4.12: Task scaling factor produced by the state-of-the-art QP solver as M changes in the experiments of Sect. 4.3.5-B. For a meaningful comparison, only the first 6.2 seconds of motion are considered in every case.

the assigned task. The disparity between the two solutions comes from the ability of the Opt-eSNS of removing constraints from the (optimal) saturation set. Figure 4.10 reports the task scaling factors and cost function ($\frac{1}{2}(\boldsymbol{u}-\boldsymbol{u}_r)^T \boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_r)$) values over time obtained by the two algorithms. The difference in the results is especially visible in the first second of motion and it is highly reflected on the trend of the elbow velocity along the y direction. As visible in Fig. 4.11, the non-optimal saturation sets obtained through the basic eSNS lead to sudden variations in the first second of motion, whereas the Opt-eSNS returns smoother motions.

To validate the optimality of the results of the Opt-eSNS, the obtained solution has been compared with the one returned by a state-of-the-art QP solver (qpOASES by Ferreau, Kirches, et al. (2014)), operating on the same set of constraints and the cost function in (4.38). The average task scaling factor and the average value of $\frac{1}{2}(\boldsymbol{u}-\boldsymbol{u}_r)^T \boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_r)$ (which in this special case coincides with the joint velocity norm function) are reported in Tab. 4.4 for the different algorithms used in the experiments of this section. It can be noticed that the Opt-eSNS provides both a higher average task scaling factor and a lower cost function value in the considered experiment, compared to the basic eSNS algorithm.

Algorithm	Task scaling factor s	$\frac{1}{2}(\boldsymbol{u}-\boldsymbol{u}_r)^T \boldsymbol{H}(\boldsymbol{u}-\boldsymbol{u}_r)$ [rad^2/s^2]
Basic e-SNS	0.9939	1.0533
Opt e-SNS	0.9944	1.0459
qpOases ($M = 10^3$)	0.9917	1.0443
qpOases ($M = 10^8$)	0.9944	1.0459

Table 4.4: Average task scaling factor and cost function value for the experiments of Sect. 4.3.5-B.

Finally, the importance of choosing a good value for the parameter M when using a state-of-the-art QP solver should be remarked, as opposed to the absence of such parameter in the Opt-eSNS. Values of M that are too small could lead to non optimal solutions. In the proposed experiments, e.g., a value of $M = 10^3$ returned worse results than the basic eSNS in terms of average task scaling factor (see Tab. 4.4). On the other hand, bigger values of M may lead to numerical instability and, thus, to unfeasible QP problems (Fig. 4.12).

C. Simulations with the mobile dual-arm system

This section presents the results of a set of simulations involving a mobile dual arm manipulator with 17 DoFs (Fig. 4.13): the mobile base is equipped with omnidirectional wheels and it is therefore modeled as a sequence of two prismatic joints and a revolute one, all located at the center of the base; additionally, two LBR iiwa robots are mounted on top of the mobile base. The main objective of the simulations is to prove the effectiveness of the proposed algorithms in handling multiple levels of priority. Moreover, the performance of the different eSNS variants is evaluated. The simulations are carried out in MATLAB® environment running on an Intel® Core™i7-8850H (2.60GHz) CPU with 16 GB of RAM.

The tasks to execute are distributed on three levels of priority. At the first level, the center point of the mobile base (yellow point in Fig. 4.13) is required to track a desired one-dimensional trajectory along the direction of the Y axis. The trajectory starts at $y = 0$ and ends in $y = 0.5$, following a trapezoidal velocity profile. The total planned time is 8 seconds. The top row of Fig. 4.13 shows the desired y trajectory (drawn for a fixed value $x = 0$) as a red segment. Additionally, limitations on the position and the velocity of each joint are considered: the limit values used for the mobile base are reported in Tab. 4.5, whereas the limits from Tab. 4.3 are used for the two robotic arms. The position limits of the first two joints are represented by the blue rectangle in Fig. 4.13.

At the second priority level, the left arm is required to track a desired three-dimensional positional Cartesian trajectory with its end-effector center point. The trajectory consists in the star-like path with sinusoidal velocity profile introduced in Sect. 4.3.5-A. The total

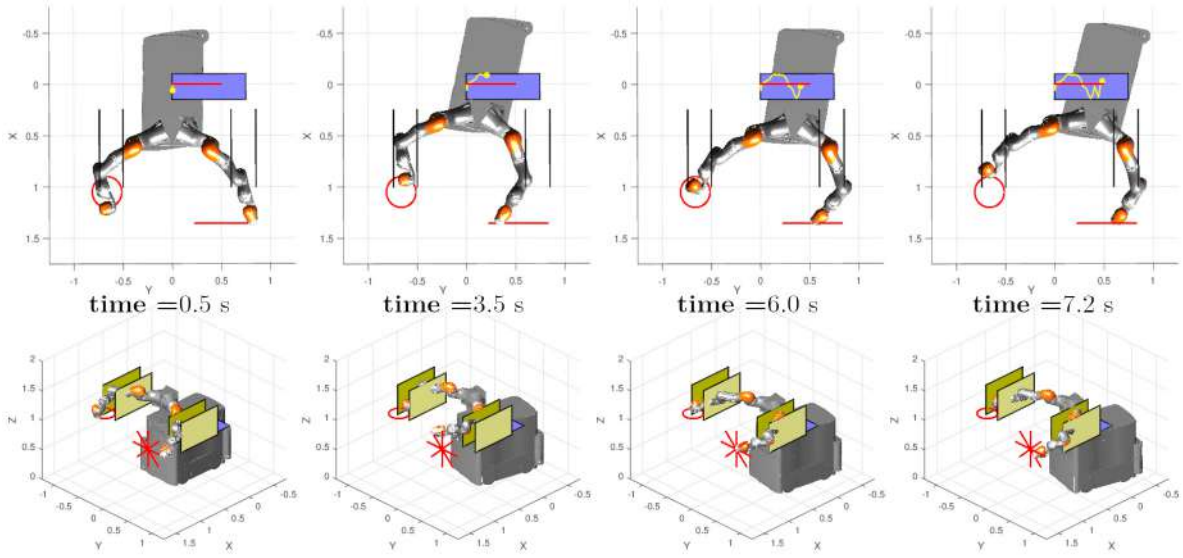


Figure 4.13: Motion sequence of the mobile dual-arm robot in the simulation of Sect. 4.3.5-C. The top row shows a top view of the motion, whereas the bottom row offers a side view.

	Lower Position Limit	Upper Position Limit	Velocity Limit	Initial Position
Mobile Base Joint 1	$-0.1\,\mathrm{m}$	$0.15\,\mathrm{m}$	$\pm 0.5\,\mathrm{m/s}$	$0\,\mathrm{m}$
Mobile Base Joint 2	$0\,\mathrm{m}$	$0.75\,\mathrm{m}$	$\pm 0.5\,\mathrm{m/s}$	$0\,\mathrm{m}$
Mobile Base Joint 3	$-0.5\,\mathrm{rad}$	$0.5\,\mathrm{rad}$	$\pm 0.5\,\mathrm{rad/s}$	$0\,\mathrm{rad}$
Left Elbow y Coord.	$0.6\,\mathrm{m}$	$0.85\,\mathrm{m}$	$+\inf\,\mathrm{m/s}$	$0.6216\,\mathrm{m}$
Right Elbow y Coord.	$-0.75\,\mathrm{m}$	$-0.5\,\mathrm{m}$	$-\inf\,\mathrm{m/s}$	$-0.7127\,\mathrm{m}$

Table 4.5: Limits and initial positions for the joints of the mobile base and the elbow center points for the simulations of Sect. 4.3.5-C.

planned time is again 8 seconds (1 second per star-segment). Moreover, limitations on the position along the y-axis are imposed on the left elbow center point. These limitations are represented by the yellow planes in Fig. 4.13, while the numeric values of the lower and upper position limit are reported in Tab. 4.5.

Priority Level	Inequality Constraints	Equality Constraints
1	Joint position and velocity limits (dim. 17)	Desired trajectory for the mobile base y coordinate (dim. 1)
2	Position limits on left elbow y coordinate (dim. 1)	Desired trajectory for the left end effector center point (dim. 3)
3	Position limits on left elbow y coordinate (dim. 1)	Desired trajectory for the right end effector center point (dim. 3)

Table 4.6: Task hierarchy for the simulations of Sect. 4.3.5-C.

On the third level of priority, a three-dimensional positional Cartesian trajectory is assigned to the end-effector center point of the right arm. The trajectory consists of a circular path defined in the XY-plane (center $c_0 = [1.0496 \ -0.6635 \ 1.0690]$ m, radius $r_0 = 0.15$ m) with a trapezoidal velocity profile. The planned time to cover the circular path is also 8 seconds. Moreover, limitations on the position along the y-axis are imposed on the right elbow center point. As for the left arm, such limitations are represented by yellow planes in Fig. 4.13, while the numeric values of the lower and upper position limits are reported in Tab. 4.5. An overview of the considered set of tasks is given in Table 4.6.

The tasks are solved by using the basic eSNS, operating at velocity level with $\boldsymbol{H} = \boldsymbol{I}$ and $\boldsymbol{u}_r = \boldsymbol{0}$. All the velocity bounds are computed according to (4.9), with $\boldsymbol{K} = 10\boldsymbol{I}$, whereas all task references are computed according to 4.5 using $\boldsymbol{K} = 100\boldsymbol{I}$. The initial joint configuration of the mobile platform is reported in Tab. 4.5, while left and right arms start from $\boldsymbol{q}_{0,l}$ and $\boldsymbol{q}_{0,r}$, respectively, with

$$\begin{aligned} \boldsymbol{q}_{0,l} &= \begin{bmatrix} -1.2 & -0.57 & 0 & 0.78 & 0 & 0 & 0 \end{bmatrix}]^T \text{ rad} \\ \boldsymbol{q}_{0,r} &= \begin{bmatrix} -1.9 & 0.27 & 0 & -0.98 & 1.8 & -1.2 & 0 \end{bmatrix}^T \text{ rad} \end{aligned}.$$

The control cycle time used in the simulation is 1 ms.

The trend of the joint position and velocity over time is reported in Fig. 4.14, whereas Fig. 4.15 shows the normalized position along the y-axis of the elbow center points; all the quantities are reported normalized with respect to their respective admissible range of motion. Finally, Fig. 4.16 reports the trend of the scale factors and the tracking errors for each priority level.

The intensive saturation of different joint and task space variables can be noticed in Fig. 4.14 and 4.15. Having the highest priority, joint limits are never violated. Satisfying the equality constraint of the first priority level is also possible, as shown by the constant task scaling factor (equal to 1) and the small tracking error in Fig. 4.16. On the other hand, the demanding task specification produces task scaling on the second (from time

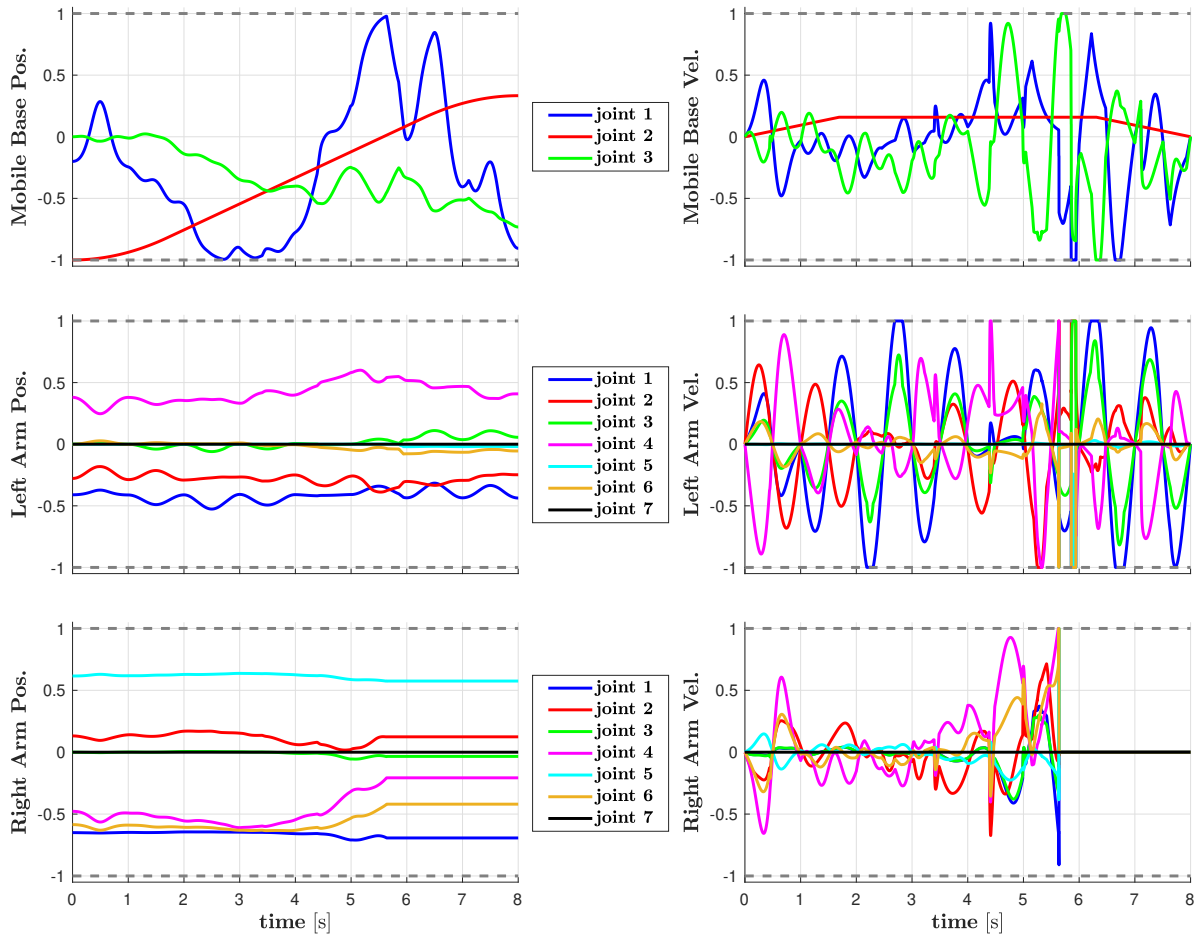


Figure 4.14: Normalized joint position and velocity produced by the velocity-based eSNS solver in the Simulation of Sect. 4.3.5-C. Joint motion is reported separately for the mobile base (top), the left arm (center) and the right arm (bottom).

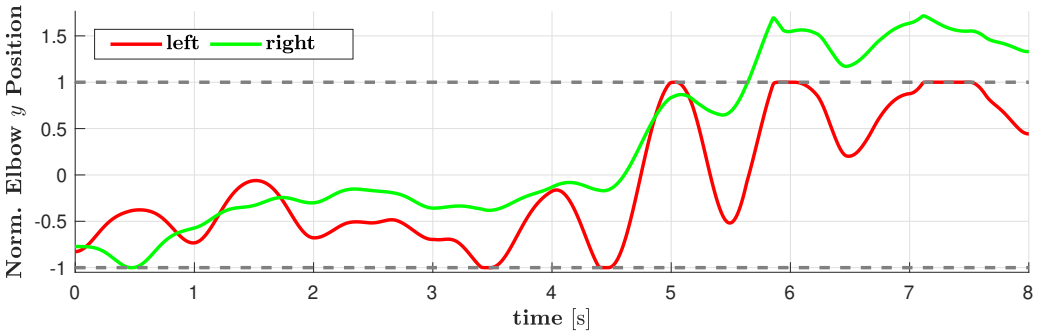


Figure 4.15: Normalized elbow positions in y-direction produced by the velocity-based eSNS solver in the Simulation of Sect. 4.3.5-C.

$t = 5.86$ s until $t = 5.940$ s) and third (from $t = 5.634$) priority level. More specifically, the task scaling factor of third task rapidly decreases to zero as the related constraints become incompatible with the task hierarchy. As a consequence, the task is completely

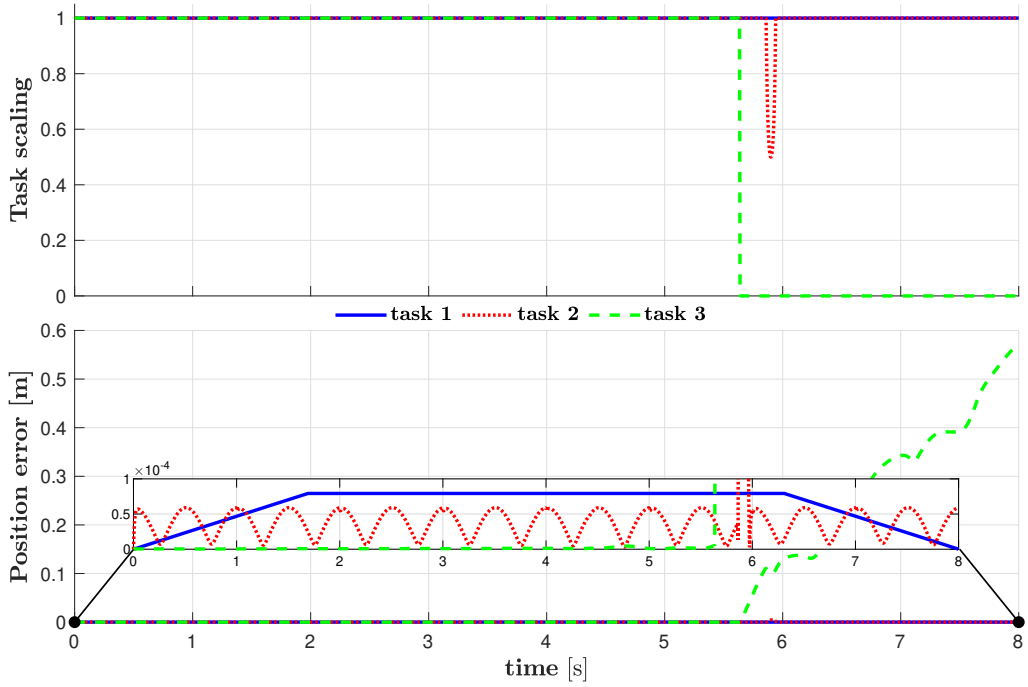


Figure 4.16: Task scaling factors (above) and norm of Cartesian tracking position errors (below) produced by the velocity-based eSNS solver in the simulation of Sect. 4.3.5.-C

sacrificed, leading to violation of the right elbow position limit (Fig. 4.15) and an increase of the error in tracking the circular trajectory (Fig. 4.16). It is worth noticing that at each priority level the tracking error remains limited throughout the entire motion and increases only when the corresponding task scaling factor becomes smaller than 1. This can be easily noticed in the enlargement of the bottom plot in Fig. 4.16.

To evaluate the performance of the different eSNS variants, the same simulation has been repeated using the Fast-eSNS, the Opt-eSNS and the Opt-eSNS with warm start. The joint motion produced by the Fast-eSNS is, as expected, identical to the one obtained with the basic eSNS and it is therefore not reported. The main advantage of this variant lies in its computational efficiency. This can be appreciated in Fig. 4.17, where the total number of iterations and the execution time of each eSNS variant are reported. It can be easily noticed that the Fast-eSNS always provides faster computation compared to the basic eSNS, with the difference on the execution time that is as significant as the number of total iterations increases. The Opt-eSNS returns, in light of the optimality check performed inside the algorithm, slightly different solutions. This can be directly noticed in Fig. 4.18, which reports the velocity of the joints that present a more significant difference, compared to the solution obtained with the basic eSNS. Moreover, differences can also be noticed in the number of total iterations performed by the algorithm (Fig. 4.17), which is in some cases higher than the basic eSNS due to the removal of constraints from the saturation set. Figure 4.17 also shows that the Opt-eSNS presents computation times comparable to the basic eSNS, proving that the computational burden added by the optimality check is very limited. Since (as expected) the saturation sets do not change so much and/or so often between two consecutive instants of time, the total number

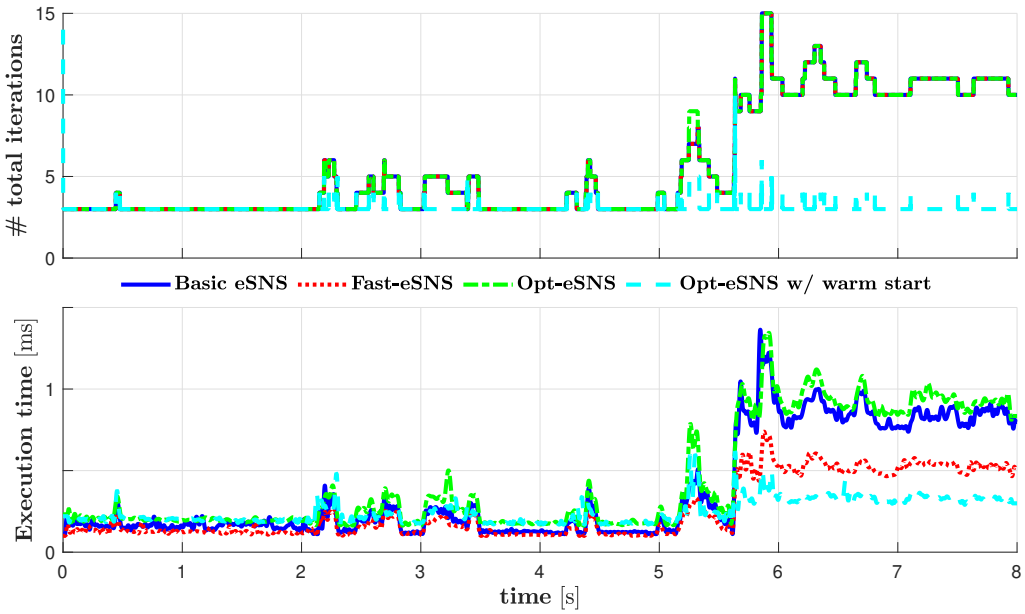


Figure 4.17: Number of total iterations (top plot) and execution time (bottom plot) of the different eSNS variants for the simulations of Sect. 4.3.5-C.

