Università degli Studi di Salerno

Dottorato di Ricerca in Informatica e Ingegneria dell'Informazione
Ciclo 32 – a.a 2018/2019

TESI DI DOTTORATO / PH.D. THESIS

Dynamic Programming for Optimal Planning and Control of Redundant Robot Manipulators

ENRICO FERRENTINO

SUPERVISOR: **PROF. PASQUALE CHIACCHIO**

PHD PROGRAM DIRECTOR: **PROF. PASQUALE CHIACCHIO**

Dipartimento di Ingegneria dell'Informazione ed Elettrica
e Matematica Applicata
Dipartimento di Informatica

Contents

Chapter 1

Introduction

1.1 The big picture

The financial crisis of 2007-2008 has marked an evident discontinuity in the way progress is conceived. Many countries world-wide realized the weaknesses of their economical systems, strictly connected to the impoverishment of their production chain and the transfer of employment from high-productivity sectors to less productive or non-productive ones. In the last decade, several countermeasures have been taken by government bodies to re-invent and re-structure their industries, to convert their productive system towards jobs and goods of an high technological content, to the purpose of increasing the competitiveness of their economies and retake a slow, but continuous growth.

To this end, the strategical plan of many countries looks towards a new model of economical development, that must be efficient, sustainable and inclusive. *Smart factory* and *Industry 4.0* certainly are popular keywords that well depict the change in the way our companies organize, exchange and produce. In this context, automation and robotics play a fundamental role. Industrial automation was progressively introduced and developed in the second half of the last century and the beginning of 2000s. In 1961, General

Motors started automating spot welding operations in cars manufacturing with the Unimate robot. The first programmable logic controller was developed at the end of the same decade. In the '70s, industrial automation started spreading in most manufacturing companies around the world, while more versatile robotic arms arose on the market, capable of performing a variety of different tasks, including pick and place operations and assembly of mechanical parts. With the advent of personal computers, new programming languages and networks, industrial automation became distributed, cheaper and way more flexible. Direct-drive designs for robotic manipulators started appearing in the '80s, following the revolutionary studies of Takeo Kanade. In the following decades, manufacturing robots and apparatuses started becoming synchronized, allowing for full-plant automation in many sectors. While, nowadays, collaborative and adaptive systems are already the reality, the current challenge consists in making automated manufacturing smarter, safer and more advanced. On the one hand, the market is constantly shifting towards a customer-centered model, asking for an higher level of product customization, on the other, factories need to increase their productivity and efficiency. In the background, the environment poses the unique challenge of making the production sustainable.

From the technical viewpoint, this general vision translates, more practically, in requirements of flexibility, efficiency and sustainability that the robotic and autonomous systems, that make up the new model of industry, have to comply with. Systems are flexible when they can easily re-adapt to different objectives and constraints in a short time, with negligible efforts and costs. Flexibility and re-adaptation are also linked to availability, intended as the characteristic of a plant to be functioning for the most of its lifetime, allowing for automatic fault detection and diagnosis and online maintenance. Systems are efficient when they are utilized at the maximum of their capacity, when they are optimized in a way that their limit is structural and not connected to the way they are operated. Efficiency is naturally linked to sustainability, in that resources are not wasted, but are totally dedicated to the

fulfillment of the production task.

There is no efficiency without the possibility of measuring the performance of a system, and indeed, one of the pillars of the industrial revolution we are living consists in the capability of collecting data and providing means to estimate the margin for improvement. While, in the past, the adoption of robotic systems was confined to a few big manufacturing industries, nowadays they are affordable and available even to medium and small enterprises. Day by day, their usage is becoming widespread, which further increases the impact of technologies and algorithms that are able to optimize their behavior and make them more efficient.

Manufacturing industries are not the only playground for advanced robotics, but other flourishing markets, and as strategical for the economy of the future, are aerospace and service robotics. The interesting matter is that the same requirements of efficiency, flexibility and sustainability hold in these fields. The development of aerospace technologies is supported by the spillover it entails into other sectors and by the evidence that the challenges it introduces contribute increasing the competitiveness of industries. Robots are employed for the exploration and colonization of celestial bodies. They allow a reduction of mission costs, free humans from dangerous tasks and perform activities otherwise unfeasible for men. Robots act as human precursors, explore unknown environments providing useful information and preparing them for human settlement, explore extreme or remote areas where human deployment is not currently feasible and support men in the maintenance of current assets [1]. But applications are not limited to exploration and colonization, as robots are employed or regarded as future enablers for missions involving on-orbit servicing [2], active space debris removal [3] and in-space manufacturing [4]. In space, because of limited energy sources, power efficiency is certainly central [5], but availability, reliability and safety are also important factors.

1.2 Motivation and scope

Depending on the application and characteristics of the specific robotic system, several definitions of efficiency may be given. In automated manufacturing industries, where the objective is to increase the throughput of the plant, a system is efficient if it can output goods as fast as possible. In a planetary construction scenario, where the energy sources are limited and expensive, the efficiency lies in the capability of completing the assigned operation by consuming as less energy as possible. These are only some examples, but efficiency can be found in the quality of the product that is manufactured or the task that is performed, in the lifetime of the robotic system itself or costs associated to periodic maintenance, among other things. All these cases have a common denominator that is the definition of one or more performance indices that have to be optimized, i.e. minimized or maximized, in order to achieve efficiency.

This dissertation focuses on the applications that require robotic manipulation and, in particular, on robotic systems characterized by kinematic redundancy. The introduction of a kinematic redundancy gives the system an higher degree of dexterity and mobility that can be exploited to optimize the performance indices of interest. Redundancy can be structural, when the robot is designed to be redundant for generic tasks, or functional, when the robot is operated so as to be redundant. The concept applies to anthropomorphic serial chains, that are by far the most widespread robotic systems in manufacturing industries, but it is general enough to be extended to more complex systems. In fact, kinematic redundancy can be found in humanoids, mobile manipulators (terrestrial, aerial), parallel robots and systems of cooperating robots, that are more common in service and aerospace robotics. To date, in order to satisfy the increasing demand for flexible and more efficient robots, an increasing number of serial robot manufacturers offers products characterized by structural kinematic redundancy. The optimization problem that looks into the exploitation

of redundancy to optimize a given performance index is commonly referred to as *redundancy resolution*.

Many robotic tasks require to follow a specific path, such as in welding, cutting, gluing and in some assembly and disassembly tasks. This is opposed to *point-to-point* motion, that is rather typical in pick-and-place operations. Often, although following a path is not strictly required by the task itself, when the surrounding environment is cluttered by many obstacles, the most convenient approach is still to assign a specific path that avoids them. The definition of the speed with which the path is tracked is also a decision variable that, in most applications, is necessary to optimize. The criterion to follow, as mentioned, could be related to time minimization, but also tracking accuracy and, more generally, quality measures defined for the specific process. The optimization problem that looks into the definition of velocity along a given path is commonly referred to as *trajectory planning* or *time-parametrization*.

Both the optimization problems just stated can be addressed (and solved) at planning level, hence off-line, before the robot moves, or at control level, while the task is being executed and the system has to react to unplanned events. These two components are not, in general, alternative, but constitute two layers of a common architecture. If the communication is enabled between them, better performances can be achieved. The specific optimization technique to adopt at one stage or the other depends, among other things, on the available computational budget, intended as the time available to plan an operation, and constraints related to the computer architecture, in terms of CPU and memory. This aspect is crucial as it regulates how often the loop between planning and control can be closed. The shorter the loop period is, the more reactive the robot and the poorer the quality of the optimization (*local optimality*). On the other hand, if the loop period is larger, the robot is less reactive, but, in general, the objective function can be better optimized (*global optimality*).

Nowadays, many applications require the robots to live and oper-

ate in unstructured, unknown and highly-dynamical environments, which undoubtedly shifts the attention of research towards real-time planning and control. However, there still are situations, that are quite frequent in reality, where the process of off-line planning is not fully optimized in view of the strategical objectives we described. Indeed, the major employment of anthropomorphic arms in industries still concerns the execution of repetitive tasks in structured environments, where the task is planned once and executed cyclically. In the aerospace sector, off-line planning instances exist in the mission design phase for feasibility and budget assessments. Also, some sequences for in-orbit manipulation can be pre-planned in an optimal way. In missions that are characterized by windowed communications, a long planning time is usually available on ground to deliver more efficient commands to the spaceborne asset. On the contrary, when the domain requires highly-reactive behaviors, off-line planning can help measuring the performance of on-line algorithms so as to estimate the room available for improvement. The scenarios just described are the domain of this dissertation.

1.3 Objectives

Whether they are executed on-line or off-line, redundancy resolution and trajectory planning have been mostly treated as two independent, consecutive optimization processes [6, 7, 8], as depicted in Figure 1.1. A workspace path is given, that is mapped, through inverse kinematics, into the joint space, then, the joint-space path is time-parametrized to yield a trajectory that can be executed through motion control at joint level. The optimization performed at the former stage can be functional to the optimization performed at the latter. For instance, if the objective of trajectory planning is to define a minimum-time motion, redundancy resolution may optimize, according to an heuristic approach, the acceleration/deceleration capabilities of the manipulator along the path. Overall, the two processes “cooperate” to achieve a common

objective.

Unfortunately, in an attempt to reach the global optimum, the resolution of two independent globally-optimal problems does not guarantee the achievement of the overall global optimum. Indeed, if the objective is to minimize the path traversing time (or any other quality measure that depends on time), but the time along the path is only defined at the level of trajectory planning, there is no possibility for redundancy resolution to achieve such a goal.

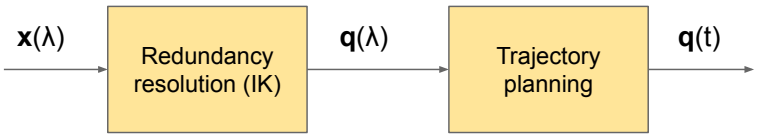


Figure 1.1: Redundancy resolution and trajectory planning as two independent processes; $\mathbf{x}(\lambda)$ is the workspace path, $\mathbf{q}(\lambda)$ the joint-space path and $\mathbf{q}(t)$ the joint-space trajectory

More recently, the two problems have been combined and solved together [9, 10, 11], as in Figure 1.2, although, because of the problem complexity and the underlying resolution technique, global optimality cannot be guaranteed. To provide a unified solution for both optimization problems is also the objective of this dissertation. Although several performance indices can be defined, the focus of this thesis is on time optimality.

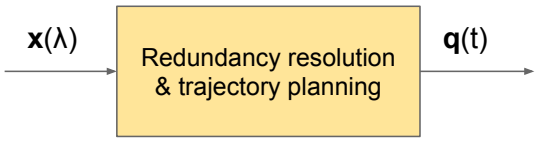


Figure 1.2: Redundancy resolution and trajectory planning as one unique process; $\mathbf{x}(\lambda)$ is the workspace path and $\mathbf{q}(t)$ the joint-space trajectory

In all the off-line applications that motivate this work, although the environment where the robot operates is structured, static,

known or partially known, not all the optimization techniques can capture the complexity of the real system. A typical approach to guarantee robustness and safety, prevent failures and maximize availability, is to downgrade the robot capacity and deliver conservative plans. This is necessary because the real system is often too complex to be modeled in mathematical terms, yet some optimization techniques show greater flexibility than others. This dissertation adopts dynamic programming as the main underlying methodology and central idea to cope with the complexity of reality. Also, because of its employment, in previous works, for both the optimization problems described above, it naturally arises as the unifying approach. Dynamic programming is extremely flexible in the accommodation of arbitrary constraints and objective functions [12, 13, 14] and, compatibly with the available planning time, can guarantee global optimality. When applied to real systems, it well describes the constraints at hand and no conservative hypotheses are necessary. Differently from many other optimization techniques, where more constraints make the search of the optimum cumbersome, in dynamic programming they are beneficial and allow reducing the computational complexity.

This thesis is structured in three main chapters, that address each of the blocks in Figure 1.1 and Figure 1.2 separately, in the following order: *redundancy resolution*, *trajectory planning* (time-optimal planning, in particular), *time-optimal planning for redundant robots*. All the chapters have a common structure. At first, the related literature is recalled, through an overview of the optimization strategies that can be adopted to solve the problem at hand. Then, the dynamic programming approach is presented. Similarities and differences in the application of dynamic programming for each domain are outlined. Last, each chapter is concluded with one or more experiments performed in simulation (for chapters 2 and 3) or in reality (for Chapter 4).

Each chapter contains some elements of novelty, briefly summarized here.

- Chapter 2 presents a topological approach to redundancy

resolution with dynamic programming. The objective is to exploit some topological features of the inverse kinematic mapping (i.e. *self-motion manifolds*, *homotopy classes* and *extended aspects*) to design a dynamic programming algorithm that is able to overcome the limits of the optimization techniques based on calculus of variations. The algorithm is based onto two additional innovations concerning redundancy parametrization and redundant manipulators topology. In particular, a minimum parametrization of redundancy has been derived that allows reducing the number of differential equations that describe the optimal motion of a redundant manipulator. On the topological side, homotopy classes of self-motion manifolds and joint-space paths have been analyzed and two different homotopy relations have been identified, here termed $\mathcal{C}$-*homotopy* and $\mathcal{C}$-*path-homotopy*.

- Chapter 3 presents a novel genetic algorithm to perform time-optimal planning along pre-scribed paths for non-redundant manipulators. The main advantage of this approach consists in the possibility of solving the singularities affecting time-optimal solutions. To this respect, a comparison with dynamic programming is provided.

- Chapter 4 presents a novel dynamic programming algorithm to address the problem of time-optimal planning of redundant robots. It combines the topological approach of Chapter 2 with the classical dynamic programming scheme for time-optimal planning of Chapter 3. The main advantage of the algorithm consists in the simplicity with which real-world constraints are imposed, that allows for a smooth execution on real hardware.

The thesis is concluded in Chapter 5 by summarizing the achievement of this work and suggesting some lines of development for future research in this area.

Chapter 2

Inverse kinematics of redundant robots

2.1 Introduction

When a task is assigned in a manipulator's workspace, we define the *inverse kinematics* as the problem of finding the joints positions allowing fulfillment of the task. If the task is more precisely characterized in terms of velocity and acceleration in the workspace, the inverse kinematics problem can be solved at differential level, in order to find the corresponding velocities and accelerations in the joint space.