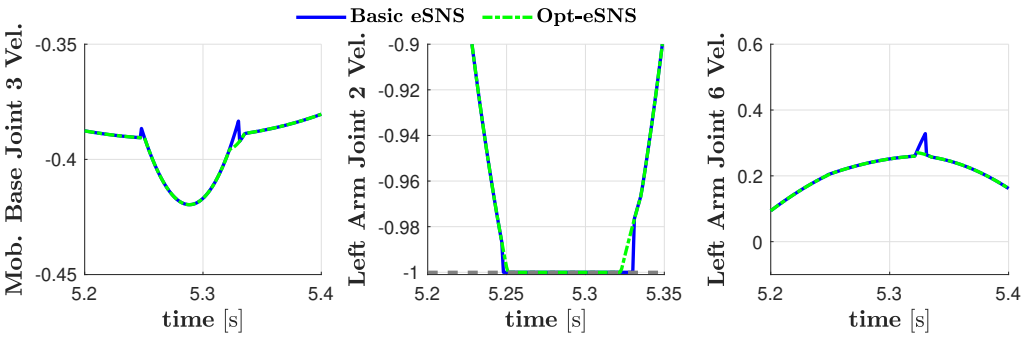


Figure 4.18: Comparison between the joint velocity obtained with the Opt-eSNS and the basic eSNS in the simulations of Sect. 4.3.5-C; for brevity, only the joints presenting significant differences are reported.

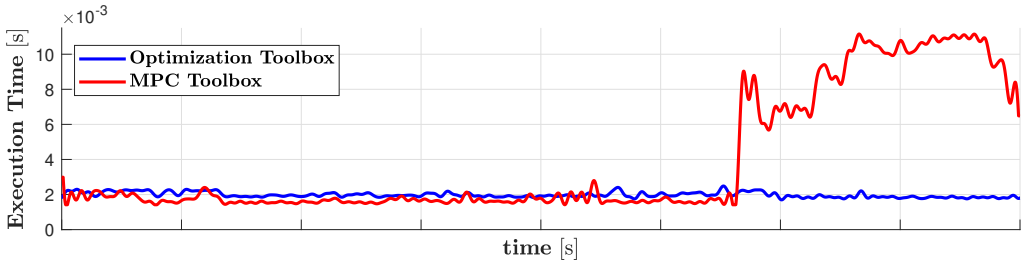


Figure 4.19: Execution time of the state-of-the-art QP solvers in the simulations of Sect. 4.3.5-C.

of iterations is dramatically reduced when using the Opt-eSNS with warm start. As a consequence, the execution time is sometimes comparable or even lower than the Fast-

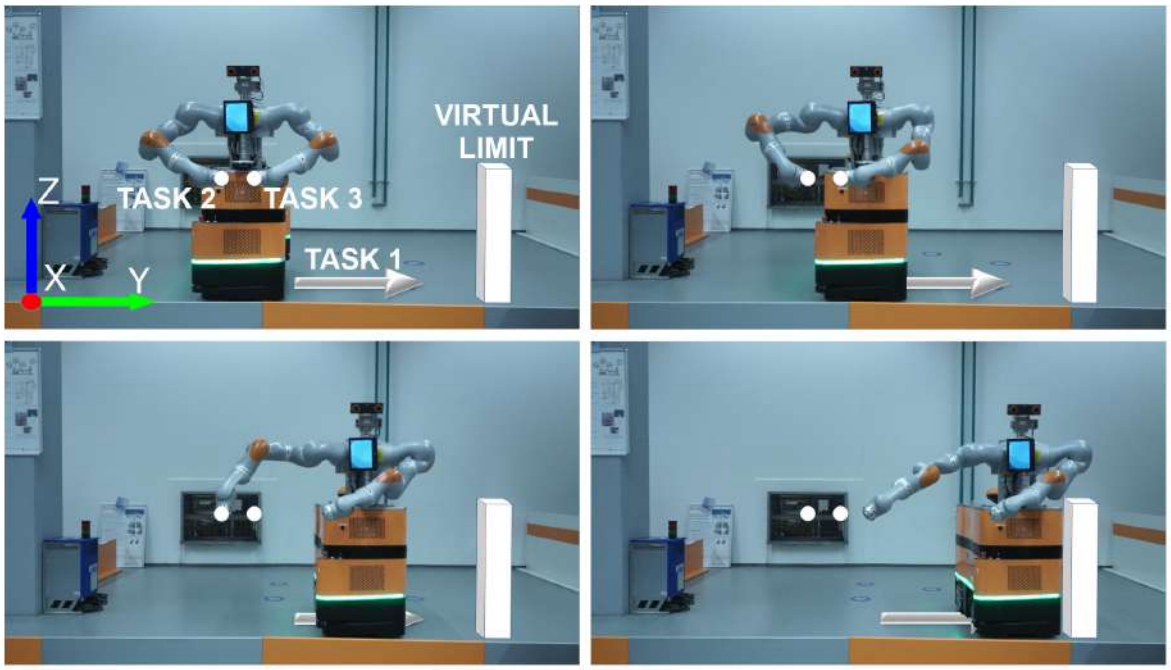


Figure 4.20: Mobile dual-arm robot moving along the y-axis while keeping end-effector positions in place in the experiment of Sect. 4.3.5-D; the tasks are dropped in decreasing numerical order when they become unfeasible.

eSNS. This is reasonable, when the difference in terms of the number of total iterations required by the two algorithms becomes significant. It also provides a good insight on how much the performance of the eSNS algorithm could be sped up, if the efficient computation of the Fast-eSNS would be combined with the warm start technique from the Opt-eSNS. Such further development is left as future work.

Finally, the obtained execution times have been compared with the ones returned by state-of-the-art QP solvers working on optimization problems in the form (4.38). For a meaningful comparison, solvers that are implemented as pure MATLAB® code (as the algorithms used in this section) have been selected. In particular, the Optimization Toolbox™and the Model Predictive Control Toolbox™offer two different active-set methods with warm start. The obtained execution times are reported in Fig. 4.19. A direct comparison with Fig. 4.17 shows that the state-of-the-art QP solvers present significantly higher computation times than all presented eSNS variants throughout the motion.

D. Experiment with the mobile dual-arm system

This last experiment highlights the computational efficiency and applicability to real-time control of the Fast-eSNS. The algorithm performs online redundancy resolution on the mobile dual-arm robot with three 3-dimensional prioritized tasks.

The first task is to move the omnidirectional mobile base sideways along the y-axis in Fig. 4.20, while keeping its x-position and its orientation about the z-axis fixed to their initial values. Additionally, limitations on the position and the velocity of each

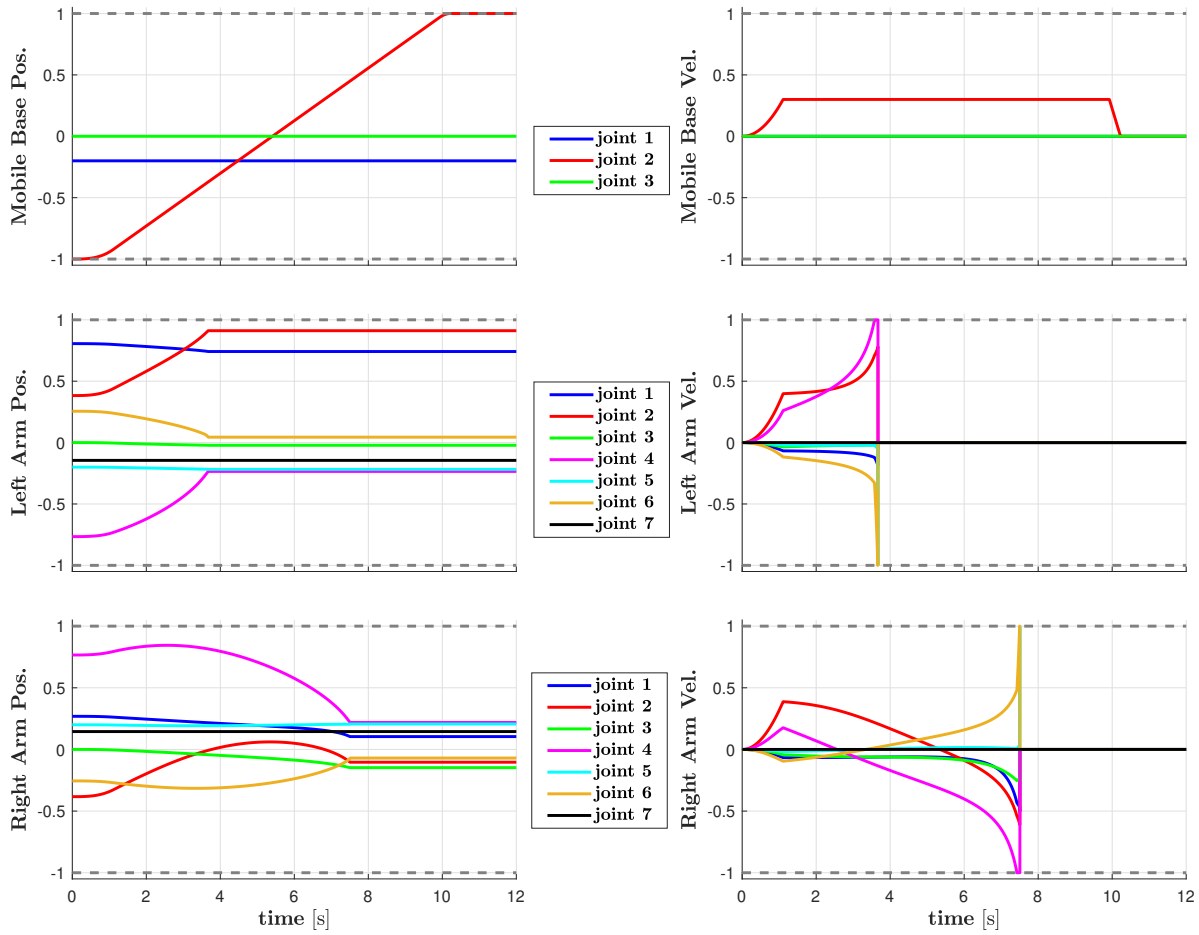


Figure 4.21: Normalized joint position and velocity produced by the velocity-based solver for the mobile dual-arm robot in the experiment of Sect. 4.3.5-D.

joint are considered. Compared to the limits used in the simulations of previous section, the maximum allowed position for the second joint of the mobile base has been increased to 1.4 m. This produces a virtual limit for the mobile base in the direction of the y-axis, represented in Fig. 4.20 by the white cuboid. The second and third tasks consist in keeping the end-effector center points of both arms in place, controlling only their position.

The velocity-based solver is used in the experiment, with $\boldsymbol{H} = \boldsymbol{I}$ and $\boldsymbol{u}_r = \boldsymbol{0}$. The initial joint configuration of the mobile platform is

$$\boldsymbol{q}_{0,b} = \begin{bmatrix} 0\,\text{m} & 0\,\text{m} & 0\,\text{rad} \end{bmatrix}^T,$$

while left and right arms start from $\boldsymbol{q}_{0,l}$ and $\boldsymbol{q}_{0,r}$, respectively, with

$$\boldsymbol{q}_{0,l} = \begin{bmatrix} 2.3562 & 0.7854 & 0 & -1.5708 & -0.5847 & 0.5236 & -0.4363 \end{bmatrix}^T \text{ rad}$$
$$\boldsymbol{q}_{0,r} = \begin{bmatrix} 0.7854 & -0.7854 & 0 & 1.5708 & 0.5847 & -0.5236 & 0.4363 \end{bmatrix}^T \text{ rad.}$$

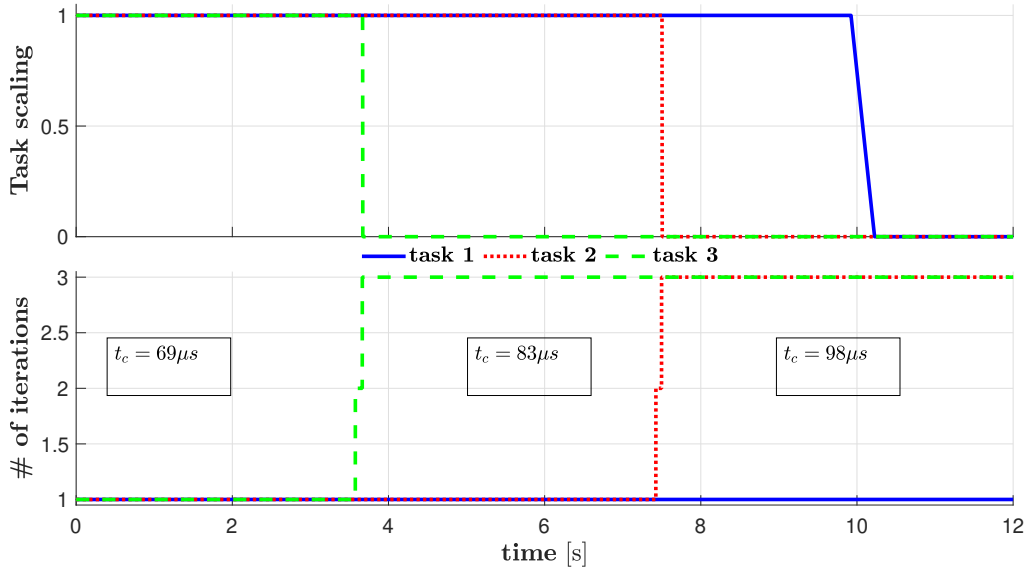


Figure 4.22: Task scaling factor and number of iterations per task of the velocity-based solver for the mobile dual-arm robot in the experiment of Sect. 4.3.5-D. Execution times are given for 3, 4, and 7 total iterations.

The computational resources are identical to the experiments in Sect. 4.3.5-A and 4.3.5-B. The control cycle time is again 1 ms. All the velocity bounds are computed according to (4.9), with $\boldsymbol{K} = 10\boldsymbol{I}$, whereas all task references are computed according to (4.5) using $\boldsymbol{K} = 100\boldsymbol{I}$.

Figure 4.21 reports the trend of the output joint position and velocity, whereas Fig. 4.22 shows the number of iterations and the evolution of the task scaling factors produced by the algorithm. As it can be seen in Fig. 4.22, the lowest priority task, i.e., task 3, is sacrificed first when it comes into conflict with the task 1, followed by task 2. Finally, the robot comes to a halt when the mobile base reaches the specified virtual limit.The computation time, t_c, required by the solver is also reported in Fig. 4.22 for a different number of total iterations.

4.3.6 Discussion

The previous sections have presented a general framework for the motion control of redundant robots. Thanks to a unified formulation of the redundancy resolution problem, the framework can arbitrary resolve redundancy at velocity, acceleration or torque level. Thus, both kinematic and torque control are possible, providing the framework with high versatility with respect to the different application domains and robot interfaces.

The unified formulation has led to the definition of a generalized control problem, which retains a certain number of essential features. First of all, both equality and inequality constraints can be defined in any task space and with arbitrary priority, allowing high flexibility in the task specification. Methods to define task velocity/acceleration references (equality constraints) and bounds (inequality constraints) have also been discussed. In particular, a novel shaping of the velocity and acceleration bounds has been

introduced, which simplifies and at the same time extends existing solutions in the literature. Moreover, the unified control problem considers the minimization of an arbitrary weighted control effort and an additional input vector used to control possibly remaining redundant DoFs.

A unified formulation also allows for the design of a single redundancy resolution algorithm. Here, a novel approach is introduced, i.e., the eSNS method, obtained by extending the existing SNS algorithm from the literature. The eSNS shares the same structure of SNS and retains all its features: strict prioritization of the constraints according to the specified hierarchy of tasks, handling of inequality constraints via saturation sets, integrated (multitask) scaling technique to extend the execution of unfeasible tasks in time while preserving task directions. However, the application to the generalized control problem and the handling of all its features discussed above have required significant extensions to the method, as thoroughly detailed in Sect. 4.3.4. Additional algorithms have also been presented, namely the Fast-eSNS and Opt-eSNS, which tackle important aspects such as computation efficiency and optimality of the solution.

The effectiveness of the eSNS in solving the generalized control problem has been proved through simulations and experiments with different kinematic structures in Sect. 4.3.5. The optimality of the solution obtained via the Opt-eSNS has been validated by comparing the results with the ones obtained by state-of-the-art QP solvers. The results of Sect. 4.3.5 have also shown a significant decrease in the computation time when the Opt-eSNS is warm-started, as well as when the Fast-eSNS is employed. A combination of these two variants should bring even faster computation times while retaining optimal control inputs. Such a solution is left as future work. Another possible computation speed-up might also be obtained by passing the obtained saturation set from one level of priority to the next one similarly to the solution by Escande, Mansard, et al. (2014). Such a solution could be investigated and compared to the Opt-eSNS with warm start.

Finally, the performance of the torque-based eSNS during physical Human-Robot Interaction (pHRI) should be evaluated. In such scenario, special attention should be put in the consideration of external forces and on the choice of the matrix $\boldsymbol{H}$ to avoid injection of active energy in the closed-loop system during interaction in the presence of saturation. An approach going in this direction has been recently proposed by Osorio and Allmendinger (2022).

4.4 Discrete-Time Implementation Issues

As illustrated in Sect. 4.2, redundancy resolution techniques have been extensively addressed in the literature and a number of strategies have been proposed over the years. Although in practice all the corresponding algorithms run on digital controllers, almost all the solid results consolidated over the years have been found in the continuous-time domain. Indeed, not so many papers address the study of redundancy resolution methods considering the discrete-time nature of the dynamic system at hand. Stability aspects of the inverse kinematics problem have been addressed in the discrete-time domain by Das, J. E. Slotine, et al. (1988) and De Maria and Marino (1985) proposing Lyapunov-based arguments. More recently, Bjerkeng, Falco, et al. (2014) and Falco and Natale (2011)

have instead presented stability analyses in the context of kinematic (sensor-based) control of redundant robots. The result is a set of necessary and sufficient conditions, leading to useful guidelines for gain selection in relation to the sampling time. Other works, like the one by Suleiman, Ayusawa, et al. (2018) have focused instead on taking the sampling time of the discrete-time system as a free variable to increase robustness of the redundancy resolution algorithm and simplify the task-space planning.

The discrete-time nature of the actual systems at hand is often overlooked in real applications. Yet, specific choices in the implementations can significantly affect the behavior of the redundancy resolution algorithm in terms of performance and/or stability. This section is dedicated to the study of some relevant questions in the field of discrete-time redundancy resolution. A first study is dedicated to the effect of the integration method used to compute joint positions on the performance and the robustness of velocity-based solvers. Then, a convergence analysis is conducted for a velocity-based redundancy resolution algorithm involving time-dependent task functions, as this aspect has not yet been investigated in the literature.

To analyze the aforementioned aspects, it is necessary to briefly recall the definition of a task function from Sect. 3.3.1. For the scope of the following analyses, it is sufficient to consider the task function as dependent only from the robot joint configuration $\boldsymbol{q} \in \mathcal{Q} \subseteq \mathbb{R}^n$ and the time $t \in \mathbb{R}_0^+$. The m-dimensional task function, $n \geq m$, is then defined as

$$\boldsymbol{e} : (\boldsymbol{q} \in \mathcal{Q}, t \in \mathbb{R}_0^+) \rightarrow \boldsymbol{e}(\boldsymbol{q}(t), t) \in \mathbb{R}^m.$$

The function $\boldsymbol{e}$ is assumed of class $\mathcal{C}^1$. Furthermore, the analysis will be limited to equality constraints. Thus, the corresponding task objectives are considered fulfilled if $\boldsymbol{e} = \boldsymbol{0}$. As already seen in the previous chapter, computing a solution requires the inversion of the constraint equation at the first-order differential level

$$\dot{\boldsymbol{e}}(\boldsymbol{q}(t), t) = \boldsymbol{J}_c(\boldsymbol{q}(t), t)\dot{\boldsymbol{q}}(t) + \boldsymbol{e}_t(\boldsymbol{q}(t), t) = \boldsymbol{0}, \tag{4.50}$$

with $\boldsymbol{J}_c = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{q}}$ being the constraint Jacobian matrix, and $\boldsymbol{e}_t = \frac{\partial \boldsymbol{e}}{\partial t}$. In particular, from the analysis in Sect. 4.3, the solution to (4.50) is computed in a closed-loop fashion as

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{J}_c^{\#}(\boldsymbol{q}(t), t)(-\boldsymbol{e}_t(\boldsymbol{q}(t), t) - \gamma \boldsymbol{e}(\boldsymbol{q}(t), t)). \tag{4.51}$$

In deriving (4.51), a constraint reference velocity defined as in (4.5) has been assumed, with a gain matrix $\boldsymbol{K} = \gamma \boldsymbol{I}$, $\gamma > 0$. Moreover, # denotes a generic right pseudo-inverse operator. The feedback term in (4.51) ensures an exponential convergence of $\boldsymbol{e}$ to zero, with a convergence rate depending on the value of γ. Since it inverts the differential constraint equation (4.50) and embeds an error term, the solution in (4.51) is often referred to in the literature as Closed-Loop Inverse Kinematics (CLIK) (Balestrino, De Maria, et al. 1984; Das, J. E. Slotine, et al. 1988; Sciavicco and Siciliano 1986).