A manipulator is defined as *kinematically redundant* when the number of degrees of freedom required by the task to execute is lower than the number of degrees of freedom provided by the manipulator's kinematic chain. In the simple case of a 3-DOF serial planar manipulator describing a circular trajectory, it is kinematically redundant if the end-effector orientation is arbitrary, while it is not if its orientation is constrained. Kinematic redundancy must not be confused with *actuation redundancy* and *measurement redundancy*, that are typical of closed kinematic chains. Examples of redundancy for a parallel manipulator are provided in

Figure 2.1. Actuation redundancy occurs when the end-effector is over-constrained by the actuators, while measurement redundancy occurs when the number of sensors is greater than the number of actuated joints [15]. The reader may recognize that, for the parallel structures of Figure 2.1, once the end-effector position and orientation are given, the kinematically redundant robot on the left admits infinite configurations for the upper legs. On the contrary, the redundantly actuated robot in the middle does not and all four actuators are constrained by the end-effector pose. Throughout this dissertation, for brevity, the terms *redundant* and *redundancy* will be used to implicitly refer to kinematic redundancy.

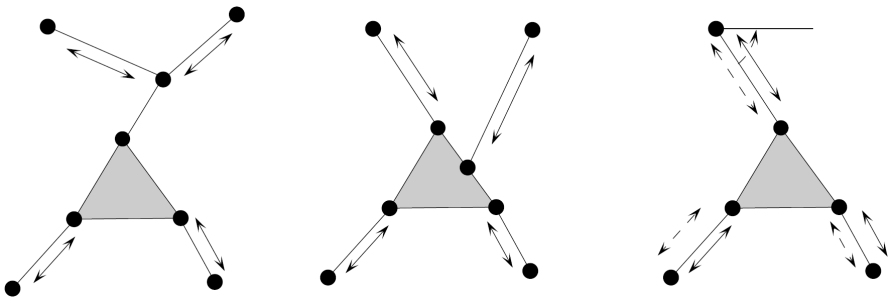


Figure 2.1: Kinematic redundancy (left), actuation redundancy (center) and measurement redundancy (right) for a planar parallel mechanism, where sensors are identified by dashed lines and arc [15]

If the manipulator is redundant, the inverse kinematics problem admits an infinite set of solutions almost everywhere in its workspace. Each solution represents a different kinematic configuration that can be reached at each point of the end-effector trajectory, while the task is achieved. In literature, sequences of such configurations are often referred to as *self-motions* or *internal motions*.

The augmented dexterity of redundant manipulators can be exploited to achieve several objectives, which are usually desirable for a multitude of real applications [16], such as:

- local or global optimization of performance indices;

- improvement of manipulability and singularity avoidance;
- obstacle avoidance and constrained motions.

In this thesis, we focus on redundancy resolution via global optimization of performance indices. In Section 2.2, the general mathematical formulation is given. Sections 2.3 and 2.4 address the problem of optimizing generic performance indices at each time instant t and globally, respectively. In Section 2.4, the reader may also find an overview of the most relevant performance indices based on the first-order Euler-Lagrange necessary conditions that, to date, have been used by several researchers in their works. Section 2.5 provides an alternative formulation for the solutions found throughout Section 2.4, which simplifies the set of the differential equations to be solved. In Section 2.6, several topological features of the inverse kinematics mapping are presented, on the basis of which, in Section 2.7, a dynamic programming algorithm for redundancy resolution is defined. The chapter is concluded with Section 2.8, where the designed algorithm is applied to a 7-DOF spatial manipulator in simulation.

2.2 Problem formulation

A *path* is assigned in a m-dimensional task space as $\mathbf{x}(\lambda)$, where $\lambda \in [0, \Lambda]$ is a parameter.

If λ is the time, a trajectory is assigned. Otherwise, the time law, i.e. the relation between λ and the time should be given to obtain a trajectory.

A redundant manipulator has n joints (with $n \geq m$) and is described by the direct kinematic functions $\mathbf{k}(\mathbf{q})$ and their derivatives. If $(\bullet)$ is a position or angular position variable, we define $(\bullet)' = \frac{d(\bullet)}{d\lambda}$ and call it λ-*velocity* or *parametric velocity*. Likewise, its derivative with respect to λ will be referred to as λ-*acceleration*

or *parametric acceleration*. Thus, said $\mathbf{J}(\mathbf{q}) = \frac{d\mathbf{k}}{d\mathbf{q}}$ the manipulator's Jacobian, we have:

$$\begin{aligned} \mathbf{x} &= \mathbf{k}(\mathbf{q}) && (2.1)\\ \mathbf{x}' &= \mathbf{J}(\mathbf{q})\mathbf{q}' && (2.2)\\ \mathbf{x}'' &= \mathbf{J}(\mathbf{q})\mathbf{q}'' + \mathbf{J}'(\mathbf{q},\mathbf{q}')\mathbf{q}' && (2.3) \end{aligned}$$

The problem is to find the solution $\mathbf{q}(\lambda)$ that keeps the end-effector on the path and globally minimizes a performance index G, function of joints positions, joints λ-velocity and parameter λ:

$$\begin{aligned} \min_{\mathbf{q}(\lambda)} \quad & \int_0^{\Lambda} G(\mathbf{q},\mathbf{q}',\lambda)d\lambda \\ \text{s.t.} \quad & \mathbf{x} = \mathbf{k}(\mathbf{q}) \end{aligned} \tag{2.4}$$

Notice that this is the usual formulation of the problem, but, in real applications, there actually are other constraints, such as those on joint limits, joint velocity limits, joint acceleration limits and torque limits, that might not always be treated by choosing a suitable performance index G.

2.3 Local solutions

Before analyzing the solutions to (2.4), it is worth considering the case in which the performance index G has to be locally optimized (i.e. at each trajectory point). Local solutions are those that, for every λ, solve the inverse kinematics at a differential level.

2.3.1 First-order

In order to minimize a generic performance index, the problem can be posed as

$$\begin{aligned} \min_{\mathbf{q}'\in\Re^n} \quad & \frac{1}{2}(\mathbf{q}' - \boldsymbol{\phi}')^T(\mathbf{q}' - \boldsymbol{\phi}') \\ \text{s.t.} \quad & \mathbf{x}' = \mathbf{J}(\mathbf{q})\mathbf{q}' \end{aligned} \tag{2.5}$$

that is for each λ to find the closest solution $\mathbf{q}'$ to the vector $\boldsymbol{\phi}'$ that still lets the manipulator follow the path (it is assumed that $\mathbf{x}' \in \mathcal{R}(\mathbf{J})$, range space of $\mathbf{J}$). It is easy to show, by using the Lagrangian multipliers technique, that, outside of singularities, the solution is [17, 18]

$$\begin{aligned} \mathbf{q}'(\lambda) &= \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{x}'(\lambda) + \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right]\boldsymbol{\phi}' \\ \mathbf{q}(\lambda) &= \int_0^{\lambda} \mathbf{q}'(\tau)d\tau + \mathbf{q}_0 \end{aligned} \tag{2.6}$$

First term in the r.h.s. of (2.6) are the minimum-norm joint velocities that are necessary to follow the path [19]. The second term are the joint velocities projected in the null-space of the Jacobian leading to internal motion. In (2.6), $\boldsymbol{\phi}'$ can be chosen as $\boldsymbol{\phi}' = -k\frac{\partial G}{\partial \mathbf{q}}$ in order to minimize the performance index G.

With this choice, the solution (2.6) *tends* to minimize the performance index without reaching local minima, as $\frac{\partial G}{\partial \mathbf{q}}$ is not necessarily zeroed at each λ. In other words, the instantaneous resolution at velocity level only guarantees that the motion is locally improved by incremental movement from the current arm state [20]. Moreover, the choice of the parameter k can lead to different (and sometimes unacceptable) solutions.

It is also important to notice that, in order to fully define solution (2.6), the initial conditions have to be given. Different initial conditions lead to different solutions, providing different results for the performance index.

In addition, the solution is not conservative (or *cyclic*) in the whole space [21], i.e. a closed task-space path does not necessarily lead to a closed joint-space path.

Lastly, at the end of the path, internal motion could still be performed to reach the minimum possible value of G.

In a few cases, it is possible to find a simpler closed-form solution to optimize certain performance indices. For instance, if the

performance index to be minimized is

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{W}\mathbf{q}' \tag{2.7}$$

it is well know that the best local solution is

$$\mathbf{q}'(\lambda) = \mathbf{J}_W^\dagger(\mathbf{q})\mathbf{x}'(\lambda) \tag{2.8}$$

where $\mathbf{J}_W^\dagger(\mathbf{q}) = \mathbf{W}^{-1}\mathbf{J}^T(\mathbf{q})\left[\mathbf{J}(\mathbf{q})\mathbf{W}^{-1}\mathbf{J}^T(\mathbf{q})\right]^{-1}$ is the weighted pseudoinverse of the Jacobian for non-singular $\mathbf{J}(\mathbf{q})\mathbf{W}^{-1}\mathbf{J}^T(\mathbf{q})$.

A different choice could be that of choosing the null-space joint velocities in (2.6) as those that minimize the performance index at each time instant, that is reaching $\frac{\partial G}{\partial \mathbf{q}} = 0$ at each λ, if the choice $\boldsymbol{\phi}' = -k\frac{\partial G}{\partial \mathbf{q}}$ is made. This gives the local minimum at each λ but could lead to discontinuities in the velocities. In this case, the minimization of the performance index can also be performed at $\lambda = 0$ (if not all of the initial joint variables are given). Notice also that, even though a point-wise optimal solution is obtained, the resulting velocity could be unacceptably large if the manipulator is close to a singular configuration.

It is also worth noticing that, regardless of the form of the solution, closed-loop inverse kinematics schemes [22] have to be necessarily adopted to reduce the integration error when the solution is implemented in discrete time.

2.3.2 Second-order

Likewise, it is possible to demonstrate that the second-order inverse kinematics solution, locally optimizing a generic performance index, is

$$\begin{aligned}
\mathbf{q}''(\lambda) &= \mathbf{J}^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})\boldsymbol{\phi}'' \\
\mathbf{q}'(\lambda) &= \int_0^\lambda \mathbf{q}''(\tau)d\tau + \mathbf{q}'_0 \\
\mathbf{q}(\lambda) &= \int_0^\lambda \mathbf{q}'(\tau)d\tau + \mathbf{q}_0
\end{aligned} \tag{2.9}$$

where, for the sake of clarity, the dependencies on λ and $\mathbf{q}$ have been omitted at the right-hand side of equation (2.9).

Similarly to the first-order case, the first term in the r.h.s. of (2.9) represents the minimum-norm joints λ-accelerations, while the second term contains the joints λ-accelerations projected onto the null-space of the Jacobian.

It is important to note, also for the discussions that will follow, that the second-order redundancy resolution scheme may lead to anomalous behaviors, described in [20] and [23]. Such behaviors consist in large torques, accelerations and velocities, that have also been observed for a variety of algorithms, including those obtained by changing the value of $\boldsymbol{\phi}''$ or by using the inertia-weighted pseudoinverse $\mathbf{J}_H^\dagger$ in place of $\mathbf{J}^\dagger$. Indeed, because the local solutions do not have visibility over the whole trajectory, they may lead the manipulator toward configurations where a partial loss of redundancy may happen (i.e. alignment of links), resulting in the aforementioned issues.

An alternative solution to (2.9) can be found by differentiating (2.6). In this case, one would obtain

$$\mathbf{q}''(\lambda) = \mathbf{J}^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})(\mathbf{J}^{\dagger\prime}\mathbf{x}' - \mathbf{J}^{\dagger\prime}\mathbf{J}\boldsymbol{\phi}' + \boldsymbol{\phi}'') \quad (2.10)$$

which demonstrates that even though minimum-norm velocities were chosen (i.e. $\boldsymbol{\phi}' = 0$ and $\boldsymbol{\phi}'' = 0$), minimum-norm accelerations would not be necessarily obtained [24].

2.4 Global solution

Unlike local optimization techniques, global ones aim at minimizing a certain performance index over the whole trajectory, providing a solution to problem (2.4). Global solutions might not (and usually do not) achieve local optimization, i.e. values of λ may exist where the globally-minimized cost function is greater than

the locally-minimized cost function, but they guarantee that the performance index is optimized in a global sense.

Some distinctive features of global solutions are:

- they have to be necessarily computed off-line. It will be shown in this section that global optimization problems often result in Boundary Value Problems (BVP) which are time consuming and sometimes hard to solve, even with modern technologies. However, while the heavy computation burden required for global methods rules out real-time control, for common industrial problems requiring repetitive motion, the global technique may be perfectly suited for finding a solution which will be used over and over again. Likewise, in all the cases where pre-planning is possible, such as in a multitude of space applications, the optimum trajectory could be computed off-line and sent to the spaceborne asset to be executed.
- initial conditions could be not given but computed as a part of the solution;
- it will be shown that only necessary conditions can be given for a solution to be optimal. This means that global optimization techniques may still end up in local minima/-maxima: there could be multiple trajectories satisfying the necessary conditions and even multiple optimal trajectories (i.e. the true minimum is not necessarily unique); also, these trajectories do not necessarily belong to the same class of homotopy [25, 26];
- global solutions obtained from the resolution of the BVP could be anomalous, meaning that, because of the vicinity to singularities, the quantities in play could uncontrollably grow, resulting in unfeasible profiles;
- global solutions are not necessarily cyclic per se (i.e. same initial and final joints configurations), if not explicitly imposed as a constraint of the problem.

Since the problem (2.4) is the optimization of an integral form cost function subject to some constraints, calculus of variations can be used to obtain the necessary conditions from which the closed-form *optimal* solution can be derived. In particular, it has been demonstrated that both the Euler-Lagrange equations and the Pontryagin's maximum principle allow to obtain such a solution and ultimately provide coherent results.

While the Pontryagin's maximum principle is more suitable for those cases where the constraints are provided in a differential form, the Euler-Lagrange method can be directly applied to direct kinematics, immediately solving problem (2.4). In order to ease the connection between the present dissertation and previous works, the majority of which uses the Euler-Lagrange conditions, the same method is chosen here as the reference. However, for the sake of completeness, mentions of the Pontryagin's maximum principle are provided in Section 2.4.3.

2.4.1 Euler-Lagrange necessary conditions

The Lagrangian function is defined as:

$$L = G(\mathbf{q}, \mathbf{q}', \lambda) + \boldsymbol{\mu}^T[\mathbf{x} - \mathbf{k}(\mathbf{q})] \tag{2.11}$$

where G is the performance index of (2.4) and $\boldsymbol{\mu}$ is the vector of Lagrange multipliers. It can be demonstrated that the first-order necessary conditions for a minimum or a maximum of the integral performance index are

$$\begin{aligned} &\frac{\partial L}{\partial \mathbf{q}} - \frac{d}{d\lambda}\left(\frac{\partial L}{\partial \mathbf{q}'}\right) = 0 \\ &\mathbf{x}(\lambda) = \mathbf{k}(\mathbf{q}(\lambda)) \end{aligned} \tag{2.12}$$

which are n second-order differential equations (equivalent to $2n$ first-order differential equations), known as Euler's equations or Euler-Lagrange's equations, and m algebraic equations. Together

with them, the following $2n$ two-point boundary conditions have to be satisfied:

$$\left(\frac{\partial L}{\partial \mathbf{q}'}\right)^T \cdot \delta\mathbf{q}(\lambda_B) = 0 \quad \text{for } \lambda_B = 0, \Lambda \tag{2.13}$$

Since algebraic equations and its first derivatives can be used in principle to simplify the differential equations, $2(n-m)$ differential equations remain to be solved. In Section 2.5.2, a method is proposed to reduce the system of equations, either by choosing one of the $\mathbf{q}$ variables, or by suitably selecting a combination of them. The limitations of such a method are also discussed.

The differential equation in (2.12) can also be written as

$$\frac{d}{d\lambda}\mathbf{G}_{q'} - \mathbf{G}_q + \mathbf{J}^T\boldsymbol{\mu} = 0 \tag{2.14}$$

and its boundary condition as

$$\mathbf{G}_{q'}^T \cdot \delta\mathbf{q}(\lambda_B) = 0 \quad \text{for } \lambda_B = 0, \Lambda \tag{2.15}$$

where $\mathbf{G}_{q'} = \frac{\partial G}{\partial \mathbf{q}'}$ and $\mathbf{G}_q = \frac{\partial G}{\partial \mathbf{q}}$.

Assuming that we wanted to minimize the cost function, the necessary condition (Legendre's condition) for the solution to be a minimum is that, along the solution,

$$\frac{\partial^2 L}{\partial \mathbf{q}'^2} \geq 0 \tag{2.16}$$

As remarked at the beginning of Section 2.4, these are only necessary conditions, which means that solutions could be found which do not correspond to the actual global minimum. As a matter of fact, in [25], it has been observed that each solution satisfying the Euler-Lagrange equations will be optimal within its homotopy class, but not necessarily optimal in a global sense. In other words, the Euler-Lagrange equations are not able to distinguish optimal solutions in differing homotopy classes. Since more than

one homotopy relation can be defined between joint-space paths, we detail this matter further in Section 2.6.2.

If a cost function $G(\mathbf{q}, \mathbf{q}', \mathbf{q}'', \lambda)$ is given which is dependent on the second-order derivative of the joints positions, the second-order necessary conditions have to be considered instead:

$$\frac{\partial L}{\partial \mathbf{q}} - \frac{d}{d\lambda}\left(\frac{\partial L}{\partial \mathbf{q}'}\right) + \frac{d^2}{d\lambda^2}\left(\frac{\partial L}{\partial \mathbf{q}''}\right) = 0 \tag{2.17}$$

This formulation allows to accommodate a multitude of cost functions defined on acceleration-dependent terms. For instance, in [23], the square-norm of torques is minimized.

2.4.2 Euler-Lagrange boundary conditions

For the Euler-Lagrange necessary conditions just described, let us analyze the boundary conditions. Several cases are possible:

- both the initial and final joint configurations are free;
- only the initial joint configurations are assigned, while the final ones are free;
- both the initial and final joint configurations are assigned;
- initial and final joint configurations are not assigned, but they are required to be the same;
- initial joint positions and velocities are assigned, while the final ones are free.

2.4.2.1 Initial and final configurations not assigned

If the initial and final configurations are not assigned, the "natural" boundary conditions (2.15) apply. Recalling (2.6), one may write:

$$\delta\mathbf{q}(\lambda) = \mathbf{J}^{\dagger}(\mathbf{q})\delta\mathbf{x}(\lambda) + \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right]\boldsymbol{\phi}' \tag{2.18}$$

At the two extremal points $\lambda = 0$ and $\lambda = \Lambda$, $\mathbf{x}$ is given, implying $\delta\mathbf{x} = 0$ and $\delta\mathbf{q}$ on the left-hand side of the equation corresponds to internal motion. This implies $\mathbf{G}_{q'}^T(\mathbf{I}-\mathbf{J}^\dagger\mathbf{J})\boldsymbol{\phi}' = \mathbf{0}$ (the dependency on $\mathbf{q}$ and $\mathbf{q}'$ has been omitted to simplify notation), but, since it has to hold for each $\boldsymbol{\phi}'$, the boundary conditions become

$$(\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})\mathbf{G}_{q'} = 0 \quad \Rightarrow \quad \mathbf{G}_{q'} \in \mathcal{R}(\mathbf{J}^T) \tag{2.19}$$

where $\mathcal{R}(\mathbf{J}^T)$ is the range space of $\mathbf{J}^T$, so that, being $\mathcal{N}(\mathbf{J})$ the null space of $\mathbf{J}$, $\mathcal{R}(\mathbf{J}^T) = \mathcal{N}^\perp(\mathbf{J})$ holds. Although (2.19) are $2n$ conditions, without loss of generality, one may replace the null-space projector $(\mathbf{I} - \mathbf{J}^\dagger\mathbf{J})$ with the transpose of a basis $\mathbf{N} \in \Re^{n\times r}$ of the Jacobian's null space to reduce the number of conditions to $2(n - m) = 2r$ [25]:

$$\mathbf{N}^T(\mathbf{q}(\lambda_B))\mathbf{G}_{q'}(\lambda_B) = \mathbf{0} \quad \text{for } \lambda_B = 0, \Lambda \tag{2.20}$$

For typical performance indices, as shown in Section 2.4.5, such boundary conditions constrain the value of $\mathbf{q}'$ at the boundaries, while the actual joints initial and final configurations $\mathbf{q}(\lambda_B)$ will be an output of the problem. Being only half of the variables, i.e. $\mathbf{q}'$, constrained at the initial time and the same constrained at the final time, a Two-Points Boundary Value Problem has to be solved in this case.

Natural boundary conditions guarantee that the value of the performance index is lower (in case of minimization) or higher (in case of maximization) than any other value obtained with assigned boundary conditions.

2.4.2.2 Initial configuration assigned

In this case, at the initial point, $\delta\mathbf{q} = 0$ and there are no natural boundary conditions. For the resolution of the differential equations, n values shall be provided as input to the problem for the initial joint positions. However, since the initial configuration shall obviously respect the task constraint, $n - m$ conditions

are sufficient, while the remaining m are computed from inverse kinematics. At the final point, the $n-m$ natural conditions apply.

In this case, n variables, i.e. $\mathbf{q}$, are assigned at the initial time, while the other n, i.e. $\mathbf{q}'$, are constrained at the final time, which means that, again, a Two-Point Boundary Value Problem has to be solved.

2.4.2.3 Initial and final configurations assigned

At both the initial and final point, $\delta\mathbf{q} = 0$ and there are no natural boundary conditions. This is again a Two-Point Boundary Value Problem, as only n variables, i.e. $\mathbf{q}$, are assigned at the beginning and at the end of the trajectory.

For some performance indices, the initial and final configurations could be chosen according to some local optimization criterion which is selected coherently with the global optimization criterion. For example, if the objective was to minimize the variation of the manipulability index

$$I = \int_0^{\Lambda} \left[\frac{d}{d\lambda} (\det(\mathbf{J}\mathbf{J}^T)) \right]^2 d\lambda \tag{2.21}$$

the boundary configurations could be chosen so that the manipulability index is as large as possible. This requires to solve the following problem at the boundaries [27]:

$$\max_{\mathbf{q} \in \Re^n} \det(\mathbf{J}\mathbf{J}^T) \tag{2.22}$$

In other cases, the initial and final configurations could be "manually" fixed on the basis of constraints of the workspace, which is a common possibility in real applications. In particular, if the initial and final configurations are chosen as equal, which is possible if the trajectory is closed in the task space, it will be closed in the joint space too. This choice made, Choi et al. [28] pointed out that, for the cases where the redundancy degree is one, i.e. $n - m = 1$, the problem could be transformed into an Initial Value Problem

(IVP), which is trivial to solve compared to BVPs. The idea behind this resolution strategy is that, once the initial position is assigned, the initial velocity can be sought that ensures conservativity in the joint space, i.e. $\mathbf{q}(\Lambda) = \mathbf{q}(0)$. Once the initial joint velocity has been parametrized with respect to a scalar parameter, the objective is achieved by iterating on such a parameter, until the cost function

$$K = \frac{1}{2}\|\mathbf{q}(\Lambda) - \mathbf{q}(0)\|^2 \tag{2.23}$$

gets below a certain pre-assigned tolerance.

2.4.2.4 Equal initial and final configurations

In this case the configurations are not assigned, but it is only required that they are the same. If they are the same, their variations must also be the same. This leads to the additional condition that the initial and final joint velocities must be the same [29]. This means that the optimization problem (2.12) can be solved with the addition of the following two constraints (*conservativity requirements*):

$$\begin{aligned} \mathbf{q}(0) &= \mathbf{q}(\Lambda) \\ \mathbf{q}'(0) &= \mathbf{q}'(\Lambda) \end{aligned} \tag{2.24}$$

Even in this case, we are encountering a highly non-linear boundary value problem, however, the periodicity of the solution could be exploited to develop an approximate solution. Wang and Kazerounian [29] demonstrated that it is possible to approximate the redundant joint coordinates with Fourier series and, from these, to seek the solution through an iterative optimization algorithm. Their results are extremely close to the optimal solution, while the execution time is tens to hundreds times less than that needed to solve the boundary value problem.

2.4.2.5 Assigned initial configuration and velocity

In this case, all the boundary conditions are at the initial point. The problem can then be solved as an Initial Value Problem. But, while at the initial point $\delta\mathbf{q} = 0$ and there are no natural boundary conditions, this is not true at the final point. The solution will violate the natural boundary conditions at the final point that is a necessary condition for the minimum, ending up in a *weak* minimum. The *strong* minimum could be achieved with a sudden jump to the velocity provided by the imposition of the natural condition, but this would imply a discontinuity in the parametric velocities between the assigned value at $\lambda = 0$ and equation (2.14) at $\lambda > 0$. Since such a discontinuity is physically impossible, the best strategy would be to "tend" to the *natural* velocity as soon as possible and then continue following (2.14) [24].

2.4.3 Pontryagin's maximum principle

The Pontryagin's maximum principle is applicable to all the cases where an integral cost function has to be optimized, subject to constraints made of differential equations. The first application of such a method to redundant manipulators appears in [30], where problem (2.4) is actually rewritten by using the first-order inverse kinematics in place of the positions kinematics (with $\mathbf{u} = \boldsymbol{\phi}'$):

$$\begin{aligned} &\min_{\mathbf{q}(\lambda)} \int_0^{\Lambda} G(\mathbf{q}, \mathbf{q}', \lambda) d\lambda \\ &\text{s.t. } \mathbf{q}' = \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{x}' + \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right] \mathbf{u} \end{aligned} \tag{2.25}$$

It is worth noting that in (2.25), as in [30], the first-order inverse kinematics is written in its most general form, with the inclusion of the null-space joints λ-velocities, as done in (2.6). In (2.25), $\mathbf{u} \in \Re^n$ can be seen as a system input that is able to affect the evolution of the state $\mathbf{q}$, when $\mathbf{x}'$ is given.

Now define the functions $\mathbf{g}$ as

$$\mathbf{g}(\mathbf{q}, \mathbf{u}, \lambda) = \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{x}' + \left[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\right]\mathbf{u} \tag{2.26}$$

Thus the problem to be solved, more generally, becomes

$$\begin{aligned} &\min_{\mathbf{q}(\lambda)} \int_0^{\Lambda} G(\mathbf{q}, \mathbf{q}', \lambda) d\lambda \\ &\text{s.t. } \mathbf{q}' = \mathbf{g}(\mathbf{q}, \mathbf{u}, \lambda) \end{aligned} \tag{2.27}$$

The following Hamiltonian is introduced, with $\boldsymbol{\eta} \in \Re^n$ called *co-state* vector.

$$H(\mathbf{q}, \mathbf{q}', \boldsymbol{\eta}, \mathbf{u}, \lambda) = -G(\mathbf{q}, \mathbf{q}', \lambda) + \boldsymbol{\eta}^T \mathbf{g}(\mathbf{q}, \mathbf{u}, \lambda) \tag{2.28}$$

The Pontryagin's maximum principle asserts that, if $\mathbf{u}^*(\lambda)$ is the optimal control sequence, i.e. the one maximizing the Hamiltonian, the optimal state sequence, i.e. joint-space trajectory, can be obtained by solving the following differential equations:

$$\boldsymbol{\eta}'(\lambda) = -\left(\frac{\partial H}{\partial \mathbf{q}}\right)^T \tag{2.29}$$

$$\mathbf{q}'(\lambda) = \left(\frac{\partial H}{\partial \boldsymbol{\eta}}\right)^T \tag{2.30}$$

The reader may recognize that (2.30) is equivalent to the constraint in (2.25).