Starting from (4.51), the evolution of the joint positions $\boldsymbol{q}$ can be obtained by integration as

$$\boldsymbol{q}(t) = \boldsymbol{q}(0) + \int_0^t \boldsymbol{J}_c^{\#}(\boldsymbol{q}(\tau), \tau)(-\boldsymbol{e}_t(\boldsymbol{q}(\tau), \tau) - \gamma \boldsymbol{e}(\boldsymbol{q}(\tau), \tau))d\tau. \tag{4.52}$$

In discrete-time implementations, (4.52) takes the form

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{J}_c^{\#}(\boldsymbol{q}(\tau), \tau)(-\boldsymbol{e}_t(\tau) - \gamma \boldsymbol{e}(\boldsymbol{q}(\tau), \tau))d\tau, \tag{4.53}$$

where $\boldsymbol{q}_k = \boldsymbol{q}(t_k)$ and $t_k = kT$, with $k \in \mathbb{Z}_0^+$ and T being the sampling time. In practice, the integral on the right-hand side of (4.53) can only be approximated, with a resulting numerical drift of the solution that is only prevented by the consideration of the feedback term.

4.4.1 Integration methods: a comparison between Euler and RK4 methods

A relevant element of CLIK methods is the integration step required to compute the evolution of the joint positions. As already mentioned, the integral in (4.53) can only be approximated, in practice. The approximation is typically performed using the simplest method, i.e., *explicit Euler* also known as *forward Euler*. However, the question arises as to whether using higher order integration methods can significantly affect the performance of the redundancy resolution algorithm. A preliminary answer to this question comes from the work by Sariyildiz and Temeltas (2011), which compares numerical integration methods of different orders in a trajectory tracking application. The results show that the choice of the numerical integration method significantly affects the performance in terms of tracking error and simulation time. Additional works (Alsultan, M. Ali, et al. 2018; Senthilkumar, M. Lee, et al. 2013) study the accuracy obtained when using Runge-Kutta methods (Kutta 1901; Runge 1895) in the dynamic analysis of simple kinematic structures for a set point regulation problem. The study in this section makes contribution to the field by carrying out a detailed comparison of the performance in different scenarios of a velocity-based CLIK algorithm when using the Runge-Kutta 4 (RK4) method with that of the 1st order explicit Euler method.

The analysis focuses on pure inverse kinematics problems, considering both regulation and trajectory following scenarios. In other words, the task function is defined as

$$\boldsymbol{e}(\boldsymbol{q}(t), t) = \boldsymbol{p}(\boldsymbol{q}(t)) - \boldsymbol{p}_d(t), \tag{4.54}$$

with $\boldsymbol{p}, \boldsymbol{p}_d \in \mathcal{P} \subseteq \mathbb{R}^7$ being the actual and desired pose of the robot end-effector with respect to a global coordinate system, respectively. As in Sect. 3.3, the pose of the end-effector is expressed as $\boldsymbol{p} = [\boldsymbol{p}_p^T \quad \boldsymbol{p}_o^T]^T$, with $\boldsymbol{p}_p = [x \quad y \quad z]^T$ representing the position, and $\boldsymbol{p}_o = [\eta \quad \boldsymbol{\epsilon}]^T$ expressing the orientation in terms of unit quaternion represented by vectors of dimension 4. Analogously, the desired pose is expressed by the position vector $\boldsymbol{p}_{p,d} = [x_d \quad y_d \quad z_d]^T$ and the orientation $\boldsymbol{p}_{o,d} = [\eta_d \quad \boldsymbol{\epsilon}_d]^T$. Thus, for a correct computation of the orientation error, the task function (4.54), representing the Cartesian error between $\boldsymbol{p}$ and $\boldsymbol{p}_d$, is rewritten as

$$\boldsymbol{e}(\boldsymbol{q}(t), t) = \begin{bmatrix} \boldsymbol{e}_p \\ \boldsymbol{e}_o \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_p(\boldsymbol{q}(t)) - \boldsymbol{p}_{p,d}(t) \\ \eta(\boldsymbol{q}(t))\boldsymbol{\epsilon}_d(t) - \eta_d(t)\boldsymbol{\epsilon}(\boldsymbol{q}(t)) - \boldsymbol{S}(\boldsymbol{\epsilon}_d(t))\boldsymbol{\epsilon}(\boldsymbol{q}(t)) \end{bmatrix} \in \mathbb{R}^6, \tag{4.55}$$

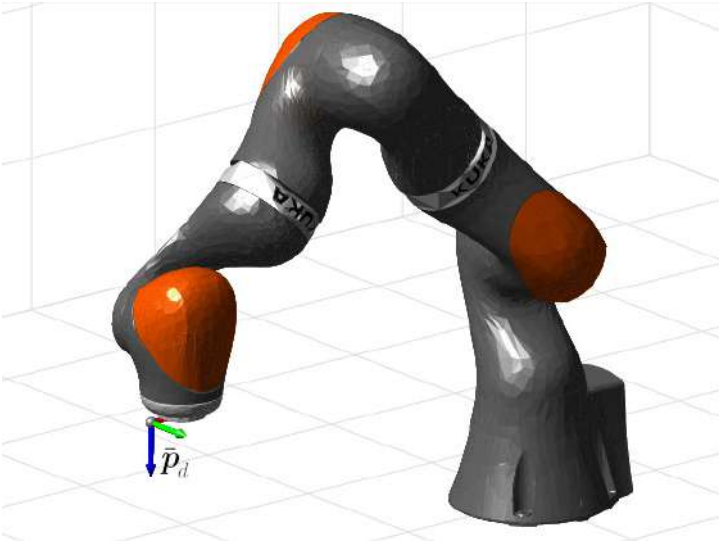


Figure 4.23: KUKA LBR iiwa: initial configuration and desired pose $\bar{\boldsymbol{p}}_d$ for Case study 1.

where the quaternion error is considered, with the help of the skew-symmetric operator $\boldsymbol{S}(\cdot)$. Moreover, by using the transformation matrices (3.7), it is possible to rewrite (4.50) as

$$\boldsymbol{J}(\boldsymbol{q}(t), t)\dot{\boldsymbol{q}}(t) - \boldsymbol{v}_d(t) = \boldsymbol{0}, \tag{4.56}$$

with $\boldsymbol{J}$ being the robot end-effector Jacobian, and $\boldsymbol{v}_d$ the vector of desired linear and angular velocity for the end-effector frame.

As a consequence of the considerations above, the joint position update (4.53) becomes

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{J}^{\#}(\boldsymbol{q}(\tau))(\boldsymbol{v}_d(\tau) - \gamma \boldsymbol{e}(\boldsymbol{q}(\tau), \tau))d\tau. \tag{4.57}$$

Starting from the (4.57), a discrete-time solution can be found approximating the time integral via the explicit-Euler technique. This operation yields

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + T(\boldsymbol{J}^{\#}(\boldsymbol{q}_k)(\boldsymbol{v}_d(t_k) - \gamma \boldsymbol{e}(\boldsymbol{q}_k, t_k)), \tag{4.58}$$

which can be rewritten as

$$\begin{aligned} \boldsymbol{q}_{k+1} &= \boldsymbol{q}_k + T\dot{\boldsymbol{q}}_k^{Eul} \\ \dot{\boldsymbol{q}}_k^{Eul} &= \boldsymbol{J}^{\#}(\boldsymbol{q}_k)(\boldsymbol{v}_d(t_k) - \gamma \boldsymbol{e}(t_k, \boldsymbol{q}_k)). \end{aligned} \tag{4.59}$$

On the other hand, approximating the time integral via RK4 yields

$$\begin{aligned} \boldsymbol{q}_{k+1} &= \boldsymbol{q}_k + T\dot{\boldsymbol{q}}_k^{RK4} \\ \dot{\boldsymbol{q}}_k^{RK4} &= \frac{1}{6}(\boldsymbol{r}_1 + 2\boldsymbol{r}_2 + 2\boldsymbol{r}_3 + \boldsymbol{r}_4) \end{aligned} \tag{4.60}$$

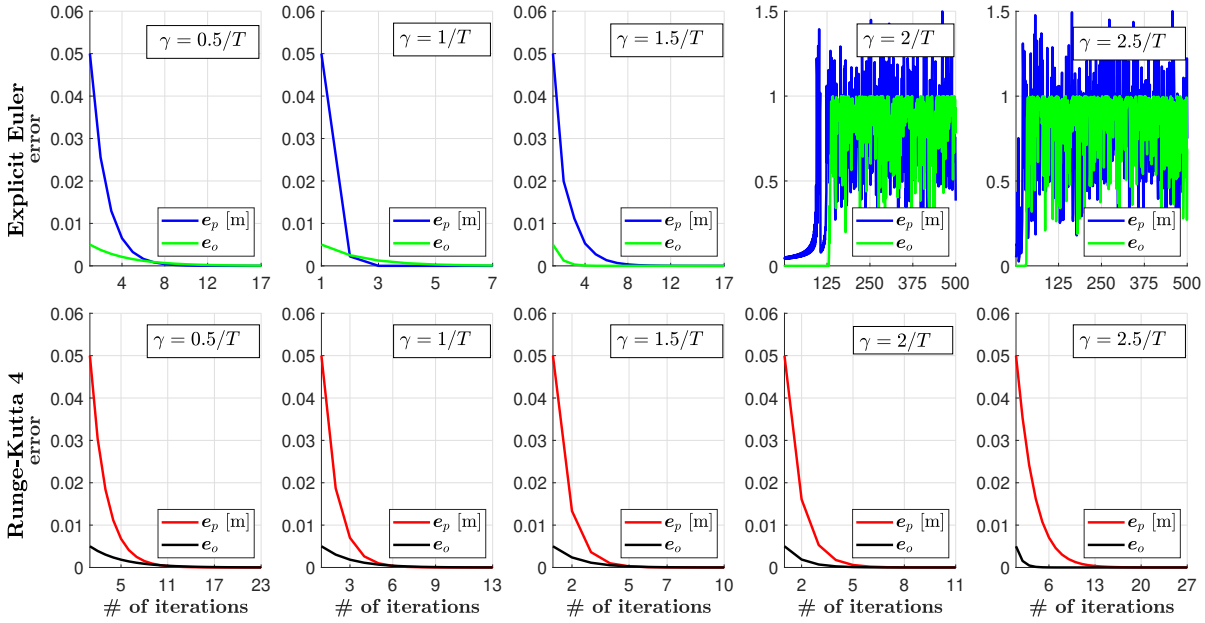


Figure 4.24: Case study 1: Cartesian error evolution for Explicit Euler (top) and Runge-Kutta 4 (bottom) for constant reference $\boldsymbol{p}_d(t) = \bar{\boldsymbol{p}}_d$ under variation of the gain γ. After 500 iterations with no convergence the simulations are stopped.

with

$$
\begin{aligned}
\boldsymbol{r}_1 &= \boldsymbol{J}^{\#}(\boldsymbol{q}_k)(\boldsymbol{v}_d(t_k) - \gamma \boldsymbol{e}(t_k, \boldsymbol{q}_k) \\
\boldsymbol{r}_2 &= \boldsymbol{J}^{\#}\left(\boldsymbol{q}_k + \frac{T}{2}\boldsymbol{r}_1\right)\left(\boldsymbol{v}_d\left(t_k + \frac{T}{2}\right) - \gamma \boldsymbol{e}\left(t_k + \frac{T}{2}, \boldsymbol{q}_k + \frac{T}{2}\boldsymbol{r}_1\right)\right) \\
\boldsymbol{r}_3 &= \boldsymbol{J}^{\#}\left(\boldsymbol{q}_k + \frac{T}{2}\boldsymbol{r}_2\right)\left(\boldsymbol{v}_d\left(t_k + \frac{T}{2}\right) - \gamma \boldsymbol{e}\left(t_k + \frac{T}{2}, \boldsymbol{q}_k + \frac{T}{2}\boldsymbol{r}_2\right)\right) \\
\boldsymbol{r}_4 &= \boldsymbol{J}^{\#}(\boldsymbol{q}_k + T\boldsymbol{r}_3)(\boldsymbol{v}_d(t_{k+1}) - \gamma \boldsymbol{e}(t_{k+1}, \boldsymbol{q}_k + T\boldsymbol{r}_3)).
\end{aligned}
$$

The proposed comparison consists of three case studies carried out in a MATLAB® simulation environment with a KUKA LBR iiwa 7-DoFs robot (Fig. 4.23). Both constant and time-varying Cartesian references are considered, as well as motions passing through kinematic singularities. The influence of the feedforward action, $\boldsymbol{v}_d$, is additionally analyzed. Finally, results are presented under variations of the sampling time and/or the feedback gain. All the simulations employ a classic Moore-Penrose pseudoinverse.

Case Study 1: constant reference

In the first case study, a constant desired Cartesian pose $\boldsymbol{p}_d(t) = \bar{\boldsymbol{p}}_d$ for the robot end effector has been set. Thus, $\boldsymbol{v}_d(t) = \boldsymbol{0}$. The term of comparison in this case is the number of iterations required to converge to a solution, i.e., to reach the desired Cartesian pose with a given precision. The initial configuration of the robot, which can be seen in Fig. 4.23, is $\boldsymbol{q}_0 = [\ \pi/4 \quad 0 \quad -\pi/2 \quad 0 \quad \pi/4 \quad 0]^T$ rad. The resulting

Initial error (norm)	**# of iterations Explicit Euler**	**# of iterations RK4**
$\|\boldsymbol{e}_{p_0}\| = 0.100$ m, $\|\boldsymbol{e}_{o_0}\| = 0.010$	19	25
$\|\boldsymbol{e}_{p_0}\| = 0.164$ m, $\|\boldsymbol{e}_{o_0}\| = 0.026$	19	26
$\|\boldsymbol{e}_{p_0}\| = 0.190$ m, $\|\boldsymbol{e}_{o_0}\| = 0.036$	20	27
$\|\boldsymbol{e}_{p_0}\| = 0.332$ m, $\|\boldsymbol{e}_{o_0}\| = 0.497$	24	29
$\|\boldsymbol{e}_{p_0}\| = 0.578$ m, $\|\boldsymbol{e}_{o_0}\| = 0.567$	25	30
$\|\boldsymbol{e}_{p_0}\| = 0.689$ m, $\|\boldsymbol{e}_{o_0}\| = 0.620$	30	31
$\|\boldsymbol{e}_{p_0}\| = 0.722$ m, $\|\boldsymbol{e}_{o_0}\| = 0.665$	31	27
$\|\boldsymbol{e}_{p_0}\| = 0.728$ m, $\|\boldsymbol{e}_{o_0}\| = 0.608$	25	33
$\|\boldsymbol{e}_{p_0}\| = 0.744$ m, $\|\boldsymbol{e}_{o_0}\| = 0.580$	28	29

Table 4.7: Case Study 1: Number of iterations required to converge for constant reference $\boldsymbol{p}_d(t) = \bar{\boldsymbol{p}}_{\boldsymbol{d}}$ under variation of the norm of the initial error $\boldsymbol{e}_0$ and constant gain $\gamma = 1/T$.

initial pose is $\boldsymbol{p}_0 = [0.5657\,\text{m} \quad 0\,\text{m} \quad 0.2290\,\text{m} \quad 0 \quad 0 \quad 1 \quad 0]^T$, while the desired pose is $\bar{\boldsymbol{p}}_d = [0.6157\,\text{m} \quad 0\,\text{m} \quad 0.2290\,\text{m} \quad 0.0049 \quad 0 \quad 0.9999 \quad 0]^T$. Therefore, the initial value of $\boldsymbol{e}$, i.e. the initial Cartesian error, is $\boldsymbol{e}_0 = [\boldsymbol{e}_{p_0}^T \quad \boldsymbol{e}_{o_0}^T]^T$, with $\boldsymbol{e}_{p_0} = [-0.05 \quad 0 \quad 0]^T$ m and $\boldsymbol{e}_{o_0} = [0 \quad 0.0049 \quad 0]^T$. Figure 4.24 shows the evolution of the Cartesian error over the number of iterations, under variation of the gain γ. The gain value is chosen as a fraction of T, so as to make the results independent of the sampling time according to (4.59)-(4.60).

The same simulation is repeated multiple times, varying the initial error $\boldsymbol{e}_0$. This is obtained by randomly sampling initial configurations $\boldsymbol{q}_0$ for the same desired pose $\bar{\boldsymbol{p}}_d$. Table 4.7 reports the results in terms of number of iterations required to converge with respect to the norm of the initial error.

Case Study 2: time-varying reference

In the second case study, the robot end effector must follow a time-varying Cartesian reference trajectory, which consists of a circular path with a trapezoidal velocity profile

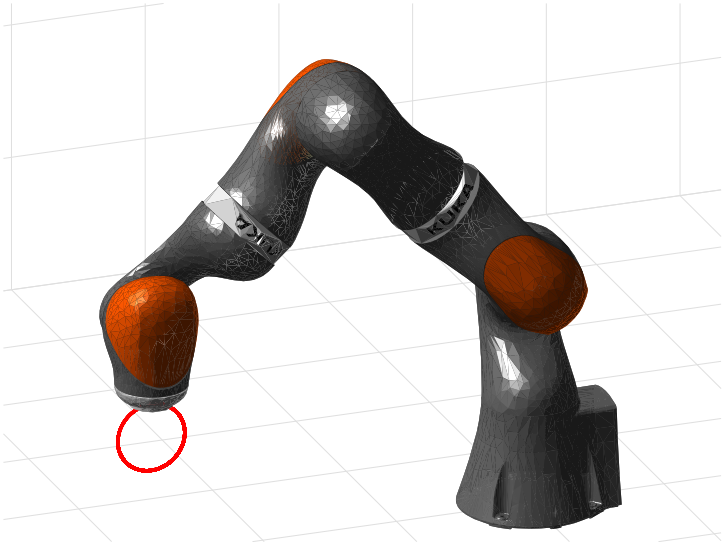


Figure 4.25: KUKA LBR iiwa: initial configuration and reference Cartesian trajectory (red line) for Case Study 2.

(Fig. 4.25):

$$\boldsymbol{p}_d(t) = \begin{bmatrix} c_x + r\cos(2\pi\sigma(t) - \pi/2) \\ c_y \\ c_z - r\sin(2\pi\sigma(t) - \pi/2) \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_d(t) = \begin{bmatrix} -2\pi r\sin(2\pi\sigma(t) - \pi/2)\dot{\sigma}(t) \\ 0 \\ -2\pi r\cos(2\pi\sigma(t) - \pi/2)\dot{\sigma}(t) \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where $\sigma(t)$ is the path parameter used to plan the trajectory, and $\boldsymbol{c} = [c_x \quad c_y \quad c_z]^T$ and r represent the center and radius of the circle, respectively. The initial joint configuration is

$$\boldsymbol{q}_0 = [0 \quad \pi/4 \quad 0 \quad -\pi/2 \quad 0 \quad \pi/4 \quad 0]^T \text{rad},$$

which produces an initial error

$$\boldsymbol{e}_0 = [0.9541 \cdot 10^{-4}\,\text{m} \quad 0\,\text{m} \quad -0.9445 \cdot 10^{-4}\,\text{m} \quad 0 \quad -0.0050 \quad 0]^T.$$

Figures 4.26 and 4.27 show the performance of the two methods in terms of tracking error, also including the case in which no velocity feedforward is used in the computation, i.e., $\boldsymbol{v}_d = \boldsymbol{0}$. Results are presented under variation of the sampling time T and for a fixed gain value $\gamma = 1/T$.

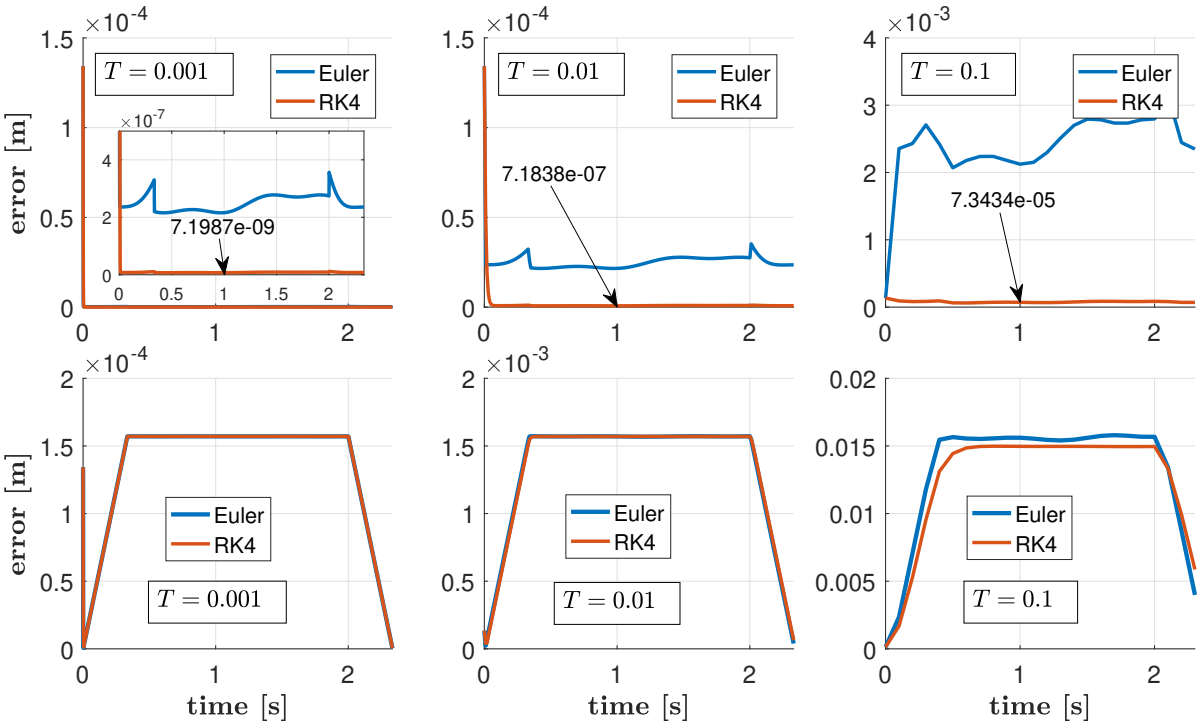


Figure 4.26: Case Study 2: Cartesian position error evolution for a time-varying reference $\boldsymbol{p}_d(t)$ and gain $\gamma = 1/T$, under variation of the sampling time T, with (top) and without (bottom) velocity feedforward. Smaller overlapping plots provide zoomed-in views.