Both sets of differential equations will depend on $\mathbf{g}$ and thus on $\mathbf{u}$. According to the principle just enunciated, the optimal control sequence has to be selected such that

$$\mathbf{u}^*(\lambda) = \max_{\mathbf{u}(\lambda)} H(\mathbf{q}, \mathbf{q}', \boldsymbol{\eta}, \mathbf{u}, \lambda) \tag{2.31}$$

In the case $\mathbf{u} \in \Re^n$, (2.31) directly translates into

$$\frac{\partial H(\mathbf{q}, \mathbf{q}', \boldsymbol{\eta}, \mathbf{u}, \lambda)}{\partial \mathbf{u}} = 0 \tag{2.32}$$

Otherwise, if $\mathbf{u} \in \mathcal{U}$, limited subset of $\Re^n$, maximums could exist on the boundary of $\mathcal{U}$, not satisfying equation (2.32).

Together with the $2n$ equations (2.29) and (2.30), the following boundary conditions have to be satisfied:

$$\boldsymbol{\eta}^T(0) \cdot \delta\mathbf{q}(0) = 0 \quad (2.33)$$

$$\boldsymbol{\eta}^T(\Lambda) \cdot \delta\mathbf{q}(\Lambda) = 0 \quad (2.34)$$

$$\mathbf{x}(0) - \mathbf{k}(\mathbf{q}(0)) = 0 \quad (2.35)$$

Although it may seem that the number of boundary conditions is $2n + m$, in the following section it will be shown that m of the conditions above are linearly dependent on others. Thus only $2n$ of the $2n + m$ constraints above are needed to solve equations (2.29) and (2.30). It is also worth remarking that the imposition of conditions (2.35) is necessary to make sure that the kinematics constraints are respected. As the integration is performed by using equations (2.30), which are equivalent to first-order differential kinematics in (2.25), imposing the kinematic constraints at $\lambda = 0$ ensures they are respected at each λ.

If the problem is formulated in such a way that boundary conditions on joints λ-velocities are given, equations (2.33) and (2.34) can be inserted in the constraints equations (2.30) to obtain alternative boundary conditions in joint λ-velocity form [31].

2.4.4 Pontryagin's boundary conditions

For the Pontryagin's necessary conditions just described, let us analyze the boundary conditions. The same cases as the Euler-Lagrange conditions are considered.

2.4.4.1 Initial and final configurations not assigned

If both $\mathbf{q}(0)$ and $\mathbf{q}(\Lambda)$ are free, the "natural" boundary conditions apply. As far as the final state is concerned, we can say that

$\delta\mathbf{q}(\Lambda) \neq 0$ as $\mathbf{q}(\Lambda)$ is completely free. The same cannot be said for $\mathbf{q}(0)$, as it is partially constrained by (2.35). By making $\delta\mathbf{q}(0)$ explicit, as in (2.18), boundary conditions (2.33) become

$$\Big[\mathbf{I} - \mathbf{J}^{\dagger}\big(\mathbf{q}(0)\big)\mathbf{J}\big(\mathbf{q}(0)\big)\Big]\boldsymbol{\eta}(0) = \mathbf{0} \tag{2.36}$$

In the case of free initial and final configurations, the boundary conditions (2.33)-(2.35) become

$$\Big[\mathbf{I} - \mathbf{J}^{\dagger}\big(\mathbf{q}(0)\big)\mathbf{J}\big(\mathbf{q}(0)\big)\Big]\boldsymbol{\eta}(0) = \mathbf{0} \tag{2.37}$$

$$\boldsymbol{\eta}(\Lambda) = \mathbf{0} \tag{2.38}$$

$$\mathbf{x}(0) - \mathbf{k}(\mathbf{q}(0)) = \mathbf{0} \tag{2.39}$$

As observed before in regards to the Euler-Lagrange conditions, only $n - m$ of the equations (2.37) are linearly independent. In facts, without loss of generality, one may replace the null-space projector with a null-space basis, leading to $n - m$ conditions.

Of the conditions above, n are given at the initial stage, n are given at the final stage, thus a Two-Point Boundary Value Problem has to be solved.

2.4.4.2 Initial configuration assigned

In this case, at the initial stage, $\delta\mathbf{q} = 0$ and there are no natural boundary conditions. For the resolution of the differential equations, n values shall be provided as input to the problem for the initial joint positions. However, since the initial configuration shall obviously respect the task constraint, $n - m$ conditions are sufficient, while the remaining m are computed from inverse kinematics. At the final point, the $n-m$ natural conditions apply.

Therefore, assuming that the initial configuration does not violate the kinematic constraints, the boundary conditions are

$$\mathbf{q}(0) = \mathbf{q}_i \tag{2.40}$$

$$\boldsymbol{\eta}(\Lambda) = \mathbf{0} \tag{2.41}$$

As n of the conditions are given at the initial stage and n are given at the final stage, a Two Point Boundary Value Problem has to be solved again.

2.4.4.3 Initial and final configurations assigned

At both the initial and final point, $\delta\mathbf{q} = 0$ and there are no natural boundary conditions. Like for the Euler-Lagrange method, this is again a Two-Point Boundary Value Problem, as only half of the variables, i.e. $\mathbf{q}$, are assigned at the beginning and at the end of the trajectory:

$$\mathbf{q}(0) = \mathbf{q}_i \tag{2.42}$$
$$\mathbf{q}(\Lambda) = \mathbf{q}_f \tag{2.43}$$

2.4.4.4 Equal initial and final configurations

In case the *conservativity requirement* is imposed, i.e. $\mathbf{x}(0) = \mathbf{x}(\Lambda) \rightarrow \mathbf{q}(0) = \mathbf{q}(\Lambda)$, the boundary conditions to be considered are [31]:

$$\mathbf{q}(0) - \mathbf{q}(\Lambda) = \mathbf{0} \tag{2.44}$$
$$\Big[\mathbf{I} - \mathbf{J}^{\dagger}\big(\mathbf{q}(0)\big)\mathbf{J}\big(\mathbf{q}(0)\big)\Big]\big(\boldsymbol{\eta}(0) - \boldsymbol{\eta}(\Lambda)\big) = \mathbf{0} \tag{2.45}$$
$$\mathbf{x}(0) - \mathbf{k}(\mathbf{q}(0)) = \mathbf{0} \tag{2.46}$$

The same considerations as Section 2.4.4.1 about the replacement of the null-space projector with a null-space basis hold here as well, resulting in $2n$ independent boundary conditions.

2.4.5 Typical performance indices

The performance indices reported in this section are computed starting from the Euler-Lagrange necessary conditions (2.12). The cost function is assumed to be dependent on joints λ-velocities at

most, not on higher order derivatives. For an example of second-order necessary conditions, the reader may refer to [31], where the square norm of torques is globally minimized. In [32], second-order necessary conditions are employed to minimize the deflection of a flexible base that supports a redundant manipulator. This example is of particular interest as the Euler-Lagrange equations are derived in a context where a coupled dynamic system made of the manipulator and its base is considered.

Although the Euler-Lagrange conditions are used throughout this section, it is possible to demonstrate that the application of the Pontryagin's maximum principle leads to equivalent results [31].

2.4.5.1 Least-square velocities

If the performance index to minimize is the square norm of λ-velocities

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{q}' \tag{2.47}$$

we have $\mathbf{G}_{q'} = \mathbf{q}'$ and $\mathbf{G}_q = \mathbf{0}$. The differential equations (2.14) become

$$\mathbf{q}'' + \mathbf{J}^T\boldsymbol{\mu} = \mathbf{0} \tag{2.48}$$

The solution must also satisfy the algebraic equation in (2.12) and its derivatives. Solving (2.48) for $\mathbf{q}''$ and putting it in the second-order differential kinematics $\mathbf{J}\mathbf{q}'' = \mathbf{x}'' - \mathbf{J}'\mathbf{q}'$, leads to

$$-\mathbf{J}\mathbf{J}^T\boldsymbol{\mu} = \mathbf{x}' - \mathbf{J}'\mathbf{q}' \tag{2.49}$$

that solved for μ and put in (2.48) leads to the solution

$$\mathbf{q}'' = \mathbf{J}^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') \tag{2.50}$$

That is, the global least-square joint λ-velocities are obtained by using the local least-square joint λ-accelerations. The natural boundary conditions are

$$\mathbf{q}' = \mathbf{J}^T\mathbf{p} \tag{2.51}$$

where $\mathbf{p}$ is a generic vector at the two extremal points that, since also the equation $\mathbf{x}' = \mathbf{J}\mathbf{q}'$ holds, can be rewritten as $\mathbf{x}' = \mathbf{J}\mathbf{J}^T\mathbf{p}$. Solving for $\mathbf{p}$ and substituting leads to the two boundary conditions

$$\mathbf{q}'(\lambda_B) = \mathbf{J}^\dagger(\mathbf{q}(\lambda_B))\mathbf{x}'(\lambda_B) \tag{2.52}$$

or, equivalently,

$$\left[\mathbf{I} - \mathbf{J}^\dagger(\mathbf{q}(\lambda_\mathbf{B}))\mathbf{J}(\mathbf{q}(\lambda_\mathbf{B}))\right]\mathbf{q}'(\lambda_B) = 0 \tag{2.53}$$

2.4.5.2 Constant-weighted least-square velocities

In this case the performance index is [24]

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{W}\mathbf{q}' \tag{2.54}$$

which gives $\mathbf{G}_{q'} = \mathbf{W}\mathbf{q}'$ and $\mathbf{G}_q = \mathbf{0}$. The differential equations (2.14) become

$$\mathbf{W}\mathbf{q}'' + \mathbf{J}^T\boldsymbol{\mu} = 0 \tag{2.55}$$

With the same reasoning, the solution must satisfy

$$\mathbf{q}'' = \mathbf{W}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)^{-1}(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') = \mathbf{J}_W^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') \tag{2.56}$$

with the two natural boundary conditions

$$\mathbf{q}'(\lambda_B) = \mathbf{J}_W^\dagger(\mathbf{q}(\lambda_B))\mathbf{x}'(\lambda_B) \tag{2.57}$$

2.4.5.3 Variable-weighted least-square velocities

The performance index above can be easily generalized to the case where the weights matrix $\mathbf{W}$ is configuration dependent and/or, even more generally, time dependent [24]:

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{W}(\mathbf{q}, \lambda)\mathbf{q}' \tag{2.58}$$

which gives $\mathbf{G}_{q'} = \mathbf{W}\mathbf{q}'$ and $\mathbf{G}_q = \frac{1}{2}\mathbf{q}'^T\frac{\partial \mathbf{W}}{\partial \mathbf{q}}\mathbf{q}'$. The differential equations (2.14) become

$$\mathbf{W}\mathbf{q}'' + \mathbf{W}'\mathbf{q}' + \mathbf{J}^T\boldsymbol{\mu} - \frac{1}{2}\mathbf{q}'^T\frac{\partial \mathbf{W}}{\partial \mathbf{q}}\mathbf{q}' = \mathbf{0} \tag{2.59}$$

With the same reasoning, the solution must satisfy

$$\mathbf{q}'' = \mathbf{J}_W^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}_W^\dagger\mathbf{J})\mathbf{W}^{-1}\left(-\mathbf{W}' + \frac{1}{2}\mathbf{q}'^T\frac{\partial \mathbf{W}}{\partial \mathbf{q}}\right)\mathbf{q}' \tag{2.60}$$

with the two boundary conditions

$$\mathbf{q}'(\lambda_B) = \mathbf{J}_W^\dagger(\mathbf{q}(\lambda_B))\mathbf{x}'(\lambda_B) \tag{2.61}$$

Kinetic energy A particular case of the above is when the matrix $\mathbf{W}$ is equal to the inertia matrix $\mathbf{H}$ [24]. In this case it is

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{H}(\mathbf{q})\mathbf{q}' \tag{2.62}$$

which gives $\mathbf{G}_{q'} = \mathbf{H}\mathbf{q}'$ and $\mathbf{G}_q = \frac{1}{2}\mathbf{q}'^T\frac{\partial \mathbf{H}}{\partial \mathbf{q}}\mathbf{q}'$.

With the same reasoning, the solution must satisfy

$$\mathbf{q}'' = \mathbf{J}_H^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}_H^\dagger\mathbf{J})\mathbf{H}^{-1}\mathbf{C} \tag{2.63}$$

where the relation $\mathbf{C} = \left(-\mathbf{H}' + \frac{1}{2}\mathbf{q}'^T\frac{\partial \mathbf{H}}{\partial \mathbf{q}}\right)\mathbf{q}'$ has been used to highlight that the last term in (2.60) corresponds to the Coriolis terms linked to the manipulator's torques.

The two natural boundary conditions are

$$\mathbf{q}'(\lambda_B) = \mathbf{J}_H^\dagger(\mathbf{q}(\lambda_B))\mathbf{x}'(\lambda_B) \tag{2.64}$$

Variation of the weighted manipulability index Another perfomance index that can be attributed to the variable-weighted

least-square velocities case is the variation of the weighted manipulability index, that has been introduced in Section 2.4.2.3, concerning boundary conditions. In this case we have:

$$G = \left[\frac{d}{d\lambda}(\det(\mathbf{J}\mathbf{W}\mathbf{J}^T))\right]^2 \tag{2.65}$$

where $w(\lambda) = \det(\mathbf{J}\mathbf{W}\mathbf{J}^T)$ is defined as the weighted manipulability index and represents the manipulating ability of the robot or the distance from singularities.

It can be demonstrated [27] that it is possible to find a matrix U computed from J and W such that

$$G = \mathbf{q}'^T\mathbf{U}\mathbf{q}' \tag{2.66}$$

which resembles the cost function (2.58). The solution to minimizing (2.65), as well as its boundary conditions, can be easily found from equations (2.60) and (2.61) respectively.

2.4.5.4 Stage-dependent weighted least-square velocities + configuration dependent term

In this case the cost function is [25]

$$G = \frac{1}{2}\mathbf{q}'^T\mathbf{W}(\lambda)\mathbf{q}' + g(\mathbf{q}) \tag{2.67}$$

which gives $\mathbf{G}_{q'} = \mathbf{W}\mathbf{q}'$ and $\mathbf{G}_q = \frac{\partial g}{\partial \mathbf{q}}$.

With the same reasoning as above, the solution must satisfy

$$\mathbf{q}'' = \mathbf{J}_W^\dagger(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}_W^\dagger\mathbf{J})\mathbf{W}^{-1}\left(-\mathbf{W}'\mathbf{q}' + \frac{\partial g}{\partial \mathbf{q}}\right) \tag{2.68}$$

with the two natural boundary conditions

$$\mathbf{q}'(\lambda_B) = \mathbf{J}_W^\dagger(\mathbf{q}(\lambda_B))\mathbf{x}'(\lambda_B) \tag{2.69}$$

2.4.6 Local versus global solutions

In Section 2.4.5 we have seen that global solutions such as (2.50), (2.56), (2.60) and (2.68) all resemble the local solution of equation (2.9), where ϕ'' may assume one form or the others to exactly reproduce each of the global solutions. Indeed, global optimization of performance indices built on velocities is equivalent to some kind of second-order local optimization of a different performance index. For instance, equation (2.50) corresponds to both the *global minimization* of least-square velocities and the *local minimization* of accelerations.

In Section 2.3.2, we recalled that other researchers pointed out that local second-order solutions may lead to anomalous behaviors. However, because of the analogy we highlighted, global solutions must necessarily be affected by the same problem. This observation deserves some additional words, as it constitutes one of the most important limitations of solutions based on Euler-Lagrange and Pontryagin necessary conditions.

In both [20] and [23], the authors observe that local minimization of torques could actually produce large, and indeed unfeasible velocities, accelerations and torques if some trajectories are given. In one of the cases, anomalies are demonstrated by using equation (2.50), which confirms that global solutions may be unfeasible too.

On the other hand, in [23], it is demonstrated that global minimization of torques always achieves the minimum, without causing any anomaly to the quantities in play, regardless of the assigned trajectory. However, the selected trajectories are very specific, i.e. straight lines of different lengths, thus we cannot conclude that, in general, performance indices exist whose global optimization leads to stable solutions for any possible trajectory.

From the theoretical standpoint, the reason for such anomalous behaviors is that pseudoinverse acceleration control is unstable in the vicinity of singularities [33]. This is confirmed by the fact that in all the examples in [20] and [23], anomalies happen when two

or more links are about to align.

In other words, saying that a performance index exists whose global optimization leads to stable solutions for any possible trajectory is equivalent to say that no trajectory exists leading the manipulator or any of its substructures in the vicinity of a singularity when globally optimizing the same performance index.

In conclusion, the distinction between anomalous and regular behaviors cannot be made on the basis of whether the performance index is locally or globally optimized, but it depends on the proximity to singularities to which the optimization of the performance index may lead.

A particular case is represented by those performance indices which, in order to be optimized, intrinsically require two or more links to align (an example is given with the concluding example of Section 2.7). In such cases, the calculus of variations fails in finding the global optimum as it generates solutions which either avoid passing through a singularity (local minimum) or are unfeasible/anomalous. In both cases, the performance index is not globally optimized.

Intuitively, these observations suggest that some connection must exist between the passage through singularities and the capability of the manipulator to find the optimum across the whole configurations space. We postpone the detailed study of this aspect to Section 2.7, where the argument is tackled from the topological point of view, after we will have given some additional elements necessary to the matter.

Summarizing, in light of the above, the user shall be aware that, by using global optimization algorithms, the solution obtained through calculus of variations may not be globally optimal with certain combinations of performance indices and trajectories. In Section 2.7, it will be shown that, with the use of dynamic programming, it is possible to overcome such issues, as the joint space configurations can be found without involving the first-order differential kinematics, but only relying on the inversion of position

kinematics. This guarantees the achievement of global optima while avoiding other undesirable behaviors.

2.5 Reduced order solutions

As anticipated in Section 2.4.1, equation (2.12) corresponds to a solution made of n second-order differential equations, equivalent to $2n$ first-order differential equations. However, since m algebraic equations, i.e. the direct kinematics, are also available, they can serve the purpose of reducing the number of differential equations to be solved.

2.5.1 $2n$ - m differential equations

Before employing the direct kinematics equations, it is worth recalling the findings of Martin et al. [25] who first noted that a reduced-order form of the solution, made of only $2n - m$ equations can be obtained. The idea is to define a vector $\boldsymbol{\gamma} \in \Re^r$ of null-space λ-velocities by projecting the joint velocities onto a basis of the null space of the Jacobian. These are used together with joint λ-velocities $\mathbf{q}' \in \Re^n$, so that, in the end, a system of $r+n = 2n-m$ equations is defined. Although they demonstrated it for the cost function (2.67), it is worth remarking that their results apply for all the solutions that can be expressed in the form

$$\mathbf{q}'' = \mathbf{J}_W^{\dagger}(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + \mathbf{P}_W\mathbf{A}(\mathbf{q}, \mathbf{q}', \lambda) \tag{2.70}$$

where $\mathbf{W}$ is a generic non-singular $n \times n$ symmetric weights matrix, possibly corresponding to the inertia matrix, $\mathbf{P}_W = (\mathbf{I} - \mathbf{J}_W^{\dagger}\mathbf{J})$ is the $\mathbf{W}$-dependent null-space projector of the Jacobian and $\mathbf{A}$ is a column vector whose form depends on the specific performance index. Expression (2.70), such as (2.12), corresponds to n second-order differential equations, equivalent, as said, to $2n$ first-order differential equations.

The reader may verify that this is indeed the form obtained for the most common performance indices mentioned above and certainly includes a broader category of solutions. In particular, despite the observations made in [28], we remark that this method applies as well for kinetic energy, for the variation of the weighted manipulability index and for any other quantity that might be expressed through a weight matrix.

The reduced order form proposed in [25] is based on a null space basis of the Jacobian. Said $\mathbf{N}$ such a basis, the weighted null-space basis $\mathbf{N}_W = \mathbf{WN}$ can be defined to handle the most general case. The relation between the weighted null-space projector and the weighted null-space basis can then be written as

$$\mathbf{P}_W = \mathbf{N}(\mathbf{N}_W^T\mathbf{N})^{-1}\mathbf{N}_W^T \tag{2.71}$$

Said $\boldsymbol{\gamma}$ the vector of null-space λ-velocities leading to internal motion, the following holds:

$$\boldsymbol{\gamma} = \mathbf{N}_W^T\mathbf{q}' \tag{2.72}$$

Recalling the generic first-order solution (2.6) and replacing the Jacobian pseudo-inverse with the weighted pseudo-inverse, we obtain:

$$\mathbf{q}' = \mathbf{J}_W^\dagger\mathbf{x}' + \mathbf{P}_W\boldsymbol{\phi}' \tag{2.73}$$

Considering that both $\mathbf{N}_W^T\mathbf{J}_W^\dagger = \mathbf{0}$ and $\mathbf{N}_W^T\mathbf{P}_W = \mathbf{N}_W^T$ hold, by putting (2.73) in (2.72), $\boldsymbol{\gamma}$ can also be written as

$$\boldsymbol{\gamma} = \mathbf{N}_W^T\boldsymbol{\phi}' \tag{2.74}$$

Now, putting (2.71) in (2.73) and using (2.74) leads to

$$\mathbf{q}' = \mathbf{J}_W^\dagger\mathbf{x}' + \mathbf{N}(\mathbf{N}_W^T\mathbf{N})^{-1}\boldsymbol{\gamma} \tag{2.75}$$

Differentiating (2.72), we have:

$$\boldsymbol{\gamma}' = \mathbf{N}_W'^T\mathbf{q}' + \mathbf{N}_W^T\mathbf{q}'' \tag{2.76}$$

Substituting (2.70) in the expression above, we obtain:

$$\boldsymbol{\gamma}' = \mathbf{N}_W'^T \mathbf{q}' + \mathbf{N}_W^T \mathbf{A}(\mathbf{q}, \mathbf{q}', \lambda) \tag{2.77}$$

Said $r = n - m$ the degree of redundancy, (2.75) and (2.77) together are a set of $n + r = 2n - m$ equations in $\mathbf{q}$ and $\boldsymbol{\gamma}$, which constitutes the reduced order form of (2.70). For instance, for a 7-DOF manipulator, such a technique would allow to integrate only 8 equations instead of 14.

As remarked in [25], finding $\mathbf{N}_W'^T$ might not be trivial, but, if an analytic form of the null-space basis exists and the degree of redundancy is one, it can be easily calculated by using the chain rule:

$$\mathbf{N}_W'^T = \mathbf{q}'^T \frac{\partial \mathbf{N}_W}{\partial \mathbf{q}} \tag{2.78}$$

2.5.2 2(n - m) differential equations

When Euler-Lagrange conditions have been introduced in Section 2.4.1, the possibility of reducing the system of $2n$ first-order differential equations and m algebraic equations, i.e. the direct kinematics, was briefly discussed. Since algebraic equations and their derivatives can be used, in principle, to simplify the differential equations, $2(n - m) = 2r$ differential equations remain to be solved. This is an important remark, as the number of differential equations only depends on the number r of redundant degrees of freedom, regardless of the total number of joints.

2.5.2.1 Parametrization through joint selection

Assume that the performance index to minimize is the square norm of velocities, but the dissertation can be easily generalized to any other performance index that can be managed with the Euler-Lagrange method, leading to n second-order differential equations. For the square norm of velocities, the Euler-Lagrange conditions

lead to the following differential equations:

$$\mathbf{q}'' = \mathbf{J}^{\dagger}(\mathbf{q}) \left[\mathbf{x}'' - \mathbf{J}'(\mathbf{q}, \mathbf{q}')\mathbf{q}'\right] \tag{2.79}$$

Without loss of generality, and to the only purpose of simplifying the notation, assume that $r = 1$, i.e. one degree of redundancy, and that we want to parametrize the redundancy with one of the joints, generically called q_i. We then pick the i-th equation from the set above:

$$q_i'' = \mathbf{J}^{\dagger}_{i,all}(\mathbf{q}) \left[\mathbf{x}'' - \mathbf{J}'(\mathbf{q}, \mathbf{q}')\mathbf{q}'\right] \tag{2.80}$$

where with $\mathbf{J}^{\dagger}_{i,all}(\mathbf{q})$ we refer to the i-th row of $\mathbf{J}^{\dagger}(\mathbf{q})$.

In order to reduce the order of the differential equation, explicitly write the differential equation linking the joint position and velocity, obtaining a system in the unknowns q_i and q_i':

$$\frac{d}{d\lambda}\begin{bmatrix} q_i \\ q_i' \end{bmatrix} = \begin{bmatrix} q_i' \\ \mathbf{J}^{\dagger}_{i,all}(\mathbf{q}) \left[\mathbf{x}'' - \mathbf{J}'(\mathbf{q}, \mathbf{q}')\mathbf{q}'\right] \end{bmatrix} \tag{2.81}$$

The terms in the second of the equations above are dependent on q_i, as well as on the other joints. Define as $\mathbf{q}_r \in \Re^m$ the position vector of the remaining joints:

$$\mathbf{q}_r = \begin{bmatrix} q_1 & \dots & q_{i-1} & q_{i+1} & \dots & q_n \end{bmatrix}^T \tag{2.82}$$

The second of the equations (2.81) can then be rewritten as:

$$q_i'' = \mathbf{J}^{\dagger}_{i,all}(q_i, \mathbf{q}_r) \left[\mathbf{x}'' - \mathbf{J}'(q_i, \mathbf{q}_r, q_i', \mathbf{q}_r')\mathbf{q}'\right] \tag{2.83}$$

Now consider the direct kinematics equations:

$$\mathbf{x} = \mathbf{k}(q_i, \mathbf{q}_r) \tag{2.84}$$

For a redundant manipulator, far from singularities, the inverse kinematics admits an infinite number of solutions. If q_i is driven

by the differential equations above, the other joints can be found by inverting equation (2.84). In general, it is not guaranteed, even in the case q_i is given, that the number of solutions will be finite. However, in the remaining of this section, with q_i given, we will assume that the set of solutions is always finite along the trajectory, while later in this section, we will analyze this issue further. We then select $\mathbf{q}_r$ from such a finite set:

$$\mathbf{q}_r = \mathbf{k}^{-1}(\mathbf{x}, q_i) \tag{2.85}$$

Also, if q_i is given, $\mathbf{k}(q_i, \mathbf{q}_r) = \mathbf{k}(\mathbf{q}_r)$, which implies that the Jacobian of $\mathbf{k}$ is a square matrix. Differentiating (2.84) with respect to λ we obtain:

$$\mathbf{x}' = \mathbf{J}(q_i, \mathbf{q}_r)\mathbf{q}' \tag{2.86}$$

Define $\mathbf{J}_{all,i}$ as the i-th column of $\mathbf{J}$ and $\mathbf{J}_r$ as the square matrix obtained by removing $\mathbf{J}_{all,i}$ from $\mathbf{J}$, getting to the following equation:

$$\mathbf{x}' = \mathbf{J}_{all,i}(q_i, \mathbf{q}_r)q_i' + \mathbf{J}_r(q_i, \mathbf{q}_r)\mathbf{q}_r' \tag{2.87}$$

Assuming that the kinematic substructure (i.e. the kinematic chain obtained by removing q_i) is not in a singularity we can solve for $\mathbf{q}_r'$, obtaining:

$$\mathbf{q}_r' = \mathbf{J}_r^{-1}(q_i, \mathbf{q}_r)\Big(\mathbf{x}' - \mathbf{J}_{all,i}(q_i, \mathbf{q}_r)q_i'\Big) \tag{2.88}$$

Putting all the results together, we obtain the following differential equations:

$$\frac{d}{d\lambda}\begin{bmatrix} q_i \\ q_i' \end{bmatrix} = \begin{bmatrix} q_i' \\ \mathbf{J}_{i,all}^{\dagger}(q_i, \mathbf{q}_r)\left[\mathbf{x}'' - \mathbf{J}'(q_i, \mathbf{q}_r, q_i', \mathbf{q}_r')\mathbf{q}'\right] \end{bmatrix} \tag{2.89}$$

which, together with equations (2.85) and (2.88), is demonstrated to be the reduced form of (2.79), with the minimum number of differential equations.

We have seen above that the minimum number of differential equations obtained by "fixing" one of the joints makes sense only when $\mathbf{J}_r$ is full-rank and thus, can be inverted. Since n of these Jacobians can be selected, one could think of choosing the one that never gets rank-deficient along the assigned trajectory.