Case Study 3: moving through singularities

In the last case study, a time-varying Cartesian reference is generated, so as to force the robot to move through a kinematic singularity during the execution of the motion. In this case, besides the tracking performance, smoothness of the resulting joint velocity is also investigated. The Cartesian trajectory consists of a circular path with a trapezoidal velocity profile (Fig. 4.28):

$$\boldsymbol{p}_d(t) = \begin{bmatrix} c_x + r\cos(2\pi\sigma(t) - \pi/2) \\ c_y \\ c_z + r\sin(2\pi\sigma(t) - \pi/2) \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_d(t) = \begin{bmatrix} -2\pi r\sin(2\pi\sigma(t) - \pi/2)\dot{s}(t) \\ 0 \\ 2\pi r\cos(2\pi\sigma(t) - \pi/2)\dot{s}(t) \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where $\sigma(t)$ is the path parameter used to plan the trajectory, and $\boldsymbol{c} = [c_x \quad c_y \quad c_z]^T$ and r represent the center and radius of the circle, respectively. The initial joint configuration

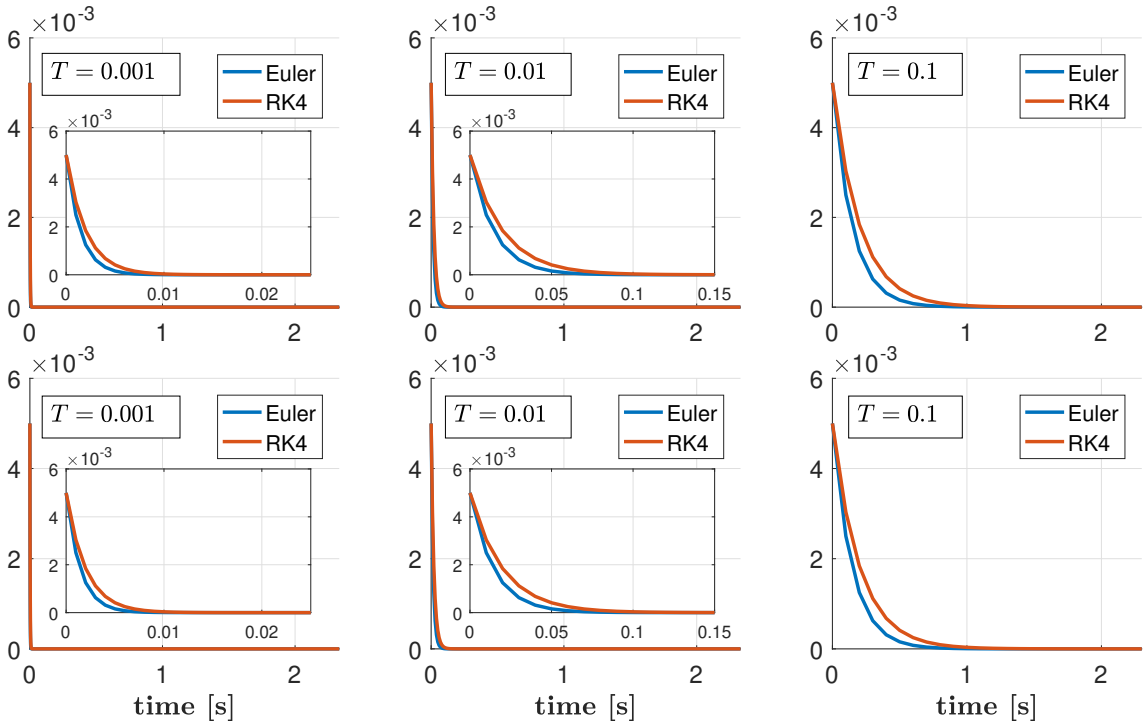


Figure 4.27: Case Study 2: Cartesian orientation error evolution for time-varying reference $\boldsymbol{p}_d(t)$ and gain $\gamma = 1/T$, under variation of the sampling time T, with (top) and without (bottom) velocity feedforward. Smaller overlapping plots provide zoomed-in views.

is

$$\boldsymbol{q}_0 = [0 \quad 1.2879 \quad 0 \quad -0.6988 \quad 0 \quad -0.4160 \quad 0]^T \text{rad},$$

which gives a corresponding initial error

$$\boldsymbol{e}_0 = [0.0103 \cdot 10^{-4}\,\text{m} \quad 0\,\text{m} \quad -0.5921 \cdot 10^{-4}\,\text{m} \quad 0 \quad -0.4816 \cdot 10^{-4} \quad 0]^T.$$

Figure 4.29 shows the resulting joint velocity evolution over the time produced by the two methods. Figure 4.30 presents, instead, the performance in terms of tracking error. All the results have been generated considering a fixed sampling time $T = 0.01\,\text{s}$, and are presented under variation of the gain γ. As an additional comparison, the simulation is repeated ignoring the feedforward term, i.e., $\boldsymbol{v}_d = \boldsymbol{0}$. The resulting joint velocity and tracking error for this case are reported in Figures 4.31 and 4.32, respectively.

Discussion

From the comparison between the (4.59) and the (4.60), it can be easily recognized that using Runge-Kutta 4 significantly increases the complexity of the algorithm. In particular, this method requires, for the kth iteration, the computation of four times the forward kinematics and the Jacobian matrix of the robot. Consequently, four pseudo-inverses must be computed, which is the most consuming operation in (4.59) and (4.60).

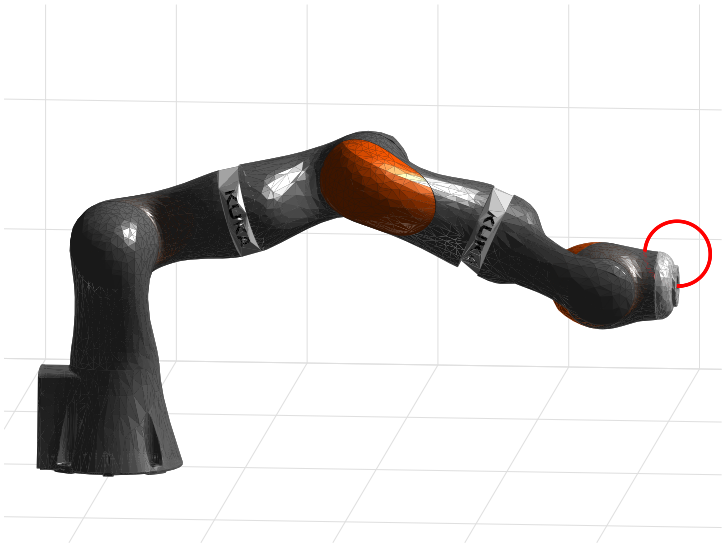


Figure 4.28: KUKA LBR iiwa: initial configuration and reference Cartesian trajectory (red line) for Case Study 3.

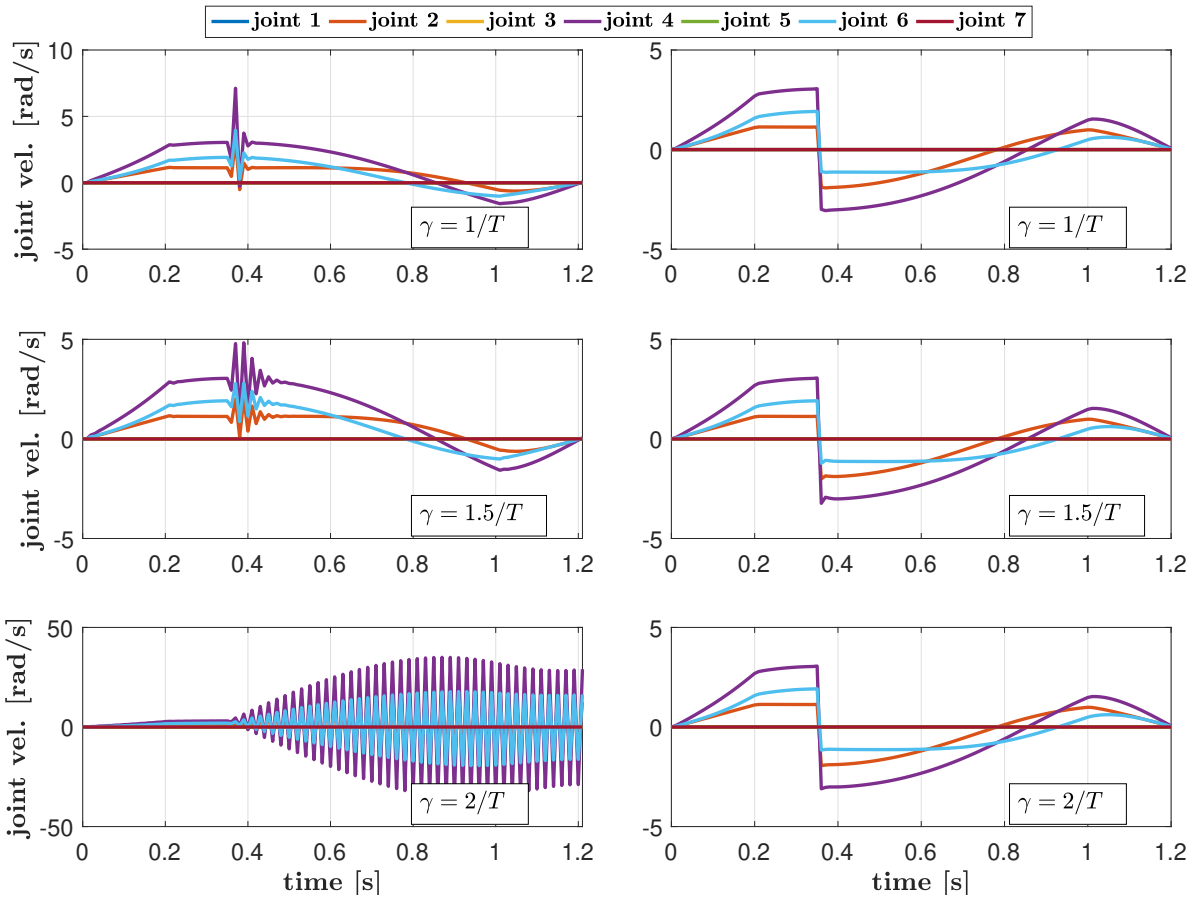


Figure 4.29: Case Study 3: Joint velocity using explicit Euler (left) and Runge-Kutta 4 (right) for time-varying reference $\boldsymbol{p}_d(t)$ with feedforward when passing through a singularity, for a fixed sampling time $T = 0.01\,\mathrm{s}$, under variation of the gain γ.

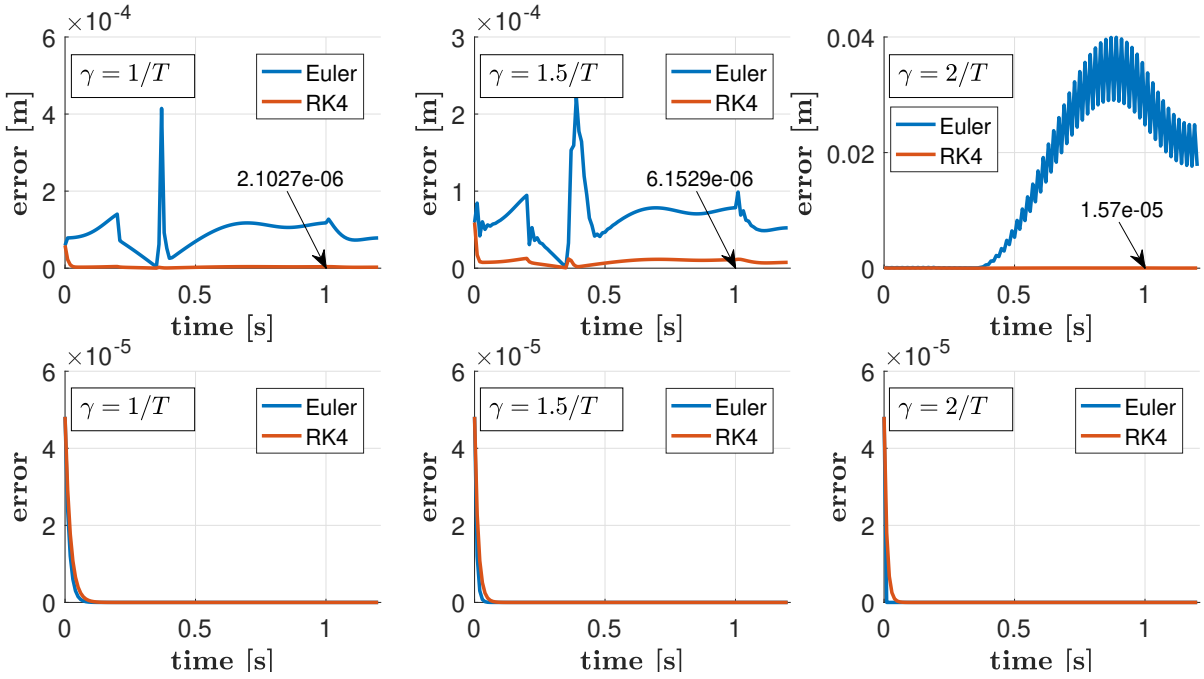


Figure 4.30: Case Study 3: Cartesian position (top) and orientation (bottom) error for time-varying reference $\boldsymbol{p}_d(t)$ with feedforward when passing through a singularity, for a fixed sampling time $T = 0.01\,\text{s}$, under variation of the gain γ.

It can then be concluded, that the CLIK with Runge-Kutta 4 is approximately four times computationally more expensive than the CLIK with Explicit Euler. Furthermore, the intermediate values $\boldsymbol{v}_d(t_k + T/2)$ and $\boldsymbol{p}_d(t_k + T/2)$ might not be available. Thus, additional computation might be required to compute a suitable estimation of these terms. In the presented simulations, the desired trajectory is generated with half of the sampling time to obtain them.

In case of a constant Cartesian reference, the first case study has shown that there is no real benefit in using Runge-Kutta 4 in the integration step. Explicit Euler seems indeed to induce faster convergence to a solution with smaller gain values. On the other hand, the CLIK with Runge-Kutta 4 keeps converging to a solution for a larger range of gain values (see Fig. 4.24). Only in the case of very large initial error, the difference in the results seems to be less clear: the CLIK with Runge-Kutta 4 converges in some cases in less or just slightly more iterations (see Tab. 4.7).

The benefit in using a higher order integration method, such as Runge-Kutta 4, becomes more evident when following a time-varying trajectory (see the second Case Study 2). In such a case, significantly smaller tracking error is observed, especially for larger sampling time and when considering the feedforward term. Indeed, the CLIK with Runge-Kutta 4 produces an error that is, on average, two orders of magnitude smaller when compared to the one produced by the CLIK with Explicit Euler. The difference in the results is instead dramatically reduced when ignoring the feedforward term (see Fig. 4.26).

The last case study shows that using Runge-Kutta 4 produces significantly smoother

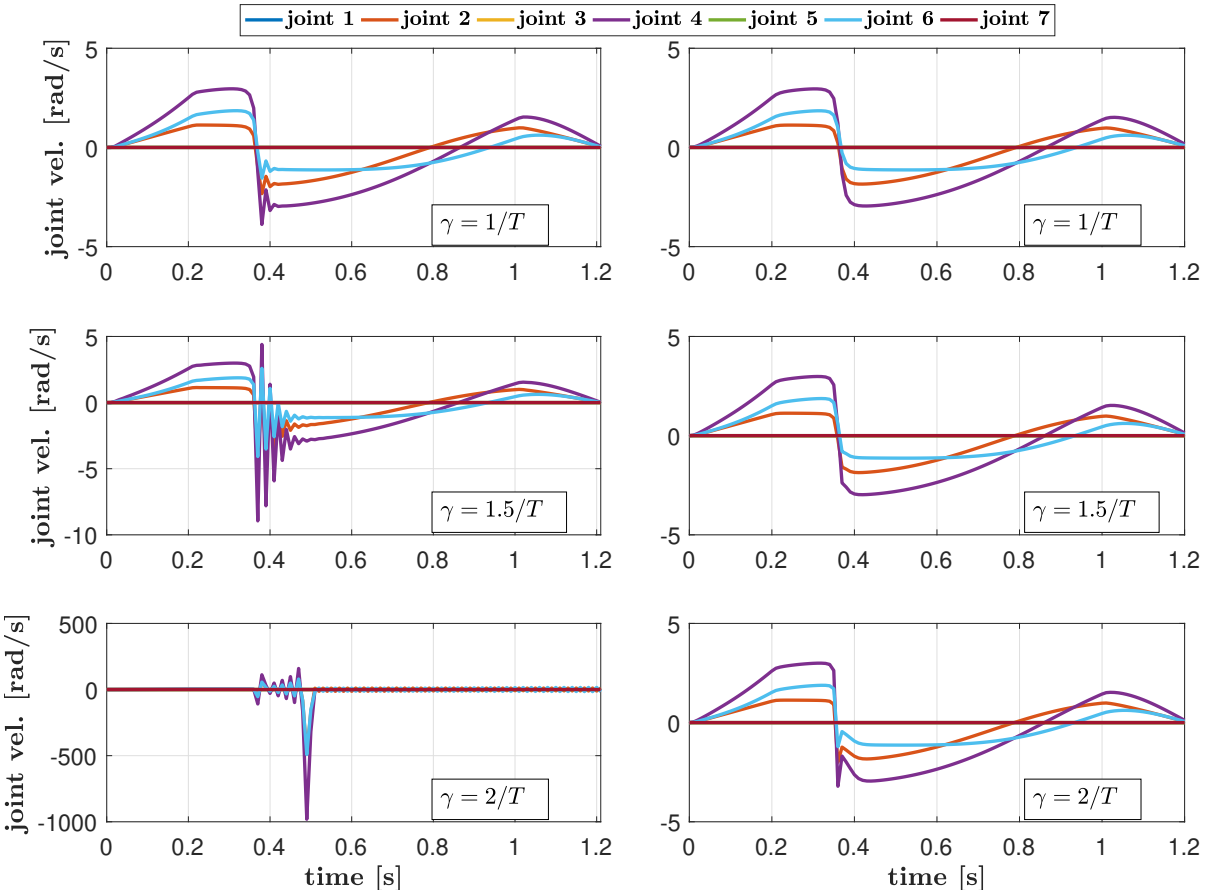


Figure 4.31: Case Study 3: Joint velocity using explicit Euler (left) and Runge-Kutta 4 (right) for time-varying reference $\boldsymbol{p}_d(t)$ without feedforward when passing through a singularity, for a fixed sampling time $T = 0.01\,\text{s}$, under variation of the gain γ.

joint velocities when moving through a singularity. This effect becomes more evident as the gain value increases (Fig. 4.29 and 4.31). This result can be explained considering that the four terms that appear in the solution in (4.60) are computed evaluating four slightly different joint configurations, which will be sampled in a neighborhood of the singular configuration. Therefore, the unpleasant effect of kinematic singularities on the solution is "filtered-out" in the sum in (4.60). The tracking performance confirms the results from second case study: with the feedforward term included, using Runge-Kutta 4 produces significantly smaller errors (see Fig. 4.30), whereas no clear difference can be observed when assuming $\boldsymbol{v}_d = \boldsymbol{0}$ (see Fig. 4.32). Once again, the CLIK with Runge-Kutta 4 also proves to keep stability for a larger range of gain values, whereas the CLIK with Explicit Euler starts diverging at $\gamma = 2/T$, as it can be seen also in Fig. 4.30 and 4.32.

Summarizing, the considered case studies show that significant benefits are observed when using RK4 to track time-varying Cartesian references and move through kinematic singularities. Moreover, the performed simulations suggest that stability is retained for larger gain values compared to traditional CLIK algorithms with explicit Euler. On the other hand, the remarkably increased complexity of the mathematical model does not

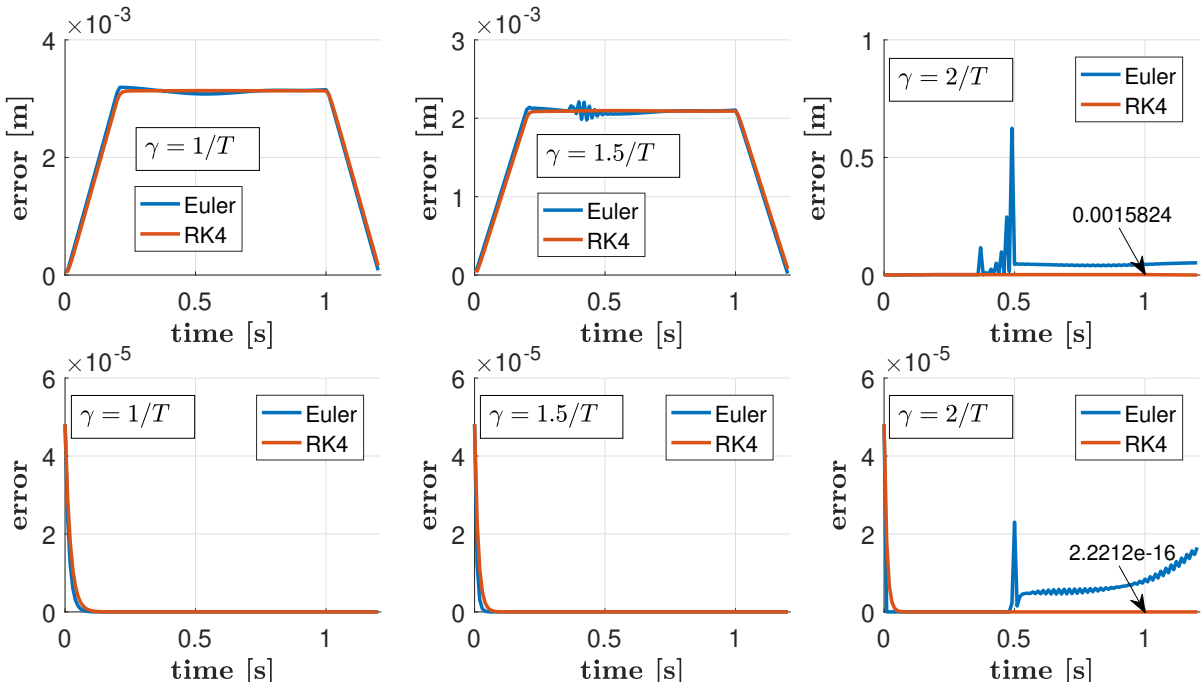


Figure 4.32: Case Study 3: Cartesian position (top) and orientation (bottom) error for time-varying reference $\boldsymbol{p}_d(t)$ without feedforward when passing through a singularity, for a fixed sampling time $T = 0.01\,\text{s}$, under variation of the gain γ.

always guarantee better performance. More specifically, the CLIK implementation using explicit Euler presents comparable or even better results for constant Cartesian reference or when ignoring the feedforward term.