Thus, in order to ensure that equation (2.88) can be used and that, as a consequence, the number of solutions of (2.85) is finite, singularities for the reduced Jacobians need to be identified. It is worth remarking that a singular Jacobian is a necessary condition for equation (2.85) to admit an infinite set of solutions, but not sufficient, i.e. singularities might be spotted for the reduced Jacobian not leading to an infinite number of solutions.

The following procedure should be followed to select the "redundant" joint:

1. Extract the possible square matrices $\mathbf{J}_r$ obtained from $\mathbf{J}$ by removing as many columns as the number of redundancy degrees. To the purpose of simplifying the dissertation, let's assume that $r = 1$ and call such matrices $\mathbf{J}_r^{(i)}$, with $i = 1..n$. The superscript in parentheses indicates the column removed from $\mathbf{J}$ to obtain the reduced Jacobian.

2. Identify conditions for which $\mathbf{J}_r^{(i)}$ becomes singular. For some manipulators, such as the one of Figure 2.3, it is possible to obtain such conditions analytically and some a-priori considerations could be made. Otherwise a trajectory shall be given and the distance from singularities shall be computed numerically in the same way as for non-redundant manipulators.

3. Once the joint is selected, which also implies a choice of a $\mathbf{J}_r^{(i)}$, trajectories that make it singular cannot be given. On the other hand, once a trajectory is given, none of the joints whose reduced Jacobian is singular for at least one trajectory point can be chosen to parametrize the redundancy.

In principle, it is possible that a trajectory is such that each of the reduced Jacobians gets rank-deficient for at least one point on the trajectory. If this is the case, one parametrization is not enough to represent the redundancy on the whole trajectory. Assume, for instance, that q_2 is being used for $t \in [t_0, t_k[$ and that $\mathbf{x}(t_k)$ forces the joints to make $\mathbf{J}_r^{(2)}$ singular. At time t_k, one could take the states computed up until then and use them as the initial condition for the same differential equations written for a different joint. In other words, equations (2.89) can be switched from one joint to the other anytime $\mathbf{J}_r$ is about to be singular.

To the purpose of clarifying this matter, Figure 2.2 provides two examples of a 3D PRRR (a) and a planar PRP (b) serial manipulators to which a 2D and a 1D task are assigned respectively.

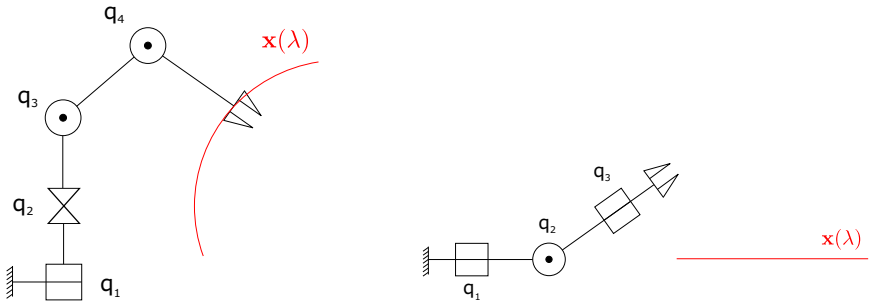


Figure 2.2: Kinematic structures for which the assigned task makes the joint q_2 meaningless in terms of redundancy parametrization

In both cases, if the manipulators were asked to track the depicted trajectories, only one value of q_2 would allow the end-effector to stay on the path. Note that, with q_2 given, both manipulators are still redundant with respect to the task they have to fulfill, which means that q_2 is not a suitable choice to represent redundancy. In facts, this confirms that the choice of the joint with respect to which the redundancy is parametrized cannot be randomly made.

If the manipulator is still redundant, even though r variables are

given, it means that two or more of the rows of $\mathbf{J}_r$ are linearly dependent, which, in turn, implies that $\mathbf{J}_r$ is not full rank for at least one point on the trajectory.

Thus, if we attempted to solve

$$\mathbf{q}_r = \mathbf{k}^{-1}(\mathbf{x}, q_2) \tag{2.90}$$

we would still get an infinite number of solutions.

It is also worth noticing that this issue is not strictly related to the fact that, said the selected joint q_i, $q_i'(\lambda) = 0\ \forall\lambda$, but to the condition that the manipulator is still redundant in achieving the task even when q_i is given. In facts, it is possible, in principle, to assign tasks, especially in constrained environments, for which one joint has to be still along the whole trajectory, but only a finite number of configurations is admissible for the other joints (i.e. the manipulator is no longer redundant with respect to the assigned task). In this case, joint q_i (just like any other joint) is still a suitable choice to parametrize the redundancy.

Example: Anthropomorphic shoulder mounted on a slide
The PRRR robot of Figure 2.2(a) is considered in this example. A 3D schematic of its kinematic chain is shown in Figure 2.3, where the base reference frame and bodies' lengths are also reported. Table 2.1 reports the Denavit-Hartenberg parameters used in this example.

From the considerations made above, we already know that if a path in the X-Z plane is given, only $q_2 = \pm\pi/2$ will satisfy the kinematic equations, while the substructure made of joints q_1, q_3 and q_4 will still be redundant as all the remaining joints operate on the path plane.

The direct kinematics equations are given by

$$\mathbf{k}(\mathbf{q}) = \begin{bmatrix} l_1 + l_2 + l_4 \sin(q_3 + q_4) + l_3 \sin q_3 \\ \cos q_2 \left(l_4 \cos(q_3 + q_4) + l_3 \cos q_3\right) \\ q_1 + \sin q_2 \left(l_4 \cos(q_3 + q_4) + l_3 \cos q_3\right) \end{bmatrix} \tag{2.91}$$

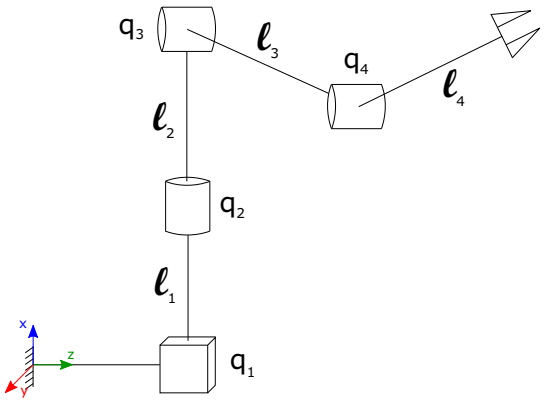


Figure 2.3: Schematic of an anthropomorphic shoulder mounted on a slide

	1	**2**	**3**	**4**
$\mathbf{d_i}$	q_1	l_1+l_2	0	0
$\boldsymbol{\theta}_i$	90°	q_2	q_3	q_4
$\mathbf{a_i}$	0	0	l_3	l_4
$\boldsymbol{\alpha}_i$	90°	90°	0	0

Table 2.1: Denavit-Hartenberg parameters for the robot of Figure 2.3

whose Jacobian is

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} 0 & 0 & l_4c_{34} + l_3c_3 & l_4c_{34} \\ 0 & -s_2\left(l_4c_{34} + l_3c_3\right) & -c_2\left(l_4s_{34} + l_3s_3\right) & -l_4c_2s_{34} \\ 1 & c_2\left(l_4c_{34} + l_3c_3\right) & -s_2\left(l_4s_{34} + l_3s_3\right) & -l_4s_2s_{34} \end{bmatrix} \tag{2.92}$$

where

$$\begin{aligned} s_i &= \sin q_i \\ c_i &= \cos q_i \\ s_{ij} &= \sin\left(q_i + q_j\right) \\ c_{ij} &= \cos\left(q_i + q_j\right) \end{aligned} \tag{2.93}$$

Proceed as indicated before in this section by computing the conditions making the reduced Jacobian matrices rank-deficient. For $l_1 = l_2 = l_3 = l_4 = 1$, we obtain:

$$|\mathbf{J}_r^{(1)}| = 0 \Leftrightarrow q_4 = 0 \vee q_4 = \pi \vee q_4 + 2q_3 = \pm\pi \tag{2.94}$$

$$|\mathbf{J}_r^{(2)}| = 0 \Leftrightarrow q_2 = \pm\frac{\pi}{2} \vee q_4 = 0 \vee q_4 = \pi \tag{2.95}$$

$$\begin{aligned} |\mathbf{J}_r^{(3)}| = 0 \Leftrightarrow q_3 + q_4 = \pm\frac{\pi}{2} \vee q_2 = 0 \vee q_2 = \pi \vee \\ \vee\, q_4 + 2q_3 = \pm\pi \vee q_4 = \pi \end{aligned} \tag{2.96}$$

$$|\mathbf{J}_r^{(4)}| = 0 \Leftrightarrow q_2 = 0 \vee q_2 = \pi \vee q_4 + 2q_3 = \pm\pi \vee q_4 = \pi \tag{2.97}$$

It is important to remark that only some of the conditions above are singularities for the whole kinematic chain. For instance, $q_4 = \pi$ clearly nullifies all the minors and is, therefore, a singularity for $\mathbf{J}$, but other combinations of the conditions above may produce the same effect.

Now assume that a trajectory on the X-Z plane is given, which is immediately recognized to drive q_2 toward the forbidden values of either $+\pi/2$ or $-\pi/2$. The same condition does not zero any other minor of $\mathbf{J}$, which means that any joint but q_2 can be selected to parametrize the redundancy for this trajectory, unless a singular condition for a different $\mathbf{J}_r^{(i)}$ can be reached, e.g. $q_3 + q_4 = \pm\pi/2$.

While all the conditions above are equivalent in terms of the effects they produce on the possibility to solve the redundant differential kinematics with only $2r$ equations, on the practical side, they do not restrict in the same way the number of trajectories that can be executed. For instance, $q_2 = \pm\pi/2$ excludes all the trajectories

spanning from one side of the slide to the other, while $q_4 = 0$ only excludes trajectories requiring the full length $l_3 + l_4$, that are more easily avoidable, by exploiting the capabilities of joint q_1 if the trajectory does not lie at the limits of the workspace.

One more aspect to consider concerns the fact that not all of the conditions above affect the position kinematics in the same way. For example, if a trajectory was such that $q_4 = 0$, whatever the selected joint was, the inverse kinematic solution would be immediately determined, if it exists. The same holds for conditions $q_3 + q_4 = \pm\pi/2$, $q_2 = 0$ and $q_2 = \pi$. On the other hand, as demonstrated above, condition $q_2 = \pm\pi/2$ of $\mathbf{J}_r^{(2)}$ implies that the inverse kinematics admit an infinite number of solutions. Lastly, condition $q_4 + 2q_3 = \pm\pi$ reduces the task space of one dimension, but the system of equations is still compatible, with a finite number of solutions for joints q_1 and q_3, while q_2 is free to rotate without affecting the task.

These observations suggest that some a-priori considerations can be made about

1. the feasibility of trajectories with a given parametrization

2. the meaningfulness of parametrizations with a given trajectory

Problem 1 concerns the derivation of conditions representing the feasible paths when q_i is defined. For instance, assume, again, that the manipulator is as in Figure 2.3 and $q_i = q_2$. We take the conditions by which the reduced Jacobian

$$|\mathbf{J}_r^{(2)}| = -c_2 s_4 \tag{2.98}$$

becomes rank-deficient and analyze the direct kinematic equations:

$$
\begin{aligned}
q_2 = \pm\frac{\pi}{2} \implies & \begin{cases} x = 2 + s_{34} + s_3 \\ y = 0 \\ z = q_1 \pm (c_{34} + c_3) \end{cases} \\
q_4 = 0 \implies & \begin{cases} x = 2 + 2s_3 \\ y = 2c_2c_3 \\ z = q_1 + 2s_2c_3 \end{cases} \\
q_4 = \pi \implies & \begin{cases} x = 2 \\ y = 0 \\ z = q_1 \end{cases}
\end{aligned}
\tag{2.99}
$$

The reader may recognize that

- the condition $q_4 = \pi$ is clearly a singularity of the whole kinematic chain, for which $rk(\mathbf{J}) = 2$;
- the condition $q_4 = 0$ regards the limits of the workspace along the x-axis and y-axis;
- the condition $q_2 = \pm\pi/2$ tells that the paths lying in the X-Z plane cannot be meaningfully parametrized through joint variable q_2, excluding the trajectory of Figure 2.2(a).

On the other hand, problem 2 concerns the determination of a suitable parametrization for a given path. For instance, assume that, for the same manipulator we would like to assign a generic path in the X-Z plane, hence constrained by the task-space condition

$$y = c_2(c_{34} + c_3) = 0 \tag{2.100}$$

from which we derive, also considering implications (2.94)-(2.97), the joint-space conditions

$$\begin{aligned} q_2 = \pm\frac{\pi}{2} &\implies |\mathbf{J}_r^{(2)}| = 0 \\ q_4 + 2q_3 = \pi &\implies |\mathbf{J}_r^{(1)}| = 0,\ |\mathbf{J}_r^{(3)}| = 0,\ |\mathbf{J}_r^{(4)}| = 0 \end{aligned} \tag{2.101}$$

The reader may recognize that $q_i = q_2$ is never a suitable parametrization in the plane X-Z, except when $y = 0$ because $q_4 + 2q_3 = \pi$ holds, corresponding to the case of the end-effector tracking a path (or at least one point) lying along the axis of the second joint. Depending on the actual path, this could be a case where every reduced Jacobian becomes rank-deficient for at least one point, requiring the user to implement a switch of the selected joint, or to adopt a different parametrization, as discussed in Section 2.5.2.2.

2.5.2.2 Parametrization through joint combinations

As noted above, the parametrizations based on a single joint are exclusively dependent on the kinematic characteristics of the manipulator, regardless of the assigned trajectory. This implies that a careful selection of the joint has to be made to allow for certain tasks to be executed. The choice has to be made between n possibilities or a switch between joints has to be implemented.

A solution which provides more flexibility is to parametrize the redundancy with respect to some joint combination $\mathbf{v} = \mathbf{k}_v(\mathbf{q})$. To other purposes, Burdick [34] used some physical quantities, like the orientation of one of the links with respect to some fixed reference system or the angle between an arbitrary plane and some joint axis. In laser cutting applications, the rotation angle of the laser tool about the axis perpendicular to the surface to cut is rather a natural choice [35]. To the purpose of our dissertation, we could choose similar physical or clearly measurable quantities, or some other joints combination that may have sense in some specific applications.