4.4.2 Convergence analysis in the discrete-time domain

As extensively pointed out in this thesis, a nice property of including a feedback term in the definition the constraint reference velocity (see Eq. (4.5)) is that an evolution of the task function as a (converging) first-order linear system is imposed. This can be better visualized by substituting (4.51) into the (4.50). Indeed, this operation yields

$$\dot{\boldsymbol{e}} + \gamma \boldsymbol{e} = \boldsymbol{0}. \tag{4.61}$$

In the continuous time domain, the only condition of convergence is $\gamma > 0$. However, limits on the value of γ exist when implementing the CLIK method in the discrete-time domain. Overcoming such limits might compromise the stability of the discrete-time system. The work by Falco and Natale (2011) has found sufficient conditions to the stability of the CLIK algorithm in case of task functions depending only on the joint coordinates $\boldsymbol{q}$. In such condition, limitations for the value of γ are found, which depend on the initial task error and the sampling time. The study in this section extends these results by considering task functions with an explicit time dependency.

The analysis assumes that explicit Euler is used as integration method, with a sam-

pling time T. Therefore, the CLIK equation (4.53) can be expressed as

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + T\boldsymbol{J}_{c_k}^{\#}(-\boldsymbol{e}_{t_k} - \gamma \boldsymbol{e}_k), \tag{4.62}$$

where $\boldsymbol{J}_{c_k} = \boldsymbol{J}_c(\boldsymbol{q}_k, kT)$, $\boldsymbol{e}_{t_k} = \boldsymbol{e}_t(\boldsymbol{q}_k, kT)$, and $\boldsymbol{e}_k = \boldsymbol{e}(\boldsymbol{q}_k, kT)$. Additionally, the study is based on the following assumptions:

1. $\exists\, \sigma^* \in \mathbb{R}^+ : \underline{\sigma}(\boldsymbol{J}_c(\boldsymbol{q},t)\boldsymbol{J}_c^T(\boldsymbol{q},t)) \geq \sigma^* \quad \forall\, \boldsymbol{q} \in \mathcal{Q},\ t \in \mathbb{R}_0^+$
2. $\exists\, \delta \in \mathbb{R}^+ : \|\boldsymbol{J}_c(\boldsymbol{q},t)\| < \delta \quad \forall\, \boldsymbol{q} \in \mathcal{Q},\ t \in \mathbb{R}_0^+$
3. $\exists\, \omega \in \mathbb{R}^+ : \|\boldsymbol{e}_t(\boldsymbol{q},t)\| < \omega \quad \forall\, \boldsymbol{q} \in \mathcal{Q},\ t \in \mathbb{R}_0^+$
4. $\exists\, \mu \in \mathbb{R}^+ : \|\frac{\partial^2 e_i(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}\| \leq \mu\ \forall \boldsymbol{q} \in \mathcal{Q},\ t \in \mathbb{R}_0^+, i \in [1, m],$

where the spectral norm, i.e., the largest singular value of a matrix, is assumed as matrix norm, and the symbol $\underline{\sigma}(\boldsymbol{X})$ denotes the smallest singular value of the matrix $\boldsymbol{X}$. Moreover, e_i denotes the ith component of $\boldsymbol{e}$, whereas $\boldsymbol{\theta}$ groups the joint coordinates and the time variable in a single vector, namely $\boldsymbol{\theta} = [\boldsymbol{q}^T \quad t]^T$. While assumption 1 aims at quantifying the remoteness from singularity of $\boldsymbol{J}_c$, the remaining assumptions impose smoothness constraints on the task definition, as they assume a bounded norm on $\mathcal{Q}$ of both the Jacobian and the Hessian of $\boldsymbol{e}$. Finally, it is worth noticing that, in view of assumptions 1 and 2 and the standard properties of the matrix norm, it is

$$\|\boldsymbol{J}_c^{\#}(\boldsymbol{q},t)\| \leq \delta/\sigma^* \triangleq \delta' \quad \forall\, \boldsymbol{q} \in \mathcal{Q},\ t \in \mathbb{R}^+. \tag{4.63}$$

The derivation of the task function dynamics uses Taylor's theorem with explicit second-order Lagrange remainders. The Taylor expansion of $\boldsymbol{e}(\boldsymbol{q}+\tilde{\boldsymbol{q}}, t+\tilde{t})$ around $(\boldsymbol{q}, t)$ for some $(\tilde{\boldsymbol{q}}, t) \in \mathbb{R}^{n+1}$ is given by

$$\boldsymbol{e}(\boldsymbol{q}+\tilde{\boldsymbol{q}}, t+\tilde{t}) = \boldsymbol{e}(\boldsymbol{q},t) + \boldsymbol{J}(\boldsymbol{q},t)\tilde{\boldsymbol{q}} + \boldsymbol{e}_t(\boldsymbol{q},t)\tilde{t} + \boldsymbol{r}(\boldsymbol{q},t,\boldsymbol{\zeta}), \tag{4.64}$$

with the Lagrange remainder $\boldsymbol{r}$ being

$$\boldsymbol{r}(\boldsymbol{q},t,\boldsymbol{\zeta}) = \frac{1}{2}\begin{bmatrix} \tilde{\boldsymbol{q}}^T \dfrac{\partial^2 e_1(\boldsymbol{q},t)}{\partial \boldsymbol{q}^2}\Big|_{\boldsymbol{q}+\zeta_1\tilde{\boldsymbol{q}}} \tilde{\boldsymbol{q}} & \tilde{t}^T \dfrac{\partial^2 e_1(\boldsymbol{q},t)}{\partial t^2}\Big|_{t+\zeta_m\tilde{t}} \tilde{t} \\ \vdots & \\ \tilde{\boldsymbol{q}}^T \dfrac{\partial^2 e_m(\boldsymbol{q},t)}{\partial \boldsymbol{q}^2}\Big|_{\boldsymbol{q}+\zeta_m\tilde{\boldsymbol{q}}} \tilde{\boldsymbol{q}} & \tilde{t}^T \dfrac{\partial^2 e_m(\boldsymbol{q},t)}{\partial t^2}\tilde{t}\Big|_{t+\zeta_m\tilde{t}} \tilde{t} \end{bmatrix}$$

for some $\boldsymbol{\zeta} \in \mathbb{R}^{n+1}$ whose elements all belong to the range $[0, 1]$. As shown in Lemma 1 of the work by Falco and Natale (2011) assumption 4 implies that $\boldsymbol{r}$ is bounded. In particular, it is

$$\exists\, \nu > 0 : \|\boldsymbol{r}\| \leq \nu \left(\|\tilde{\boldsymbol{q}}\|^2 + \tilde{t}^2\right). \tag{4.65}$$

The goal of the following analysis it to find sufficient conditions to the convergence of the system (4.62). To this end, (4.64) and (4.62) can be used to derive an expression of the dynamics of the task function in the discrete-time domain:

$$\begin{aligned}\boldsymbol{e}_{k+1} &= \boldsymbol{e}_k + T\boldsymbol{J}_{c_k}\boldsymbol{J}_{c_k}^{\#}(-\boldsymbol{e}_{t_k} - \gamma\boldsymbol{e}_k) + T\boldsymbol{e}_{t_k} + \boldsymbol{r}_k \\ &= \boldsymbol{e}_k - T\boldsymbol{J}_{c_k}\boldsymbol{J}_{c_k}^{\#}\boldsymbol{e}_{t_k} - \gamma T\boldsymbol{J}_{c_k}\boldsymbol{J}_{c_k}^{\#}\boldsymbol{e}_k + T\boldsymbol{e}_{t_k} + \boldsymbol{r}_k \\ &= (1-\gamma T)\boldsymbol{e}_k + \boldsymbol{r}_k\end{aligned} \tag{4.66}$$

with $\boldsymbol{r}_k = \boldsymbol{r}(\boldsymbol{q}_k, kT, \boldsymbol{\zeta})$. The analysis is based on the following two lemmas. The first lemma is a consequence of the comparison principle for discrete-time systems and is a special case of classical results on recurrence inequalities (Lakshmikantham and Trigiante 2002), whereas the second lemma is specifically built to simplify the proof of the following theorem.

Lemma 1. *Let b_h be a nonnegative sequence that satisfies $b_h + 1 \leq \alpha b_h + c$, where α and c are nonnegative real numbers. If $b_0 \leq a_0$, a_0 being the initial condition of the dynamic system $a_{h+1} = \alpha a_h + c$, then*

$$b_h \leq a_h \quad \forall h \geq 0.$$

Proof. The proof is by induction. The claim is true for $h = 0$. Suppose it is true for h; then, for h + 1, it is

$$b_{h+1} \leq \alpha b_h + c \leq \alpha a_h + c = a_{h+1}.$$

□

Lemma 2. *Let γ, T, ν, δ', and ω be positive real numbers, with $\gamma T < 1$. Moreover, let α be the unknown variable of the second-order equation*

$$\alpha^2 + b\alpha + c = 0, \tag{4.67}$$

where $b = \gamma T - 2 - 2\gamma T^2\nu\delta'^2\omega$, $c = 1 - \gamma T + 2\gamma T^2\nu\delta'^2\omega + \gamma^2 T^2\nu\delta'^2\beta$, and $\beta = T^2\nu(\delta'^2\omega^2 + 1)$. Equation (4.67) admits two positive solutions smaller than 1 iff

$$\gamma < \min\left(\frac{1}{T}, \frac{1 - 2T\nu\delta'^2\omega}{T^3\nu^2\delta'^2(\delta'^2\omega^2+1)}\right) \quad \textit{and} \quad \omega \leq \frac{1}{2T\nu\delta'^2}.$$

Proof. Since $\gamma T < 1$ and $\omega > 0$, it is $c > 0$ and $b < 0$, hence (4.67) always admits two positive solutions. These solutions are smaller than one iff (Jury stability criterion)

$$0 < c < 1 \tag{4.68a}$$

$$(c + 1 - b)(c + 1 + b) > 0. \tag{4.68b}$$

Since it is already $c > 0$, the condition (4.68a) is satisfied by imposing $c < 1$. This yields

$$\begin{aligned} 1 - \gamma T + 2\gamma T^2\nu\delta'^2\omega + \gamma^2 T^2\nu\delta'^2\beta &< 1 \\ \gamma T(-1 + 2T\nu\delta'^2\omega + \gamma T\nu\delta'^2\beta) &< 0 \\ -1 + 2T\nu\delta'^2\omega + \gamma T^3\nu^2\delta'^2(\delta'^2\omega^2+1) &< 0 \\ \Rightarrow \gamma < \frac{1 - 2T\nu\delta'^2\omega}{T^3\nu^2\delta'^2(\delta'^2\omega^2+1)} \quad \text{and} \quad \omega \leq \frac{1}{2T\nu\delta'^2} &\triangleq \omega_2. \end{aligned}$$

Moreover, the condition (4.68b) is always verified since $b < 0$, and thus $c + 1 - b > 0$ while imposing $c + 1 + b > 0$ yields

$$\begin{aligned} 1 - \gamma T + 2\gamma T^2 \nu \delta'^2 \omega + \gamma^2 T^2 \nu \delta'^2 \beta & \\ + 1 + \gamma T - 2 - 2\gamma T^2 \nu \delta'^2 \omega > 0, & \\ \gamma^2 T^2 \nu \delta'^2 \beta > 0 & \end{aligned}$$

which is always true under the considered hypothesis. □

Theorem 1. *Under the assumptions 1–4, if the gain γ, the initial task space error $\boldsymbol{e}_0$, and the upper bound ω are such that*

$$\begin{aligned} 0 < \gamma < \min\left(\frac{1}{T}, \frac{1 - 2T\nu\delta'^2\omega}{T^3\nu^2\delta'^2(\delta'^2\omega^2 + 1)}\right) \quad & \textit{and} \\ \|\boldsymbol{e}_0\| < \frac{\beta}{1 - \alpha} \quad & \textit{and} \\ \omega \leq \frac{1}{2T\nu\delta'^2}, & \end{aligned} \tag{4.69}$$

where $\beta = T^2\nu(\delta'^2\omega^2 + 1)$, and α is a solution of the second-order equation (4.67), or

$$\begin{aligned} 0 < \gamma < \min\left(\frac{1}{T}, \frac{1 - 2T\nu\delta'^2\omega}{T\nu\delta'^2\|\boldsymbol{e}_0\|}\right) \quad & \textit{and} \\ \tilde{e}_{0,l} < \|\boldsymbol{e}_0\| < \tilde{e}_{0,u} \quad & \textit{and} \\ \omega < \frac{1}{4T\nu\delta'^2} - \nu T \quad & \textit{and} \\ T < \frac{1}{2\nu\delta'}, & \end{aligned} \tag{4.70}$$

with $\tilde{e}_{0,l}$ and $\tilde{e}_{0,u}$ solutions of the second-order equation

$$\gamma^2 T^2 \nu \delta'^2 \tilde{e}_0^2 + (\gamma T - 2\gamma T^2 \nu \delta'^2 \omega) - \beta = 0, \tag{4.71}$$

then the CLIK algorithm in (4.62) ensures the exponential convergence of the task space error dynamics, i.e.,

$$\exists\, \alpha \in (0,1), \tilde{e}_s \in [0, \infty), \tilde{e}_t \in \mathbb{R} : \|\boldsymbol{e}_k\| \leq \tilde{e}_t \alpha^k + \tilde{e}_s \quad \forall k \geq 0. \tag{4.72}$$

Proof. Starting from (4.66), it is possible to obtain the following inequalities:

$$\begin{aligned} \|\boldsymbol{e}_{k+1}\| &\leq |1 - \gamma T| \|\boldsymbol{e}_k\| + \|\boldsymbol{r}\| \\ &\leq |1 - \gamma T| \|\boldsymbol{e}_k\| + \nu \|T \boldsymbol{J}_k^{\#} (\boldsymbol{j}_k + \gamma \boldsymbol{e}_k)\|^2 + \nu T^2 \\ &\leq |1 - \gamma T| \|\boldsymbol{e}_k\| + T^2 \nu \delta'^2 \|\boldsymbol{j}_k + \gamma \boldsymbol{e}_k\|^2 + \nu T^2 \\ &\leq |1 - \gamma T| \|\boldsymbol{e}_k\| + T^2 \nu \delta'^2 (\omega^2 + \gamma^2 \|\boldsymbol{e}_k\|^2 + 2\gamma\omega \|\boldsymbol{e}_k\|) + \nu T^2 \\ &\leq (|1 - \gamma T| + 2\gamma T^2 \nu \delta'^2 \omega + \gamma^2 T^2 \nu \delta'^2 \|\boldsymbol{e}_k\|) \|\boldsymbol{e}_k\| + T^2 \nu \delta'^2 \omega^2 + \nu T^2 \end{aligned} \tag{4.73}$$

where the bounds in (4.63), (4.65), and assumption 3 have been exploited, together with standard norm properties. Now, assume that the task error norm is bounded, i.e.,

$$\|\boldsymbol{e}_k\| \leq \phi \quad \forall k \geq 0, \tag{4.74}$$

and consider $\gamma T < 1$. Then, it is possible to rewrite (4.73) as

$$\|\boldsymbol{e}_{k+1}\| \leq \alpha \|\boldsymbol{e}_k\| + \beta \quad \forall k \geq 0,$$

with

$$\begin{aligned} \alpha &= 1 - \gamma T + 2\gamma T^2 \nu \delta'^2 \omega + \gamma^2 T^2 \nu \delta'^2 \phi, \\ \beta &= T^2 \nu (\delta'^2 \omega^2 + 1). \end{aligned}$$

It will soon be shown that the condition (4.74) is guaranteed either by the hypothesis (4.69) or (4.70). To this purpose, consider the scalar linear system

$$\tilde{e}_{k+1} = \alpha \tilde{e}_k + \beta,$$

which presents asymptotic convergence for any $\alpha < 1$. The system response is given by

$$\tilde{e}_k = \tilde{e}_0 \alpha^k + \beta \frac{1 - \alpha^k}{1 - \alpha} = \left(\tilde{e}_0 - \frac{\beta}{1 - \alpha} \right) \alpha^k + \frac{\beta}{1 - \alpha} \triangleq \tilde{e}_t \alpha^k + \tilde{e}_s.$$

In the case $\tilde{e}_0 < \beta/(1 - \alpha)$, let choose $\phi = \beta/(1 - \alpha)$. Therefore, α is implicitly defined as

$$\alpha = 1 - \gamma T + 2\gamma T^2 \nu \delta'^2 \omega + \gamma^2 T^2 \nu \delta'^2 \frac{\beta}{1 - \alpha},$$

that is the equation (4.67) in Lemma 2, which, under the conditions (4.69), admits solutions $\alpha < 1$ in view of the result of Lemma 2. Then, from Lemma 1 it results that

$$\|\boldsymbol{e}_k\| \leq \tilde{e}_t \alpha^h + \tilde{e}_s \quad \forall k > 0,$$

which proves (4.72) and ensures that (4.74) is verified.

In the case $\tilde{e}_0 \geq \beta/(1 - \alpha)$, let instead choose $\phi = \tilde{e}_0$. To have $\alpha < 1$, the following inequality must hold

$$1 - \alpha = \gamma T - 2\gamma T^2 \nu \delta'^2 \omega - \gamma^2 T^2 \nu \delta'^2 \tilde{e}_0 > 0,$$

which is ensured by the condition on γ from (4.70). Then Lemma 1 ensures that the result (4.72) is obtained. The proof is completed by noting that the condition on $\tilde{e}_0 \geq \beta/(1-\alpha)$ generates the following inequality

$$\tilde{e}_0 \geq \frac{\beta}{\gamma T - 2\gamma T^2 \nu \delta'^2 \omega - \gamma^2 T^2 \nu \delta'^2 \tilde{e}_0}$$

that is

$$(\nu T \delta'^2 \gamma^2) \tilde{e}_0^2 + (2\nu T \gamma \delta'^2 \omega - \gamma) \tilde{e}_0 + \nu T (1 + \delta'^2 \omega^2) \leq 0, \tag{4.75}$$

which, under the conditions on ω and T from (4.70) is equivalent to $\tilde{e}_{0,l} \leq \tilde{e}_0 \leq \tilde{e}_{0,u}$, with $\tilde{e}_{0,l}$ and $\tilde{e}_{u,u}$ defined in (4.71). □

It is finally worth noticing that, by setting $\omega = 0$ and $\|\boldsymbol{r}\| \leq \nu \|\boldsymbol{q}\|^2$, it is possible to verify that in the absence of time-dependent terms in the task function, the obtained results coherently fall in the ones obtained by Falco and Natale (2011).

Chapter 5

KUKA Smart Motion Generator

This chapter presents a software component, named *KUKA Smart Motion Generator (KSMG)*, developed at the KUKA Technology & Innovation Center during the course of the doctorate study. The software has been developed in the context of the funded research project *Internet of Construction (IoC)*[1] that aimed at simplifying the programming of industrial robots for prefabrication and on-site construction applications.

KSMG combines the constraint-based programming framework proposed in Chap. 3 with an expression engine that can process symbolic task descriptions, specified following the methodology formulated in Chap. 2. To generate robot motion, KSMG additionally embeds the generalized redundancy resolution introduced in Chap. 4. Thus, it implements the core modules of the stack presented in Fig. 2.11. The software is currently available within KUKA as research prototype, and a transfer to serial development is already planned.

5.1 Symbolic Task Description

A key feature of KSMG is the possibility to input task specifications according to a predefined description language. The symbolic description of a task is composed of four fields: Context, Motion Description, Constraints, and Stop Condition. The language syntax is based on the core capabilities of the C++ Mathematical Expression Library (ExprTk)[2], which have been suitable extended to satisfy the requirements of KSMG. For example, it is possible to refer to geometric primitives and objects from the environment model, and access their data through suitable operators. Moreover, a set of special functions allows for the definition of new geometric primitives, and for the specification of geometric constraints. Each element of the task description is further detailed below.

Context

The Context field allows for the definition of variables, which can conveniently hold useful data and be reused in the other fields of the task description. For example, it is

[1] http://www.internet-of-construction.com
[2] https://www.partow.net/programming/exprtk

possible to store objects data (objects and geometric primitives from the environment model are highlighted in blue)

```
cyl_radius := DeburringTool.Cylinder.radius;
```

or define parameters that can be later used as inputs for motion planning functions

```
acc_des := 0.1;
vel_cruise := 0.5;
path_lenght := 0.5;
```

.

It is also possible to define variable arrays, e.g., to store the coordinates of some points of interest

```
p1_ini := [-0.61, 4.3, 0.852];
p1_end := [-0.11, 4.3, 0.852];
p2_ini := [-0.6, 1.2, 1.2];
p2_end := [-0.1, 1.2, 1.2];
```

.

Motion Description

The Motion Description field allows for the definition of variables whose value depends on time. Similarly, geometric primitives whose pose varies over time can also be instantiated. The time dependency is expressed (and later resolved) through the use of the variable `time`, which is implicitly defined. KSMG also offers a set of functions (highlighted in green below), which can be used in the Motion Description field to define specific motion velocity profiles. For example, the line

```
s := trapezoidal(time, acc_des, vel_cruise, path_lenght);
```

defines a path parameter `s`, whose value varies from 0 to 1 over time, following a trapezoidal velocity profile. Moreover, geometric primitives with time-varying data can be instantiated in the Motion Description field. For example, the following lines

```
p1 := p1_ini + s*(p1_end - p1_ini);
p2 := p2_ini + s*(p2_end - p2_ini);
P1 := createPoint('P1', p1);
P2 := createPoint('P2', p2);
```

defines two points, `P1` and `P2`, which moves over time on a linear path with a velocity profile dictated by the evolution of `s`.

Constraints

The Constraints field is the core of the task description and enables the specification of constraints among geometric primitives, according to the methodology illustrated

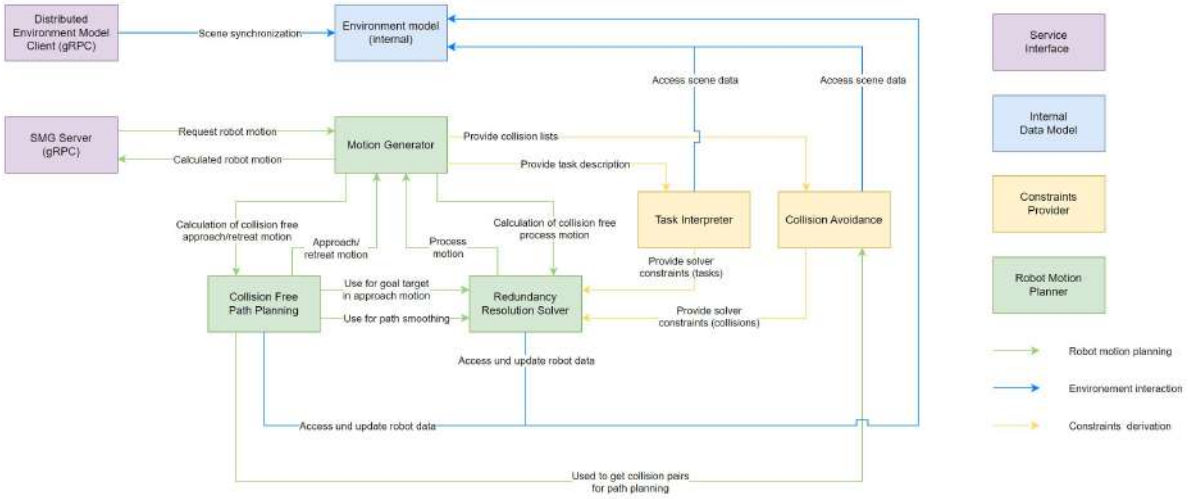


Figure 5.1: Architecture overview of the KUKA Smart Motion Generator service.

in Chap. 2. Such primitives can be implicitly defined in the environment model or originate from the definitions in the two previous fields. KSMG offers a continuously growing set of geometric constraints, each of which is applicable on a certain subset of geometric primitives. The following lines, for example, specify that a point `TCP` defined on `DeburringTool` must coincide with the previously defined `P1`, therefore realizing a point-to-point tracking application over time. Moreover, a second coincident constraint is imposed between the point `P2` and the line `Beam` defined on `LaserDevice`, therefore implementing a laser tracing application as shown in Sect. 2.2.1

```
Coincident(P1, DeburringTool.TCP));
Coincident(P2, LaserDevice.Beam));
```

.

Stop Condition

The last field of the task description specifies the condition(s) under which the execution of a task should stop, either due to fulfillment of the process requirements or errors. Thus, expressions in this field must resolve to boolean values. As an example, a task can be stopped once the planned trajectory involved in the task description has come to its end point. This situation can be easily monitored by checking whether the value of the path parameter `s` has reached the value 1

```
s >= 1.0
```

.

5.2 System Overview

KSMG is implemented as a service that accepts client requests containing a symbolic task description, and responds with a calculated robot motion. Additionally, KSMG communicates with an external server to synchronize its internal environment model with the scene the task description refers to. More specifically, geometric data (such

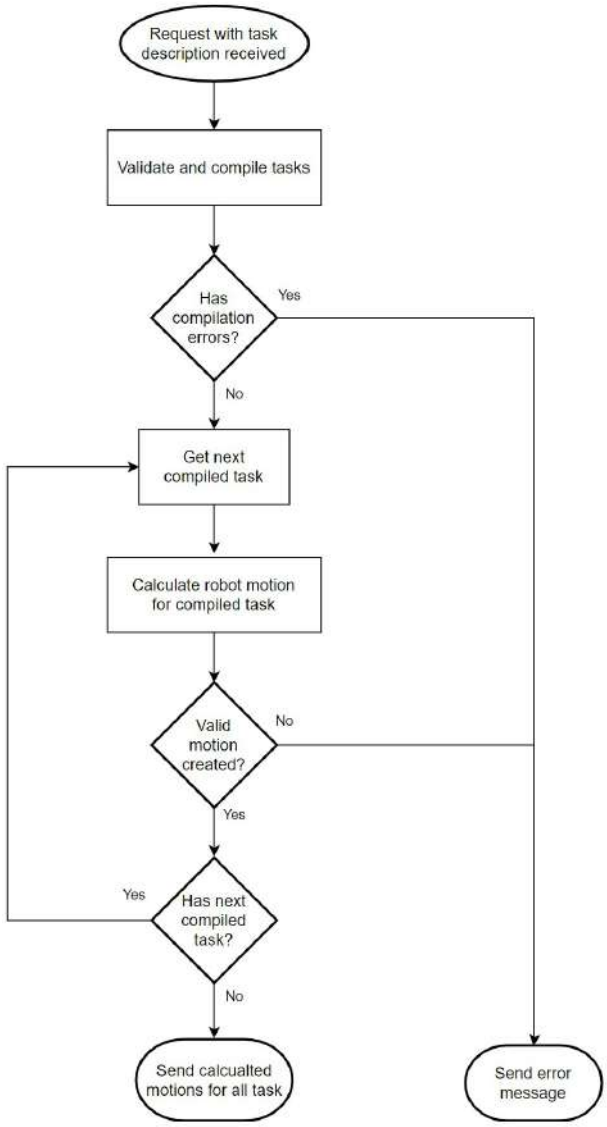


Figure 5.2: Handling of a service request by KUKA Smart Motion Generator.

as meshes and primitives), as well as kinematic data are transferred from the external server to the KSMG's internal environment model. A service request can also contain the specification of multiple tasks, which KSMG will then attempt to execute sequentially, returning the calculated robot motion for the entire sequence of tasks.

Figure 5.1 gives an overview of the modules composing KSMG. Although the service request is directly handled by the Motion Generator, it is mainly elaborated by the Task Interpreter. Indeed, this module is responsible for validating and compiling the task description. Possible errors and inconsistencies in the description at this stage cause the KSMG to return an error message to the client. Furthermore, the Task Interpreter is able to resolve all the symbols in the task description, by performing the binding to the appropriate geometric functions or the requested data in the internal environment model. Finally, the module can evaluate constraint functions and compute their first-

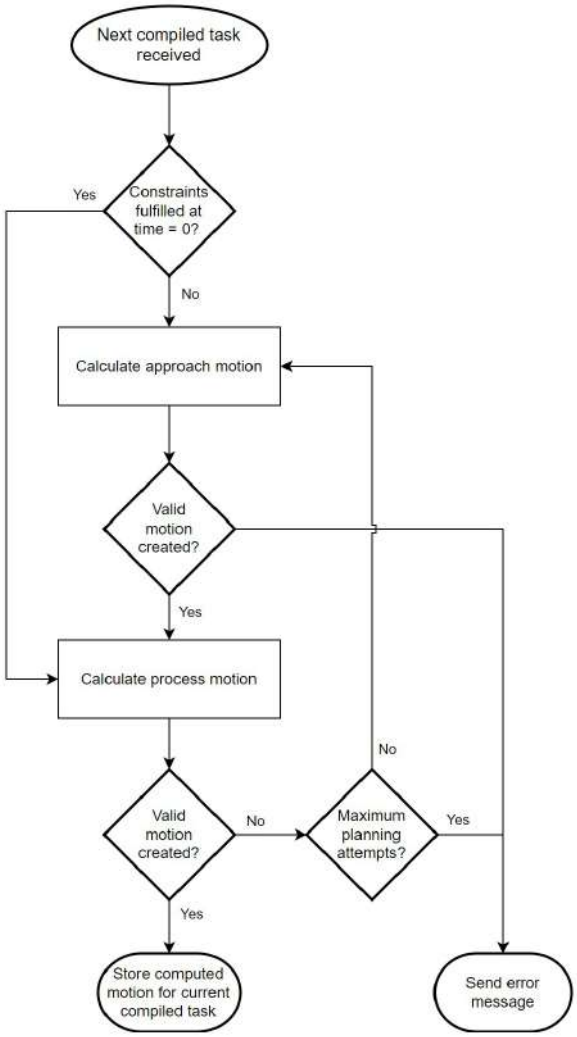


Figure 5.3: Computation of robot motion by KUKA Smart Motion Generator.

order derivatives. In other words, the Task Interpreter is able to transform the symbolic task description received by the KSMG into a numeric set of artificial constraints that follows the formalism introduced in Chap. 3. From this point on, the Motion Generator has all the necessary data to compute the robot motion. Figure 5.2 summarizes the steps executed by KSMG in handling a request.

The computation of the robot motion for each task in the task description proceeds through a sequence of steps, orchestrated by the Motion Generator module. First, it is verified whether the provided constraints are satisfied at the initial time $t = 0$. If this is not the case, the Motion Generator starts the computation of a so-called *approach motion*. In this phase, a set of possible target configurations is computed for each robot involved in the task, such that the constraints at $t = 0$ are fulfilled. This computation happens through an iterative employment of the redundancy resolution framework presented in Sect. 4.3, which is implemented in the Redundancy Resolution Solver module. Since each search is independent from the others, the calculations can also make use of parallel computing at this stage. At the end of every search, it is also verified whether the

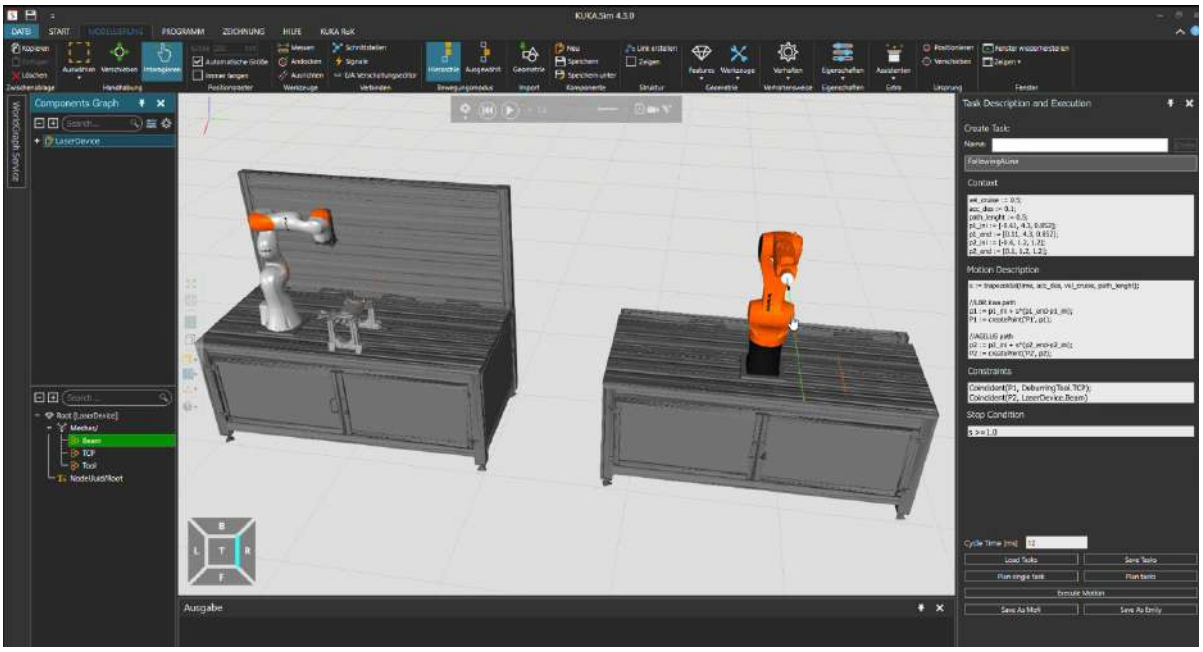
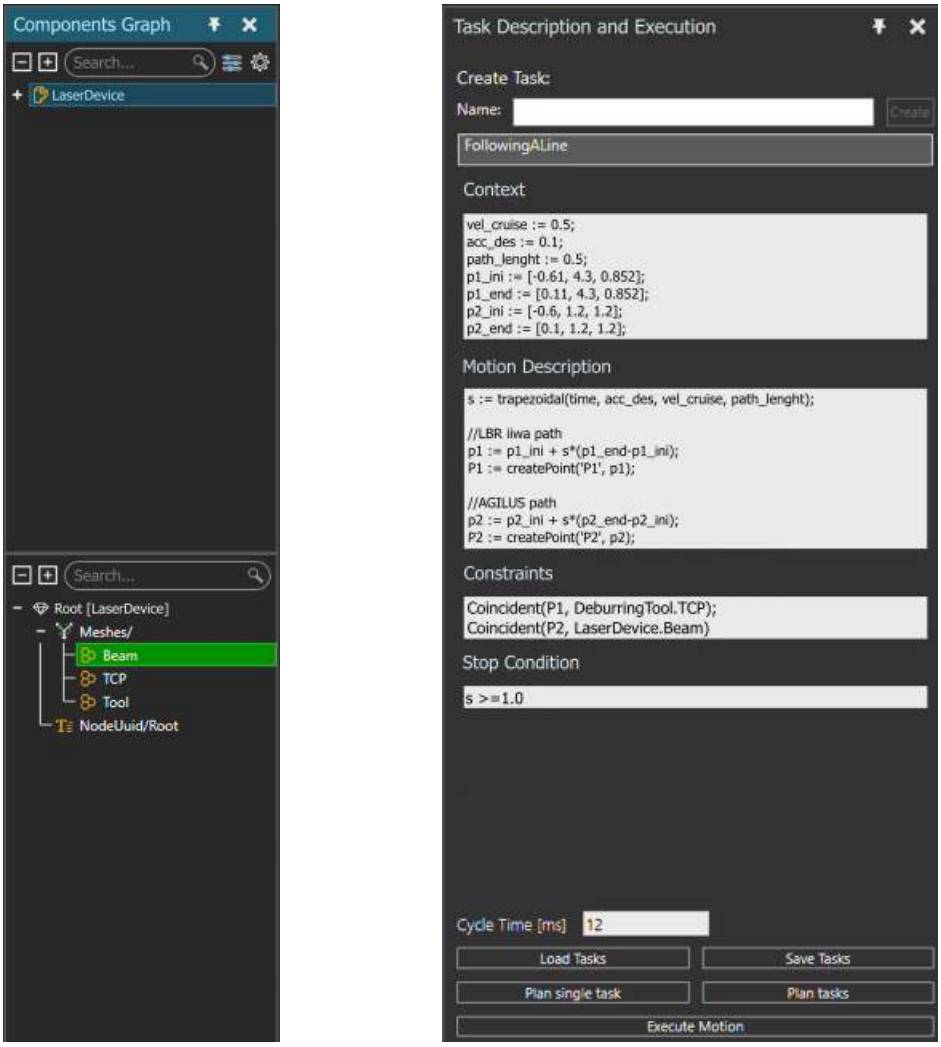


Figure 5.4: KUKA.Sim environment with KUKA Smart Motion Generator client extension.

obtained target configurations cause any collision with the environment or self-collision of the robot(s). Finally, a valid target configuration is selected for each robot and provided to the Collision Free Path Planning module. This module employs probabilistic methods, e.g. the AIT* by Strub and Gammell (2020), to compute collision-free joint space trajectories for all the robots involved in the approach motion. In case of any failure during planning, new target configurations are automatically selected from the valid set and a new planning attempt is performed. At every stage, the collision checking is performed by the Collision Avoidance module, which can access the environment model and perform mesh-to-mesh distance calculations.

Once the approach motion computation is concluded, the Motion Generator starts the computation of the so-called *process motion*, which will allow for the actual task execution. To this end, the target configurations computed for the approach motion are now set as initial configurations. Thus, starting at $t = 0$ and evolving with a fixed cycle time (additionally provided by the client), the Task Interpreter evaluates the constraints in the task description and sets up motion control optimization problems according to the procedure introduced in Chap. 3. Additional (inequality) constraints to the optimization problem are provided with the highest priority by the Collision Avoidance module, to ensure the robots avoid collisions throughout the task execution. Analogously, inequality constraints are automatically added for each robot to handle joint limitations. Finally, the generated optimization problem is solved by the Redundancy Resolution Solver using the framework from Sect. 4.3. In case of any failure, new initial configurations are selected for the process motion, for which the approach motion will need to be updated. Thus, a new planning attempt is performed. On the contrary, the calculation of the process motion ends when the expression in the Stop Condition field of



(a) (b)

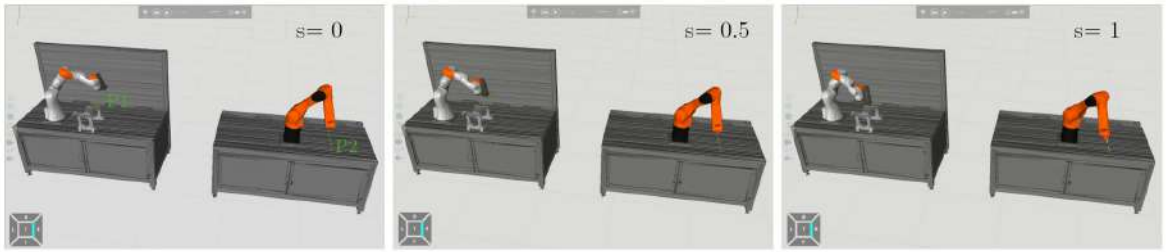
Figure 5.5: Enlargement of the Component Graph panel (a) and KSMG panel (b) from the KUKA.Sim window in Fig. 5.4.

the task description is verified. Figure 5.3 summarizes the described sequence of steps, integrating the flow chart from Fig. 5.2.

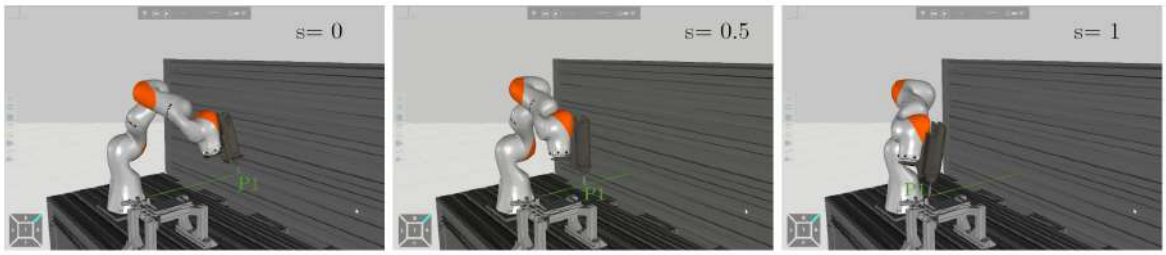
Optionally, KSMG can also be instructed to plan a final *retreat motion*, which cause all the robots in the scene to return to their initial configurations once all tasks are completed. As for the approach motion, also the retreat motion is computed employing the algorithms from the Collision-Free Path Planning module.

5.3 KUKA.Sim interface

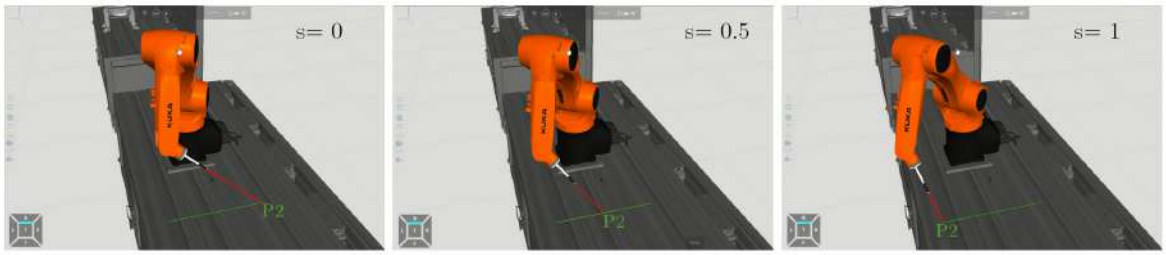
To facilitate the use of the KSMG service, an extension for KUKA.Sim has been developed, which allows the user to input a symbolic task description and start a KSMG client (Fig. 5.4).



(a) large view.



(b) KUKA LBR iiwa view.



(c) KUKA AGILUS view.

Figure 5.6: Process motion computed by the KSMG service for the task description in Fig. 5.5b.

KUKA.Sim is the official simulation software from KUKA, specifically developed for robotics applications. The software allows to create scenes with robots, tools and a variety of other objects available from a catalogue. Furthermore, it is possible to synchronize the scene with an external environment model server, which is the same accessed by the KSMG service. Finally, a number of tools and tabs allows to explore the topology of the scene and navigate through the available objects and geometric primitives. For example, the Components Graph panel (left tab in Fig. 5.4) shows the objects and the geometric primitives of a selected component. For the considered scene, clicking on the laser device mounted on the KUKA AGILUS shows, for example, the geometric primitive `Beam` (Fig. 5.5a), which represents the laser beam emitted by the device.