Once $\mathbf{v}$ is defined, as a set of r equations, the direct kinematics equations become:

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{k}(\mathbf{q}) \\ \mathbf{k}_v(\mathbf{q}) \end{bmatrix} = \mathbf{k}_a(\mathbf{q}) \tag{2.102}$$

By differentiating equation (2.102), we obtain:

$$\begin{bmatrix} \mathbf{x}' \\ \mathbf{v}' \end{bmatrix} = \begin{bmatrix} \mathbf{J}(\mathbf{q}) \\ \mathbf{J}_v(\mathbf{q}) \end{bmatrix} \mathbf{q}' = \mathbf{J}_a(\mathbf{q})\mathbf{q}' \tag{2.103}$$

The reader may recognize that such a formulation is equivalent to that of the augmented Jacobian technique [16] used in the frame of redundancy resolution via task augmentation. Here, although the same augmented Jacobian is obtained, $\mathbf{v}$ (i.e. the additional task) is not intended to be given along the trajectory, but has to be computed as part of the solution to the global optimization problem.

The objective is now to reorganize equations (2.79) in order to reduce them to a system of $2r$ equations in $\mathbf{v}$ and $\mathbf{v}'$. To do so, differentiate the second set of equations in (2.103), getting to:

$$\mathbf{v}'' = \mathbf{J}_v(\mathbf{q})\mathbf{q}'' + \mathbf{J}'_v(\mathbf{q})\mathbf{q}' \tag{2.104}$$

By substituting equation (2.79) and adding the differential equation linking the parameter to its derivative, we obtain

$$\frac{d}{d\lambda} \begin{bmatrix} \mathbf{v} \\ \mathbf{v}' \end{bmatrix} = \begin{bmatrix} \mathbf{v}' \\ \mathbf{J}_v(\mathbf{q})\mathbf{J}^{\dagger}(\mathbf{q}) \left[\mathbf{x}'' - \mathbf{J}'(\mathbf{q}, \mathbf{q}')\mathbf{q}'\right] + \mathbf{J}'_v(\mathbf{q})\mathbf{q}' \end{bmatrix} \tag{2.105}$$

which is the minimum number of differential equations if a combination of the joints is used.

Similarly to the joint selection case, if $\mathbf{v}$ is driven by the differential equations above, $\mathbf{q}$ and $\mathbf{q}'$ can be obtained by inverting equations (2.102) and (2.103)

$$\mathbf{q} = \mathbf{k}_a^{-1}(\mathbf{x}, \mathbf{v}) \tag{2.106}$$

$$\mathbf{q}' = \mathbf{J}_a^{-1} \begin{bmatrix} \mathbf{x}' \\ \mathbf{v}' \end{bmatrix} \tag{2.107}$$

As noted in [16], the augmented Jacobian suffers from the so-called *algorithmic* singularities, which are conditions in which $\mathbf{J}_a$ is rank-deficient, even though the manipulator is not singular. To the purpose of parametrizing the redundancy, one could choose the suitable joints combination which allows keeping the singularity away from the assigned task.

Also, it can be demonstrated that, outside of the singularities of $\mathbf{J}$

$$rk(\mathbf{J}_a) = n \iff \mathcal{R}(\mathbf{J}^T) \cap \mathcal{R}(\mathbf{J}_v^T) = \varnothing \tag{2.108}$$

where with $\mathcal{R}(\mathbf{A})$ we refer to the range space of $\mathbf{A}$.

Unlike the augmented Jacobian redundancy resolution method, here we are not interested in assigning a second task to the manipulator, but to ensure that the selected parametrization is always meaningful along the trajectory and, possibly, regardless of the trajectory. This implies that $\mathbf{J}_v$ can be suitably selected to make condition (2.108) always verified. For instance this can be done by computing a basis of $\mathcal{R}(\mathbf{J}_v^T)$ that completes the basis of $\mathcal{R}(\mathbf{J}^T)$ to n, i.e.

$$\mathcal{R}(\mathbf{J}_v^T) = \mathcal{R}^{\perp}(\mathbf{J}^T) \tag{2.109}$$

If this condition is satisfied, the two mappings are orthogonal [16] and $\mathbf{J}_a$ is always full-rank.

For the example of Figure 2.3, we may define

$$\mathbf{J}_v = \begin{bmatrix} j_1 & j_2 & j_3 & j_4 \end{bmatrix} \tag{2.110}$$

and solve

$$\mathbf{J}\mathbf{J}_v^T = \mathbf{0} \tag{2.111}$$

One possible solution is given by

$$\begin{aligned} j_1 &= -s_4\left(c_{34} + c_3\right) \\ j_2 &= c_2 s_4 \\ j_3 &= s_2 c_{34}\left(c_{34} + c_3\right) \\ j_4 &= -s_2\left(c_{34} + c_3\right)^2 \end{aligned} \tag{2.112}$$

which gives the augmented Jacobian

$$\mathbf{J}_a = \begin{bmatrix} 0 & 0 & \alpha & c_{34} \\ 0 & -s_2\alpha & -c_2\beta & -c_2 s_{34} \\ 1 & c_2\alpha & -s_2\beta & -s_2 s_{34} \\ -s_4\alpha & c_2 s_4 & s_2 c_{34}\alpha & -s_2\alpha^2 \end{bmatrix} \tag{2.113}$$

with

$$\begin{aligned} \alpha &= c_{34} + c_3 \\ \beta &= s_{34} + s_3 \end{aligned} \tag{2.114}$$

This parametrization chosen, $\mathbf{v}$ is not attributable, in general, to any physical quantity that one may easily visualize, even for very simple 3-DOF planar manipulators. Also, its analytic form is not trivial to find. As a consequence, the system of equations results to be more suitable for numerical CLIK-based implementations [22].

Lastly, it is important to notice that the formulation presented here represents a more general case of the joint selection method presented in Section 2.5.2.1, whose outcomes could be easily refound by imposing $r = 1$ and $\mathbf{v} = v = q_i$.

2.6 Topological analysis of inverse kinematic mappings

2.6.1 Fundamentals of manipulator topology

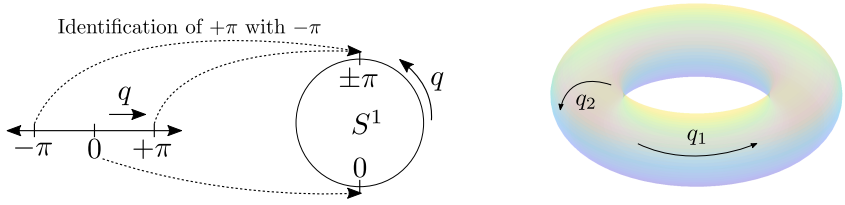
Preparatory for the discussions that follow in the next sections are some considerations concerning the topology of solutions to the inverse kinematics problem. The objective of this section is to recall some nomenclature and concepts that are useful to describe the inverse kinematic mapping from the topological standpoint, in order to fix them in the reader's mind before using them hereinafter. Results and figures of this section are taken from [34] and [25] and suitably modified when necessary to be consistent with the nomenclature used throughout this dissertation.

Consider the direct kinematic mapping of equation (2.1). The vector function $\mathbf{k}$ maps a joint configuration $\mathbf{q}$ to an end-effector location and orientation $\mathbf{x} = \mathbf{k}(\mathbf{q})$. The set of all possible joint configurations $\mathcal{C}$ is termed *joint space* or *configuration space*, while the set of all possible end-effector locations and orientations $\mathcal{W}$ is termed *workspace*. Both of them have a manifold structure. The function $\mathbf{k}$ can then be seen as the global mapping rearranging the configuration space manifold to produce the workspace manifold:

$$\mathbf{k}(\mathbf{q}) : \mathcal{C} \rightarrow \mathcal{W} \tag{2.115}$$

In the case of a revolute joint j with no limits, q_j can take all the values in $[-\pi, \pi]$. However, since $-\pi = \pi$, the configuration space of q_j closes on a circle, denoted S^1, as shown in Figure 2.4(a). In the case of a 2R planar manipulator, two circles of type S^1 can be combined together to form a *two-dimensional torus* (or just *2-torus*) as in Figure 2.4(b).

For the most general case of the manipulator with n revolute joints, the configuration space $\mathcal{C}$ is equivalent to an n-torus, which is a compact n-dimensional manifold, and can be formalized as the



(a) (b)

Figure 2.4: Configuration space of (a) a revolute joint and (b) a 2R manipulator

n-times product of the individual joints' manifolds:

$$\mathcal{C} = S^1 \times S^1 \times ... \times S^1 = T^n \tag{2.116}$$

Each of the circles that make up the torus is termed a *generator* of the torus. By cutting along generators, it is possible to represent them in a Cartesian space, which provides an alternative visualization of the configuration space, as in Figure 2.5. It is worth mentioning that the three-dimensional representation of a 3-torus is not possible, while its Cartesian representation is.

The geometric representation of $\mathcal{W}$ is more complex than the torus representation of $\mathcal{C}$ as it is made up of both linear (end-effector position) and angular (end-effector orientation) dimensions. For this reason, it will not be reported here and, to the purpose of keeping the dissertation as clear and simple as possible, examples limiting the workspace to position-only dimensions will be considered, so that a Cartesian representation of $\mathcal{W}$ will be possible.

Singularities notoriously play an important role in kinematics mapping. Thus it is important to provide a classification of joints configurations and relating workspace locations with respect to them. In the case of a non-redundant manipulator, the following definitions are given:

- a *regular point* is a configuration $\mathbf{q} \in \mathcal{C}$ for which $\mathbf{J}(\mathbf{q})$ is full-rank, i.e. the manipulator is not singular;

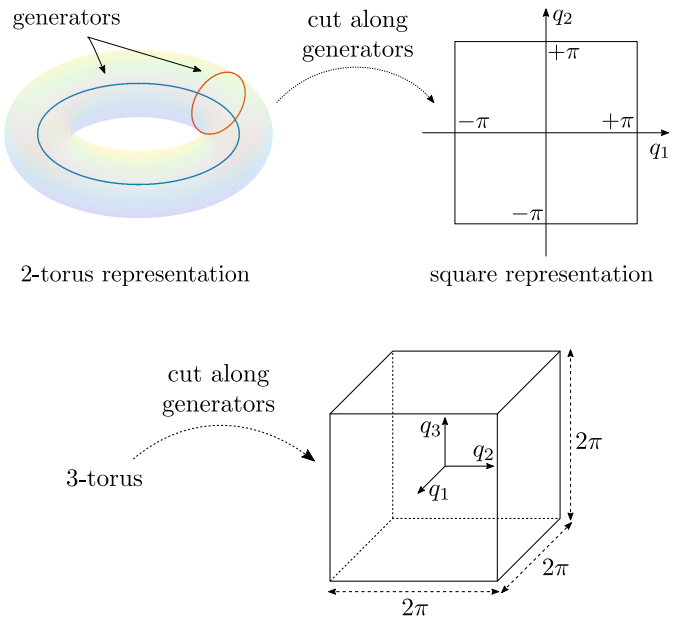


Figure 2.5: Cartesian representation of a 2-torus (top) and a 3-torus (bottom)

- a *regular value* is the image $\mathbf{x} \in \mathcal{W}$ of a regular point, i.e. a location of the workspace where the manipulator is not singular;
- a *critical point* is a configuration $\mathbf{q} \in \mathcal{C}$ for which $\mathbf{J}(\mathbf{q})$ is rank-deficient, i.e. the manipulator is singular;
- a *critical value* is the image $\mathbf{x} \in \mathcal{W}$ of a critical point, i.e. a location of the workspace where the manipulator is singular.

It is worth observing that, for a non-redundant manipulator:

- a regular value's pre-image is a finite set of regular points (e.g. *elbow-up* and *elbow-down* configurations for a 2R manipulator), whose cardinality only depends on the mechanical characteristics of the manipulator; such regular points are isolated, which means that it is not possible to pass from

one to the other without moving the end-effector from its current location;

- a critical value's pre-image can be made up of infinite critical points, e.g. for a 3R anthropomorphic arm, when the end-effector is situated along the first joint's axis.

Now consider a redundant manipulator with degree of redundancy $r = n - m$. Given that ∞^r inverse kinematics solutions exist for a certain $\mathbf{x} \in \mathcal{W}$, the pre-image of $\mathbf{x}$ is an r-dimensional subspace of $\mathcal{C}$. Similarly to the non-redundant case, where the pre-image of a certain $\mathbf{x}$ is made up of isolated configurations, the r-dimensional subspace could be made up of disjoint r-dimensional manifolds. The inverse kinematics mapping can then be written as:

$$\mathbf{k}^{-1}(\mathbf{x}) = \bigcup_{i=1}^{N_g} M_i(\mathbf{q}) \tag{2.117}$$

where N_g is the total number of manifolds in the pre-image, that can be demonstrated to be at most 2,2,4 and 16 for planar, spherical, regional and spatial manipulators respectively. Each of the pre-image manifolds $M_i(\mathbf{q})$ physically corresponds to a "self-motion", which is a continuous motion of the joints which leaves the end-effector motionless. For this reason each manifold in the pre-image will be referred to as *self-motion manifold*.

Differently from the non-redundant case, for certain locations in the workspace, multiple manifolds may be connected at certain configurations, building a unique self-motion manifold. This happens because the manipulator, with a continuous motion of its joints, can pass through all the points contained in the pre-image. With reference to Figure 2.6, the pre-image of $\mathbf{x}_1$ is made up of two disjoint self-motion manifolds, while the pre-image of $\mathbf{x}_2$ is made up of two adjoint manifolds forming one self-motion manifold. The reader may verify that it is not possible to obtain all the configurations keeping the end-effector in $\mathbf{x}_1$ by a continuous motion of the joints.