The KSMG extension provides instead the Task Description and Execution panel (right tab in Fig. 5.4). This panel allows the user to create a new task and fill in the four fields of its symbolic description. Additionally, the panel offers the possibility to set the simulation cycle time and some utility buttons to save and load task descriptions. Figure 5.5b shows an enlargement of the KSMG tab from Fig. 5.4, in which the task

Figure 5.7: Physical cell modeled in KUKA.Sim.

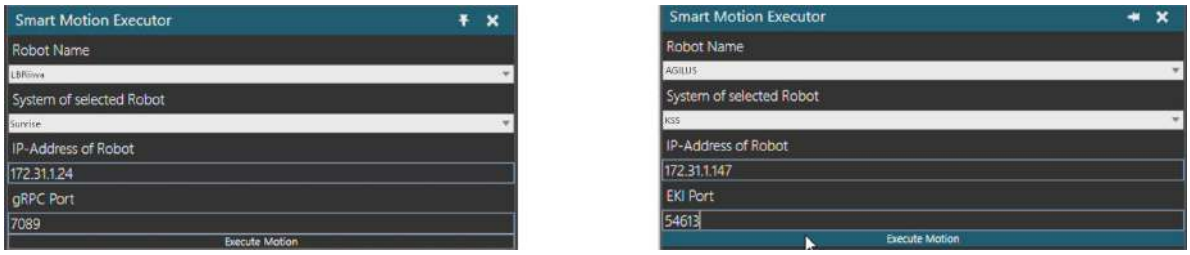
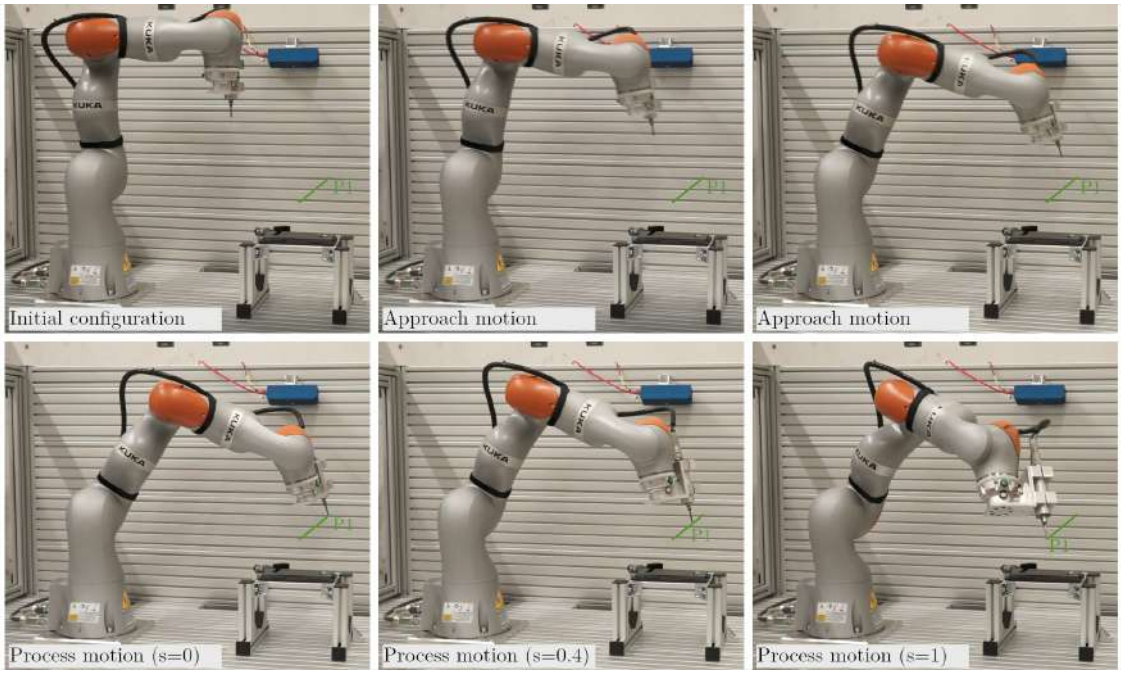


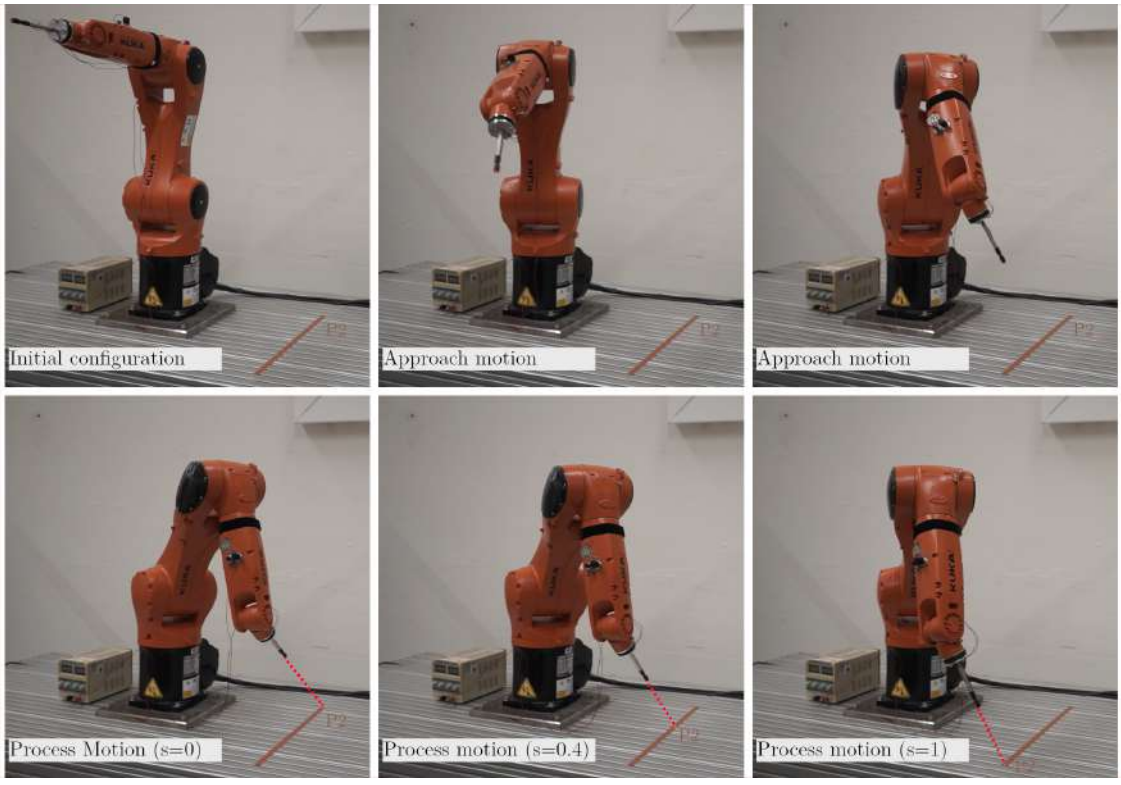
Figure 5.8: Smart Motion Executor panel.

`FollowingALine` has been created, and a task description provided as in the illustrative examples of Sect. 5.1. According to such description, the point `TCP` at the tip of the deburring tool is commanded to track the point `P1`, moving on a linear path with trapezoidal velocity profile. As the deburring tool is mounted at the flange of the KUKA LBR iiwa, such a specification will automatically generate motion for this robot. Analogously, since the geometric primitive `Beam` is required to trace the moving point `P2`, the KUKA AGILUS will also be commanded to move. The button `Plan single task` allows to send a request to the KSMG service to solve only the currently selected task. In case of multiple tasks, the button `Plan tasks` could instead be used to request the KSMG service to sequentially generate motion for all the tasks in the task description. Once the computation is completed, the generated motion can be reproduced via the button `Execute motion`. Figure 5.6 shows some snapshots of the process motion computed by the KSMG service for the two robots involved in the examined task.

Additionally, the generated motion can be directly reproduced on physical robots. Figure 5.7 shows the cell from which the scene in KUKA.Sim has been modeled. The interface to the robot controllers is realized through an additional custom panel, named Smart Motion Executor (Fig. 5.8). Here, it is possible to select the robot(s) to move, the operating system they run on, and their physical location over the network. Thus, by clicking the button `Execute motion`, the computed motion is transferred to the robot controller(s) as sequence of joint positions and automatically executed. Fig. 5.9 shows



(a) KUKA LBR iiwa



(b) KUKA AGILUS

Figure 5.9: Execution of the motion generated by the KSMG service on the physical robots.

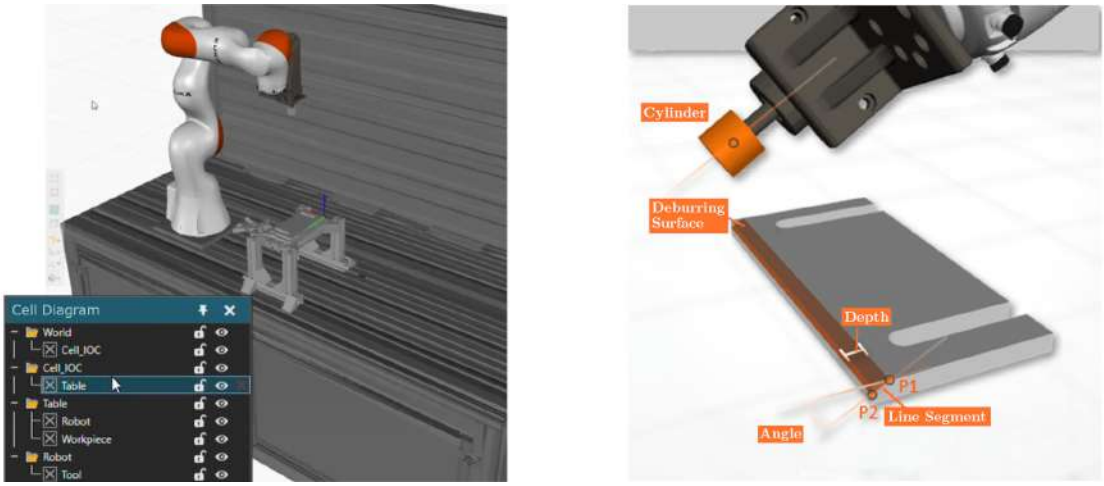


Figure 5.10: Deburring case study from the Internet of Construction project.

some snapshots of the two physical robots executing the motion computed by the KSMG service.

5.4 Deburring Application

This section presents a deburring application, addressed and solved using KSMG. The specific case study originates from the IoC project. The task requirement is to debur along the edges of a metal workpiece, placed on a rigid support that elevates it from the working table. The task has to be performed by a KUKA LBR iiwa, placed on the same working table. The deburring tool is mounted at the flange of the robot and has a cylindrical drill bit. Figure 5.10 gives an overview of the application setup. More specifically, the left image shows the modeled scene along with the Cell Diagram panel of KUKA.Sim, from which the hierarchy of objects composing the scene can be seen. It can be noticed that the cell modeled in KUKA.Sim coincides with the left part of the cell analyzed in Sect. 5.3. The right image in Fig. 5.10 shows instead the parameters and the geometric primitives involved in the task specification.

As already discussed in Sect. 2.2.3, the deburring process is characterized by two parameters. The first one, `depth`, indicates the desired penetration depth of the drill bit into the workpiece, whereas the second parameter, `angle`, represents the desired inclination of the drill bit. Using the two parameters, it is possible to identify the deburring surface, and therefore the points `P1` and `P2` that define the line segment the drill bit should be tangent with at the initial time. Shifting the points along the edges of the deburring surface over time, it is then possible to accomplish the deburring operation. As highlighted in Sect. 2.2.3, the combination of the straight workpiece edge and the cylindrical drill bit originates a deburring surface consisting of a virtual rectangle, generally internal to the workpiece, which can be used as an additional geometric primitive in the derivation of the task description.

All the geometric primitives involved in the task description are modeled in KUKA.Sim as features of the component they belong to (Fig. 5.11). For example, the tool compo-

nent contains the cylinder embedding the drill bit. Analogously, the workpiece component contains the initial points `P1` and `P2`, the line segment that connects them, and the related internal rectangle representing the deburring surface. All the modeled objects and primitives can be directly accessed and used in the symbolic task description simply by referring to their name.

The symbolic task description for the deburring process is given as

`Context`

```
edge_length := 0.095;
motion_dir := [0, -1, 0];
acc_des := 0.1;
p1_ini := getWorldPosition(P1);
p2_ini := getWorldPosition(P2);
```

`Motion Description`

```
s := trapezoidal(time, acc_des, vel_cruise, edge_lenght);
p1 := p1_ini + s*edge_length*motion_dir;
p2 := p2_ini + s*edge_length*motion_dir;
segmentP1P2 := createLineSegment('segmentP1P2', p1, p2);
plane := createPlane('plane', Rectangle);
```

, (5.1)

`Constraints`

```
Tangent(Cylinder, plane, segmentP1P2);
```

`Stop Condition`

```
s >=1;
```

where geometric primitives and constraints are provided according to the analysis in Sect. 2.2.3. Figure 5.12 shows some snapshots of the motion generated by the KSMG service using a cycle time of 5 ms. It can be noticed that both approach and process motions are collision free. Indeed, the drill bit and the workpiece are the only objects that come into contact in the scene, as desired. Moreover, the given constraints are fulfilled at all times, since the drill bit slides on the deburring surface following the provided velocity profile.

To test the KSMG framework in an even more challenging scenario, the same task is repeated in the cluttered environment in Fig. 5.13. In particular, the deburring surface (green rectangle) presents a total length of 0.5 m, and extends below a cube with edge length 0.3 m. This significantly reduces the space in which the robot is free to move. However, thanks to the exploitation of all the available redundant DoFs, KSMG is able to compute a robot motion fulfilling the task constraints at all time while avoiding any collision with the environment or with the robot itself (self collisions). At the same time, joint limits are also respected. The calculations are, as always, automatic and do not require any change in the task description. Thus, replanning the same task in a different scene comes with no effort for the user. The same would apply if the robot appointed to

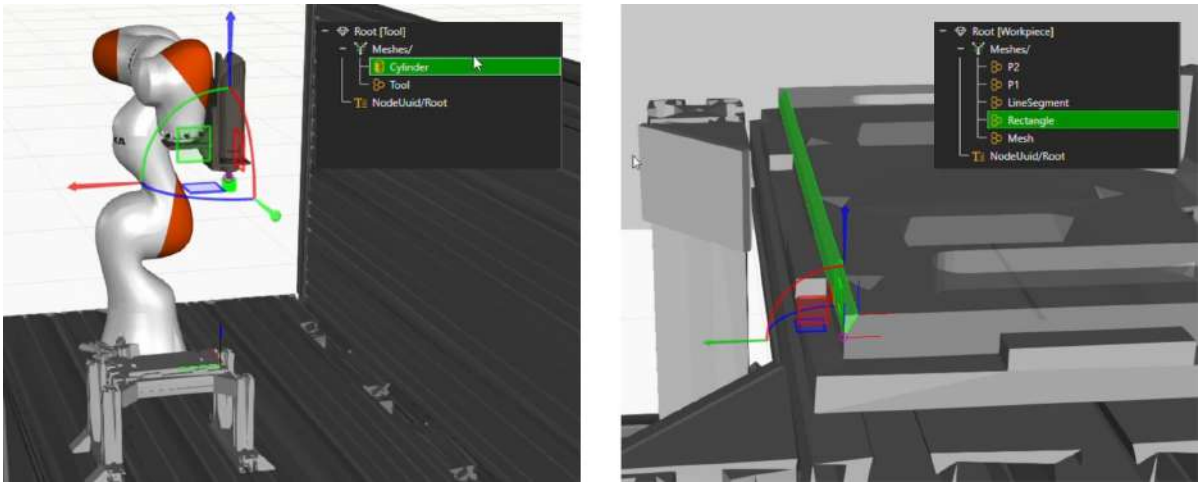


Figure 5.11: Component Graph panels for deburring tool and workpiece: the selected geometric primitives are highlighted in the KUKA.Sim scene.

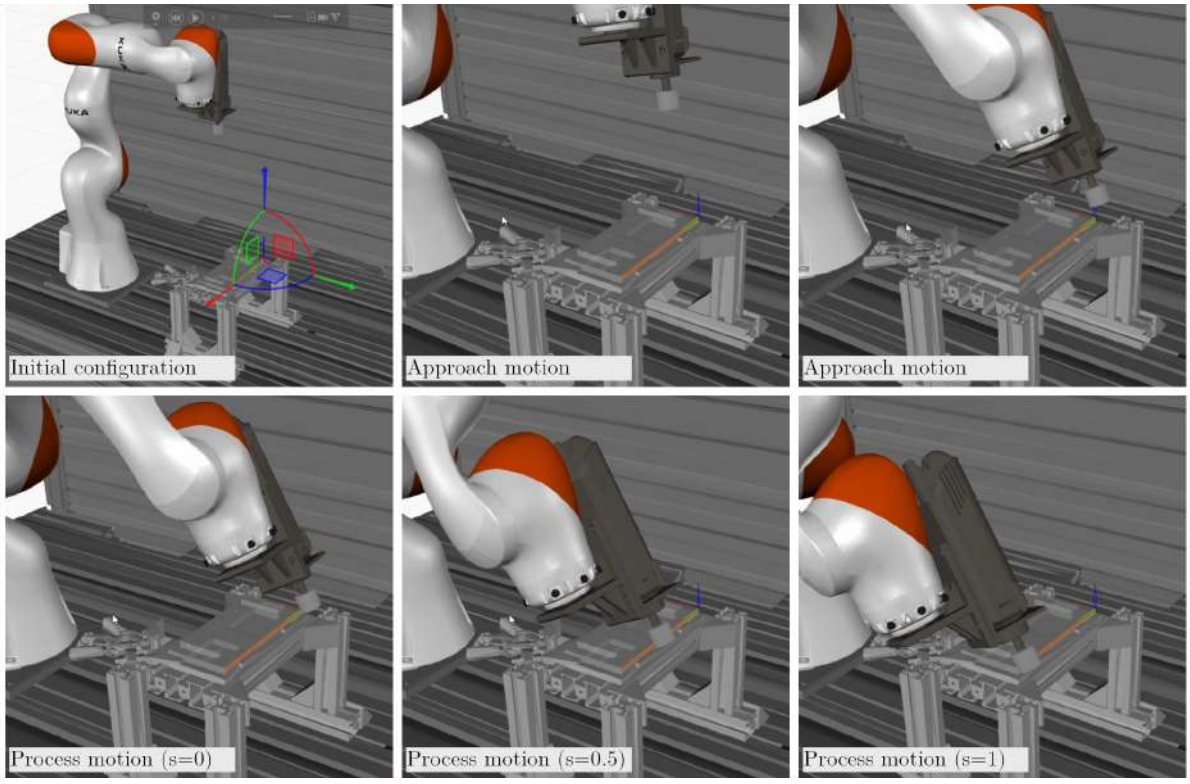


Figure 5.12: Execution of the motion generated by the KSMG service for the deburring task.

perform the task changed.

Differently from all previous examples, the generated motion is visualized in the viewer of the internal KSMG environment model. This allows to highlight the collision checking and distance computations performed by the KSMG algorithms. Indeed, the blu lines in Fig. 5.13 connects the points of the monitored collision pairs. These are

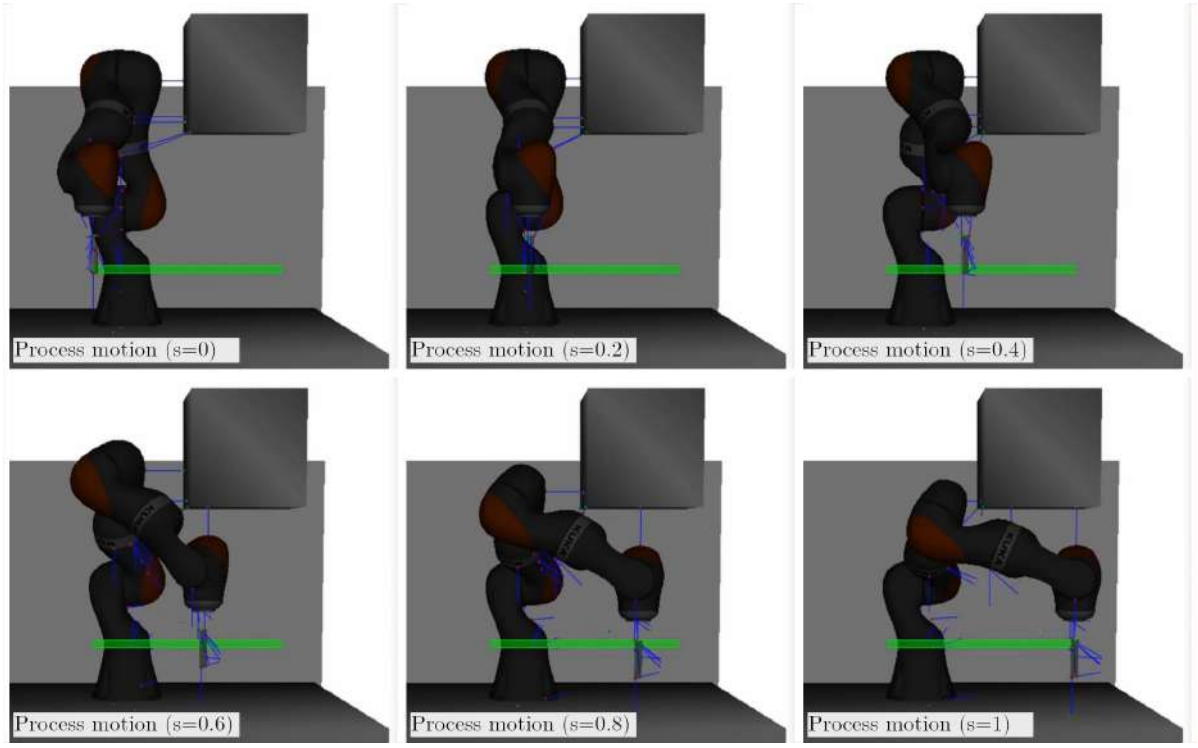


Figure 5.13: Execution of the motion generated by the KSMG service for the deburring task in the cluttered environment.

automatically updated at each instant of time by computing the points at minimum distance between meshes in the scene.

5.5 Discussion

This chapter has given an overview of the main features of KUKA Smart Motion Generator, a software component enabling the intuitive programming of industrial tasks for redundant robots. The software embeds all the concepts and the work presented in this thesis, from the specification of tasks as sets of geometric constraints, to their formulation within the constraint-based programming framework from Chap. 3, up to the resolution of redundant DoFs and the computation of the joint commands. Through the many simulations executed during the course of doctorate study, KSMG has shown the validity of the ideas developed in this thesis. In particular, the fact that all computations are automatically generated out of a symbolic task description, which is completely independent from the specific robot(s) and the scene at hand, makes reprogramming of an application particularly effortless for the user.

Recalling the programming stack in Fig. 2.11, it is possible to notice that, in its current state, KSMG implements the lower-level API, in which symbolic constraints can be directly provided. Extending it to implement the higher-level API outlined in Sect. 2.3 is certainly a direction for future work.

Chapter 6

Conclusion

This thesis provides a solution for the intuitive programming of redundant robots. Although the focus of this work is on the execution of industrially relevant applications, the methods presented in this thesis are general and could be applied to other fields. Specific contributions of this thesis include:

- identification of suitable paradigms for intuitive task description, enabling robot programming to users with limited robotics expertise and capturing the minimum number of constraints that are necessary to perform a given task;
- formulation of a novel constraint-based programming framework for the automatic translation of task descriptions (possibly involving multiple robots) into motion control problems;
- formulation of a novel general framework for hierarchical redundancy resolution under arbitrary constraints;
- analysis of some critical aspects of discrete-time implementations of redundancy resolution algorithms, specifically regarding integration methods and convergence properties;
- presentation of a software component for intuitive programming of redundant proving the validity of the developed methods.

This work postulates that industrial tasks are more naturally described by spatial relations (geometric constraints) between objects. Such relations are easily defined and understood also by non-expert programmers, as they do not involve any specific robotics knowledge. Moreover, spatial relations enable complete decoupling of the task description from the particular robot(s) appointed to execute the task. Finally, they allow for the specification of a minimum number of necessary constraints. This way, the robot(s) are regarded as redundant whenever possible. This thesis additionally suggests that industrial tasks can be intuitively formulated on two different levels of abstraction. On the higher level, a task can be represented as an action involving a certain number of objects and parameters; on the lower level, constraints can be directly specified, exploiting suitable geometric features of the objects involved.

Intuitive programming also means that, once the task requirements are formulated, the motion of the robot(s) involved should be automatically computed, possibly exploiting any available redundant DoF. The general framework for constraint-based programming presented in this thesis serves this purpose. Compared to already existing solutions, the framework does not require the manual specification of additional feature frames or coordinates, and it is capable of generating dynamically-consistent optimization problems for motion control. Furthermore, multiple robots with possibly multiple PoC are automatically handled in a unified fashion. The generality of the framework is enhanced by the fact that classic programming based on Cartesian frames can be obtained as a special case, as it is for joint motion commands.

Handling redundant robots implies solving motion control optimization problems. Here, this thesis presents a general framework for the motion control of redundant robots. Thanks to a unified formulation of the redundancy resolution problem, the framework can arbitrary resolve redundancy at velocity, acceleration or torque level. This provides high flexibility with respect to the different application domains and robot interfaces. The framework exploits the definition of a generalized control problem, solved using a novel redundancy resolution algorithm. In particular, the thesis presents the eSNS method, obtained by extending an existing algorithm from the literature in several aspects. The eSNS features hierarchical management of priority levels, handling of (equality and inequality) constraints defined in any space, task scaling, and a highly configurable cost function. Additionally, the thesis introduces two variants, Fast-eSNS and Opt-eSNS, , which tackle important aspects such as computation efficiency and optimality of the solution.

Redundancy resolution algorithms are implemented on digital controllers, yet the discrete-time nature of the resulting system is often overlooked. The two studies presented in this thesis analyze some relevant aspects. The first study consists of a convergence analysis for the solution of equality constraints considering a task function with time dependency. The study provides sufficient conditions to the convergence of the redundancy resolution algorithm depending on the value of the gain, the initial value of the task function, and a third parameter related to "rate of change" of the task function over time. A second study proposes an investigation on how different integration methods can affect the performance of discrete-time redundancy resolution algorithms in terms of stability, accuracy and smoothness of the solution. In particular, the comparison between explicit Euler and RK4 in a pure inverse kinematics application shows that significant benefits are observed when using RK4 to track time-varying Cartesian references and move through kinematic singularities. On the other hand, explicit Euler is able to provide good performance in different conditions with significantly lower computational cost.

Finally, the thesis includes an overview of KSMG, a software component enabling the intuitive programming of industrial tasks for redundant robots according to the concepts developed throughout the thesis. Indeed, KSMG includes the specification of tasks as sets of geometric constraints, their formulation within the constraint-based programming framework proposed in Chap. 3, and the computation of the joint commands with automatic redundancy resolution using the methods in Chap. 4.

The methodologies proposed in this thesis already find applications in industry. In

particular, KSMG is used within the KUKA Technology & Innovation Center to intuitively program multi-robot industrial applications in the context of research projects, and an official integration in KUKA.Sim is already planned. Nevertheless, the topics discussed in this thesis open to many future extensions. First of all, the higher level API depicted in Sect. 2.3 could be implemented in KSMG to prove its validity and effectiveness in modeling task requirements. The task description itself could also be further elaborated, so as to a find a suitable model for the description of tasks involving force interaction. This would also require an extension of the constraint-based programming framework presented in Chap. 3, to mathematically support the handling of force constraints. Such development would enable intuitive programming of a lot more applications from other relevant robotics fields. A critical aspect of the lower level API in Sect. 2.3 is the possibility of unintentional design of conflicting geometric constraints. Suitable strategies could be therefore developed to avoid such a case or to promptly detect and inform the user about it. Also the redundancy resolution framework from Sect. 4.3 offers a number of possible extensions. Most importantly, a combination of the Fast-eSNS and Opt-eSNS would bring faster computation times while retaining optimal control inputs. Preliminary work has already been carried out in this direction, showing promising results. Finally, the convergence analysis from Sect. 4.4.2 could be extended to the case of multiple priority levels.

Bibliography

Aertbeliën, E. and J. De Schutter (2014). "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1540–1546.

Alsultan, T., H. M. Ali, and Q. Y. Hamid (2018). "A numerical approach for solving problems in robotic arm movement". In: *Production & Manufacturing Research* 6.1, pp. 385–395.

Ambler, A. P. and R. J. Popplestone (1975). "Inferring the positions of bodies from specified spatial relationships". In: *Artificial Intelligence* 6.2, pp. 157–174.

Antonelli, G. (2014). *Underwater robots*. Vol. 3. Springer.

Antonelli, G., S. Chiaverini, and G. Fusco (2003). "A new on-line algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits". In: *IEEE Transactions on Robotics and Automation* 19.1, pp. 162–167.

Baizid, K., G. Giglio, F. Pierri, M. A. Trujillo, G. Antonelli, F. Caccavale, A. Viguria, S. Chiaverini, and A. Ollero (2015). "Experiments on behavioral coordinated control of an unmanned aerial vehicle manipulator system". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 4680–4685.

Balestrino, A., G. De Maria, and L. Sciavicco (1984). "Robust control of robotic manipulators". In: *IFAC Proceedings Volumes* 17.2, pp. 2435–2440.

Bartels, G., I. Kresse, and M. Beetz (2013). "Constraint-based movement representation grounded in geometric features". In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 547–554.

Basile, F., F. Caccavale, P. Chiacchio, J. Coppola, and C. Curatella (2012). "Task-oriented motion planning for multi-arm robotic systems". In: *Robotics and Computer-Integrated Manufacturing* 28.5, pp. 569–582.

Bjerkeng, M., P. Falco, C. Natale, and K. Y. Pettersen (2014). "Stability analysis of a hierarchical architecture for discrete-time sensor-based control of robotic systems". In: *IEEE Transactions on Robotics* 30.3, pp. 745–753.

Bruyninckx, H. and J. De Schutter (1996). "Specification of force-controlled actions in the" task frame formalism"-a synthesis". In: *IEEE transactions on robotics and automation* 12.4, pp. 581–589.

Chiacchio, P. and S. Chiaverini (1995). "Coping with joint velocity limits in first-order inverse kinematics algorithms: Analysis and real-time implementation". In: *Robotica* 13.5, pp. 515–519.

Chiacchio, P., S. Chiaverini, L. Sciavicco, and B. Siciliano (1991). "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmen-

tation and task priority strategy". In: *The International Journal of Robotics Research* 10.4, pp. 410–425.

Chiaverini, S. (1997). "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators". In: *IEEE Transactions on Robotics and Automation* 13.3, pp. 398–410.

Das, H., J. E. Slotine, and T. B. Sheridan (1988). "Inverse kinematic algorithms for redundant systems". In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE, pp. 43–48.

De Luca, A., G. Oriolo, and B. Siciliano (1992). "Robot redundancy resolution at the acceleration level". In: *Laboratory Robotics and Automation* 4, pp. 97–97.

De Maria, G. and R. Marino (1985). "A discrete algorithm for solving the inverse kinematic problem for robotic manipulators". In: *Proceedings 2nd International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 275–282.

De Schutter, J., T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx (2007). "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty". In: *The International Journal of Robotics Research* 26.5, pp. 433–455.

Decre, W., R. Smits, H. Bruyninckx, and J. De Schutter (2009). "Extending iTaSC to support inequality constraints and non-instantaneous task specification". In: *2009 IEEE International Conference on Robotics and Automation*, pp. 964–971.

Decré, W., H. Bruyninckx, and J. De Schutter (2013). "Extending the iTaSC constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1941–1948.

Di Lillo, P., G. Antonelli, and C. Natale (2021). "Effects of dynamic model errors in task-priority operational space control". In: *Robotica* 39.9, pp. 1642–1653.

Di Lillo, P., F. Pierri, G. Antonelli, F. Caccavale, and A. Ollero (2021). "A framework for set-based kinematic control of multi-robot systems". In: *Control Engineering Practice* 106, p. 104669.

Di Vito, D., C. Natale, and G. Antonelli (2017). "A comparison of damped least squares algorithms for inverse kinematics of robot manipulators". In: *IFAC-PapersOnLine* 50.1, pp. 6869–6874.

Dietrich, A. and C. Ott (2019). "Hierarchical impedance-based tracking control of kinematically redundant robots". In: *IEEE Transactions on Robotics* 36.1, pp. 204–221.

Escande, A., N. Mansard, and P.-B. Wieber (2014). "Hierarchical quadratic programming: Fast online humanoid-robot motion generation". In: *The International Journal of Robotics Research* 33.7, pp. 1006–1028.

Espiau, B., F. Chaumette, and P. Rives (1992). "A new approach to visual servoing in robotics". In: *ieee Transactions on Robotics and Automation* 8.3, pp. 313–326.

Falco, P. and C. Natale (2011). "On the stability of closed-loop inverse kinematics algorithms for redundant robots". In: *IEEE Transactions on Robotics* 27.4, pp. 780–784.

Ferreau, H. J., C. Kirches, A. Potschka, H. G. Bock, and M. Diehl (2014). "qpOASES: A parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6.4, pp. 327–363.

Fiore, M. D. and C. Natale (2021). "Discrete-time closed-loop inverse kinematics: A comparison between Euler and RK4 methods". In: *2021 29th Mediterranean Conference on Control and Automation (MED)*, pp. 584–589.

Fiore, M. D., G. Meli, A. Ziese, B. Siciliano, and C. Natale (2023). "A General Framework for Hierarchical Redundancy Resolution Under Arbitrary Constraints". In: *IEEE Transactions on Robotics*, pp. 1–20.

Flacco, F. and A. De Luca (2013a). "Fast redundancy resolution for high-dimensional robots executing prioritized tasks under hard bounds in the joint space". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2500–2506.

Flacco, F. and A. De Luca (2013b). "Optimal redundancy resolution with task scaling under hard bounds in the robot joint space". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3969–3975.

Flacco, F., A. De Luca, and O. Khatib (2012). "Prioritized multi-task motion control of redundant robots under hard joint constraints". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 3970–3977.

Flacco, F., A. De Luca, and O. Khatib (2015). "Control of redundant robots under hard joint constraints: Saturation in the null space". In: *IEEE Transactions on Robotics* 31.3, pp. 637–654.

Fruchard, M., P. Morin, and C. Samson (2006). "A framework for the control of nonholonomic mobile manipulators". In: *The International Journal of Robotics Research* 25.8, pp. 745–780.

Greville, T. (1960). "Some applications of the pseudoinverse of a matrix". In: *SIAM review* 2.1, pp. 15–22.

Guo-rong, W. and C. Yong-lin (1986). "A Recursive Algorithm for Computing the Weighted Moore-Penrose Inverse". In: *Journal of Computational Mathematics*, pp. 74–85.

Hoffman, E. M., A. Laurenzi, L. Muratore, N. G. Tsagarakis, and D. G. Caldwell (2018). "Multi-priority cartesian impedance control based on quadratic programming optimization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 309–315.

Hollerbach, J. M. (1983). "Dynamic scaling of manipulator trajectories". In: *1983 American Control Conference*. IEEE, pp. 752–756.

Kanoun, O., F. Lamiraux, and P.-B. Wieber (2011). "Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task". In: *IEEE Transactions on Robotics* 27.4, pp. 785–792.

Khatib, O. (1983). "Dynamic control of manipulator in operational space". In: *Proc. 6th IFToMM World Congress on Theory of Machines and Mechanisms*, pp. 1128–1131.

Khatib, O. (1986). "Real-time obstacle avoidance for manipulators and mobile robots". In: *Autonomous robot vehicles*. Springer, pp. 396–404.

Kresse, I. and M. Beetz (2012). "Movement-aware action control—Integrating symbolic and control-theoretic action execution". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3245–3251.

Kuhn, H. and A. Tucker (1951). "Nonlinear programming". In: *Proc. 2nd Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492.

Kutta, W. (1901). "Beitrag zur naherungsweisen Integration totaler Differentialgleichungen". In: *Z. Math. Phys.* 46, pp. 435–453.

Lachner, J., F. Allmendinger, E. Hobert, N. Hogan, and S. Stramigioli (2021). "Energy budgets for coordinate invariant robot control in physical human–robot interaction". In: *The International Journal of Robotics Research* 40.8-9, pp. 968–985.

Lachner, J., V. Schettino, F. Allmendinger, M. D. Fiore, F. Ficuciello, B. Siciliano, and S. Stramigioli (2020). "The influence of coordinates in robotic manipulability analysis". In: *Mechanism and machine theory* 146, p. 103722.

Lakshmikantham, V. and D. Trigiante (2002). *Theory of difference equations numerical methods and applications*. CRC Press.

Lee, J., N. Mansard, and J. Park (2012). "Intermediate desired value approach for task transition of robots in kinematic control". In: *IEEE Transactions on Robotics* 28.6, pp. 1260–1277.

Liegeois, A. et al. (1977). "Automatic supervisory control of the configuration and behavior of multibody mechanisms". In: *IEEE transactions on systems, man, and cybernetics* 7.12, pp. 868–871.

Liu, M., Y. Tan, and V. Padois (2016). "Generalized hierarchical control". In: *Autonomous Robots* 40.1, pp. 17–31.

Maciejewski, A. A. and C. A. Klein (1985). "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments". In: *The international journal of robotics research* 4.3, pp. 109–117.

Mansard, N., O. Khatib, and A. Kheddar (2009). "A unified approach to integrate unilateral constraints in the stack of tasks". In: *IEEE Transactions on Robotics* 25.3, pp. 670–685.

Mansard, N., O. Stasse, P. Evrard, and A. Kheddar (2009). "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks". In: *2009 International conference on advanced robotics*. IEEE, pp. 1–6.

Marchand, É., F. Chaumette, and A. Rizzo (1996). "Using the task function approach to avoid robot joint limits and kinematic singularities in visual servoing". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*. Vol. 3. IEEE, pp. 1083–1090.

Mason, M. T. (1981). "Compliance and force control for computer controlled manipulators". In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.6, pp. 418–432.

Mezouar, Y. and F. Chaumette (2002). "Path planning for robust image-based control". In: *IEEE transactions on robotics and automation* 18.4, pp. 534–549.

Moe, S., G. Antonelli, A. R. Teel, K. Y. Pettersen, and J. Schrimpf (2016). "Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results". In: *Frontiers in Robotics and AI* 3, p. 16.

Muñoz Osorio, J. D., M. D. Fiore, and F. Allmendinger (2018). "Operational Space Formulation Under Joint Constraints". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 51814. American Society of Mechanical Engineers.

Nikravesh, P. E. (1988). *Computer-aided analysis of mechanical systems*. Prentice-Hall, Inc.

Osorio, J. D. M., F. Allmendinger, M. D. Fiore, U. E. Zimmermann, and T. Ortmaier (2019). "Physical human-robot interaction under joint and cartesian constraints". In: *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 185–191.

Osorio, J. D. M. and F. Allmendinger (2022). "A Suitable Hierarchical Framework with Arbitrary Task Dimensions under Unilateral Constraints for physical Human Robot Interaction". In: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 66–72.

Ott, C., A. Dietrich, and A. Albu-Schäffer (2015). "Prioritized multi-task compliance control of redundant manipulators". In: *Automatica* 53, pp. 416–423.

Peters, J., M. Mistry, F. Udwadia, J. Nakanishi, and S. Schaal (2008). "A unifying framework for robot control with redundant DOFs". In: *Autonomous Robots* 24.1, pp. 1–12.

Quiroz-Omaña, J. J. and B. V. Adorno (2019). "Whole-body control with (self) collision avoidance using vector field inequalities". In: *IEEE Robotics and Automation Letters* 4.4, pp. 4048–4053.

Rall, L. B. (1981). *Automatic differentiation: Techniques and applications*. Springer.

Runge, C. (1895). "Über die numerische Auflösung von Differentialgleichungen". In: *Mathematische Annalen* 46.2, pp. 167–178.

Samson, C., B. Espiau, and M. L. Borgne (1991). *Robot control: the task function approach*. Oxford University Press, Inc.

Sariyildiz, E. and H. Temeltas (2011). "Performance analysis of numerical integration methods in the trajectory tracking application of redundant robot manipulators". In: *International Journal of Advanced Robotic Systems* 8.5, p. 63.

Schettino, V., M. D. Fiore, C. Pecorella, F. Ficuciello, F. Allmendinger, J. Lachner, S. Stramigioli, and B. Siciliano (2020). "Geometrical Interpretation and Detection of Multiple Task Conflicts using a Coordinate Invariant Index". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6613–6618.

Scheurer, C., M. D. Fiore, S. Sharma, and C. Natale (2016). "Industrial implementation of a multi-task redundancy resolution at velocity level for highly redundant mobile manipulators". In: *Proceedings of ISR 2016: 47st International Symposium on Robotics*. VDE, pp. 1–9.

Sciavicco, L. and B. Siciliano (1986). "Coordinate transformation: A solution algorithm for one class of robots". In: *IEEE transactions on systems, man, and cybernetics* 16.4, pp. 550–559.

Senthilkumar, S., M. Lee, and T.-K. Kwon (2013). "Rk-methods for robot application problems". In: *International journal of advanced smart convergence* 2.1, pp. 18–20.

Sentis, L. and O. Khatib (2004). "Prioritized multi-objective dynamics and control of robots in human environments". In: *4th IEEE/RAS International Conference on Humanoid Robots, 2004.* Vol. 2. IEEE, pp. 764–780.

Sentis, L. and O. Khatib (2005). "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives". In: *International Journal of Humanoid Robotics* 2.04, pp. 505–518.

Siciliano, B., L. Sciavicco, L. Villani, and G. Oriolo (2009). "Differential Kinematics and Statics". In: *Robotics: modelling, planning and control*. Ed. by B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Springer Science & Business Media. Chap. 3.

Siciliano, B. and J.-J. Slotine (1991). "A general framework for managing multiple tasks in highly redundant robotic systems". In: *Fifth International Conference on Advanced Robotics*. IEEE, pp. 1211–1216.

Somani, N., A. Gaschler, M. Rickert, A. Perzylo, and A. Knoll (2015). "Constraint-based task programming with CAD semantics: From intuitive specification to real-time control". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2854–2859.

Strub, M. P. and J. D. Gammell (2020). "Adaptively Informed Trees (AIT*): Fast asymptotically optimal path planning through adaptive heuristics". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3191–3198.

Suleiman, W., K. Ayusawa, F. Kanehiro, and E. Yoshida (2018). "On prioritized inverse kinematics tasks: Time-space decoupling". In: *2018 IEEE 15th International Workshop on Advanced Motion Control (AMC)*. IEEE, pp. 108–113.

Vanthienen, D., T. De Laet, H. Bruyninckx, W. Decré, and J. De Schutter (2012). "Force-sensorless and bimanual human-robot comanipulation implementation using itasc". In: *IFAC Proceedings Volumes* 45.22, pp. 759–766.

Whitney, D. E. (1969). "Resolved motion rate control of manipulators and human prostheses". In: *IEEE Transactions on man-machine systems* 10.2, pp. 47–53.

Ziese, A., M. D. Fiore, J. Peters, U. E. Zimmermann, and J. Adamy (2020). "Redundancy resolution under hard joint constraints: a generalized approach to rank updates". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7447–7453.