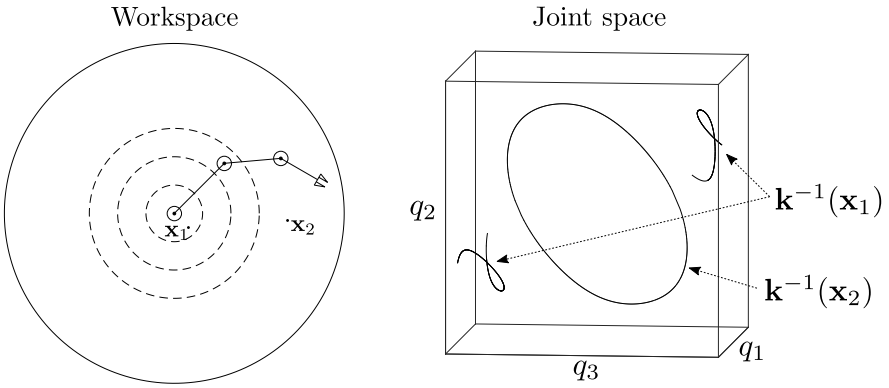


Figure 2.6: Two points in $\mathcal{W}$ (left) and their pre-images in $\mathcal{C}$ (right) for a 3R redundant manipulator: the pre-image of $\mathbf{x}_2$ is made of two self-motion manifolds closing in a circle, whereas the pre-image of $\mathbf{x}_1$ is made of two disjoint self-motion manifolds

As far as singularities are concerned, since infinite configurations $\mathbf{q}$ may exist for certain $\mathbf{x} \in \mathcal{W}$, redundant manipulators are able to choose one that does not make $\mathbf{J}(\mathbf{q})$ rank-deficient. In other words, depending on the end-effector location, redundant manipulators could be able to avoid singularities. Thus, it is necessary to reconsider the definitions given for non-redundant manipulators:

- the definitions of *regular point* and *critical point* are unchanged;
- a *regular value* is a value $\mathbf{x} \in \mathcal{W}$ whose pre-image only contains regular points;
- a *coregular value* is a value $\mathbf{x} \in \mathcal{W}$ whose pre-image contains both regular and critical points;
- a *critical value* is a value $\mathbf{x} \in \mathcal{W}$ whose pre-image only contains critical points.

Collections of coregular values form *coregular value manifolds* in $\mathcal{W}$. With reference to Figure 2.6, the dashed circumferences in

the workspace are coregular value manifolds, as the pre-images of their points contain critical (i.e. links folded on each other) and regular points. The outermost circumference is rather a collection of critical values as their pre-images only contain critical points (i.e. stretched arm). The coregular value manifolds partition the workspace $\mathcal{W}$ in subspaces called $\mathcal{W}$-*covers*.

The pre-images of coregular value manifolds are termed *coregular surfaces* and the pre-images of $\mathcal{W}$-covers are termed $\mathcal{C}$-*bundles*. Just like the coregular value manifolds partition $\mathcal{W}$ in several $\mathcal{W}$-covers, the coregular surfaces partition $\mathcal{C}$ in several $\mathcal{C}$-bundles. Because the pre-image of a regular point could be made up of distinct self-motion manifolds or of one self-motion manifold, the pre-image of a $\mathcal{W}$-cover is made up of distinct $\mathcal{C}$-bundles or of one $\mathcal{C}$-bundle, as Figure 2.7 illustrates.

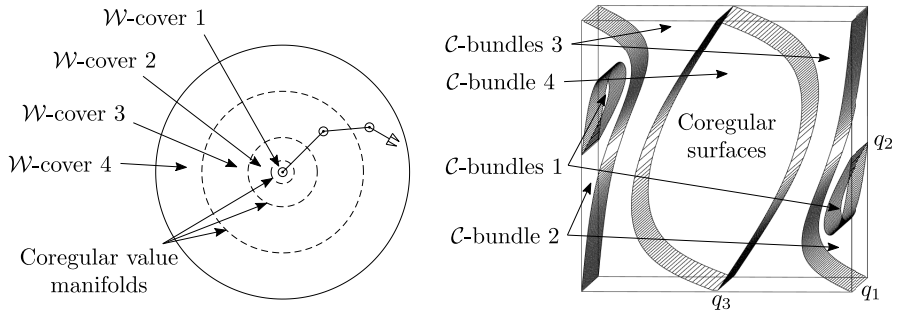


Figure 2.7: $\mathcal{C}$-bundles and coregular surfaces (right) and relating $\mathcal{W}$-covers and coregular value manifolds (left) for a 3R planar manipulator

Since the axes of the configuration space in Figure 2.7 are limited to $[-\pi, \pi]$ for each of the joints, the same pattern repeats over and over again for other 2π-wide intervals outside of $[-\pi, \pi]$. This means that the pre-images of $\mathcal{W}$-covers 4 and 2 are made up of one $\mathcal{C}$-bundle each (one connected volume in $\mathcal{C}$), while the pre-images of $\mathcal{W}$-covers 1 and 3 are made up of two disjoint $\mathcal{C}$-bundles each.

On the basis of the observations made above, the 3R planar manipulator is not able to fully reconfigure itself in $\mathcal{W}$-covers 1 and 3, as

not all the configurations in $\mathcal{C}$-bundles 1 and 3 can be reached with a continuous motion of the joints that keeps the end-effector motionless. In such situations, singularities, belonging to the coregular surfaces, bound and actually separate the $\mathcal{C}$-bundles that are pre-images of the same $\mathcal{W}$-cover. In particular, the singular configuration $\mathbf{q} = [q_1, 0, \pi]^T$ breaks the pre-image of $\mathcal{W}$-cover 1 in two distinct $\mathcal{C}$-bundles and likewise does the singular configuration $\mathbf{q} = [q_1, \pi, 0]^T$ with the pre-image of $\mathcal{W}$-cover 3.

When moving across several $\mathcal{C}$-bundles, for instance, when following a path in the workspace, extending over more than one $\mathcal{W}$-cover, the manipulator may transit from a $\mathcal{C}$-bundle in which it can freely reconfigure (i.e. visit all the inverse kinematic solutions through internal motion) to a $\mathcal{C}$-bundle where this is no longer possible. An example will be given in Section 2.7.5. Although this is not fundamental for the conclusions that we draw at the end of this section, it is an important aspect to keep in mind because it relates to the sub-optimality of certain redundancy resolution schemes. For example, local redundancy resolution techniques like (2.6), based on a first-order gradient search, at each t, can only search over one self-motion manifold, while the true optimum might lie on another self-motion manifold. Also, some supposedly global resolution methods could be prone to sub-optimal solutions due to the existence of distinct self-motion manifolds. This will be clarified in Section 2.6.3, after introducing the concept of *homotopy class* in Section 2.6.2.

2.6.2 Homotopy relations

2.6.2.1 C-homotopy

Before proceeding further in analyzing how multiple, sometimes disjoint $\mathcal{C}$-bundles affect the computation of the dynamic programming state space and ultimately provide guidelines for the design of an effective globally-optimal redundancy resolution algorithm, a few considerations about homotopies are worthwhile.

Each sequence of joint configurations, whether it corresponds to the same end-effector location (i.e. configurations in the same self-motion manifold) or to a workspace path, can be represented as a curve on the surface of the n-torus. Therefore, such can also be done for $\mathbf{k}^{-1}(\mathbf{x}_1)$ and $\mathbf{k}^{-1}(\mathbf{x}_2)$ of Figure 2.6. Although it is not immediately evident by looking at their Cartesian representation, the extremal points of each of the self-motion manifolds corresponding to $\mathbf{k}^{-1}(\mathbf{x}_1)$ are the same, as the q_1 axis extends from $-\pi$ to π, which, indeed, are the same value. This means that the self-motion manifolds over $\mathbf{x}_1$ include a 2π joint rotation along the generator of q_1 on the surface of the 3-torus. On the contrary, the pre-image of the regular value $\mathbf{x}_2$ does not. From this observation we conclude that they are not homotopic, as one cannot be continuously deformed into the other on the surface of the torus.

More formally, the homotopy class of a self-motion can be specified by an n-tuple of integers $(I_1, I_2, ..., I_n)$, where I_j counts how many integral times the self-motion manifold wraps around the j^{th} generator of the n-torus [34]. Since the n-torus is a representation of the whole configuration space $\mathcal{C}$, we define this homotopy relation as $\mathcal{C}$-*homotopy*. $\mathcal{C}$-homotopy is used in [36] to characterize the topology of redundant manipulators subject to joint limits.

There is not an easy way to characterize, in terms of cardinality, the $\mathcal{C}$-homotopy classes of self-motions for generic redundant manipulators [34]. Also, a procedure that associates $\mathcal{C}$-bundles with $\mathcal{C}$-homotopy classes does not exist. From the observation of the simple planar case, we can see that all the self-motions belonging to the same $\mathcal{C}$-bundle also belong to the same $\mathcal{C}$-homotopy class, but self-motions from different $\mathcal{C}$-bundles could also be homotopic. For example, this is the case of self-motions in $\mathcal{C}$-bundles 2 and 4. The generalization of this concept to more complex kinematic structures is an open problem.

Finally, we should note that the $\mathcal{C}$-homotopy is in no way related to the sub-optimality of local and global optimization techniques, like calculus of variations, while the presence of disjoint manifolds is.

2.6.2.2 C-path-homotopy

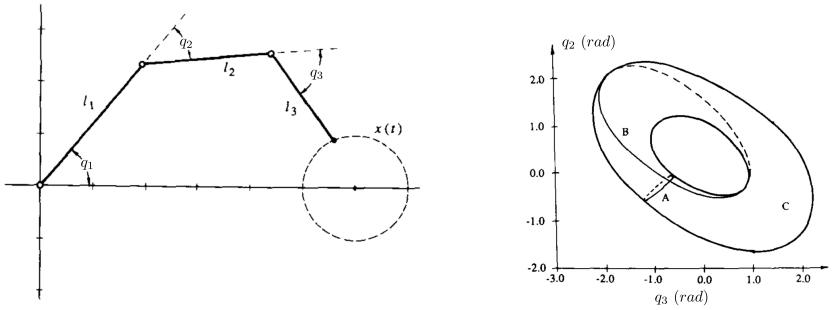
As mentioned above, self-motion manifolds are closed curves on the surface of the n-torus representing the configuration space, and so are cyclic inverse kinematic solutions of a cyclic workspace path. Thus, as for self-motion manifolds, it is possible to define an equivalence relation between closed joint space paths in terms of $\mathcal{C}$-homotopy classes they belong to.

However, when the authors of [25] conclude that global inverse kinematic techniques are unable to distinguish between solutions in differing homotopy classes, they do not look at the surface of the n-torus of the configuration space, but at the surface that, in the Cartesian joint space, sequences of self-motion manifolds (pre-images of the regular and coregular values belonging to a continuous workspace path) form. Since this homotopy relation is linked to the pre-image in $\mathcal{C}$ of the workspace path, which is a different topological space from the n-torus, we define it as $\mathcal{C}$-*path-homotopy*. The reader should be aware though that this definition is different from that of *path homotopy*, used in [28] to refer to homotopic curves with equal end-points.

Hence, even though two joint space path were $\mathcal{C}$-homotopic, they could be not $\mathcal{C}$-path-homotopic. It is convenient to address this matter by recalling the example of [25].

Consider the 3R planar manipulator of Figure 2.8(a), with links of lengths $l_1 = 3.0$, $l_2 = 2.5$ and $l_3 = 2.0$, whose end-effector is requested to follow a circular path centered in $C = [6, 0]^T$ and radius $R = 1$ in unit time in counterclockwise direction, starting from $\mathbf{x}(0) = [5, 0]^T$. As an additional requirement, initial and final joint positions must be the same: the solution must be *cyclic* (or *conservative*).

First of all, we notice that the workspace path only lies in the outermost $\mathcal{W}$-cover. As for the 3R planar manipulator of Figure 2.6, each regular value in such a $\mathcal{W}$-cover pre-maps to only one self-motion manifold. The sequence of self-motion manifolds obtained



(a) (b)

Figure 2.8: (a) geometry of the manipulator and trajectory; (b) two inverse kinematics solutions (A and B) and the surface made from self-motion manifolds on a continuous workspace path projected onto the $q_2 - q_3$ plane (C)

from inverse kinematics of all the points on the workspace path forms a three-dimensional surface which is similar to a deformed torus. The projection on the $q_2 - q_3$ plane, labeled 'C', is illustrated in Figure 2.8(b). In other words, the deformed torus is the pre-image in $\mathcal{C}$ of the workspace path, obtained as the composition of the infinite self-motion manifolds traversed by the manipulator on its path. As long as paths are circular and lie in the outermost $\mathcal{W}$-cover, they all pre-map to deformed tori.

Because of the cyclicity requirement, each solution to the inverse kinematics problem will be a closed curve on the surface of such a torus starting from one of its outermost points, reaching one of the innermost points at the path's middle point and going back to the starting point again. In Figure 2.8(b), two of these solutions, labeled 'A' and 'B', are projected onto the torus surface in $\mathcal{C}$. For the performance index used in [25] in this example, A is of lower cost than B and circles the torus about the small radius. The higher cost trajectory circles the torus about the small radius once, but, at the same time, circles the torus once about the large radius, resulting in a longer trajectory in the joint space. Each of the curves cannot be transformed into the other: according to

differential topology, A and B are in different homotopy classes [25], i.e. A and B are not $\mathcal{C}$-path-homotopic.

It is anyhow worth noticing that workspace paths crossing the boundaries of a $\mathcal{W}$-cover, as well as more complex paths than circumferences, pre-map to more complex surfaces in $\mathcal{C}$ than deformed tori (see Section 2.7.5). Thus, while in the example of Figure 2.8 we can determine whether the solutions are homotopic or not from a pure visual analysis, this will not be possible in general.

From the observations made in this section, we can conclude that globally-optimal redundancy resolution schemes, as those derived from calculus of variations, may provide locally-optimal solutions (that are indeed globally-optimal in their $\mathcal{C}$-path-homotopy class), such as trajectory B, because of the existence of multiple $\mathcal{C}$-path-homotopy classes in the pre-image of the workspace path. In order to prove that $\mathcal{C}$-homotopy and $\mathcal{C}$-path-homotopy are different homotopy relations, the reader may verify that both A and B lie in the same $\mathcal{C}$-bundle, they do not wrap around any of the generators of the n-torus representing the whole configuration space and are, therefore, $\mathcal{C}$-homotopic.

2.6.3 C-path-homotopy classes and disjoint self-motion manifolds

From a recent study [26], it emerged that a relationship exists between the disjoint self-motion manifolds theorized in [34] and the presence of $\mathcal{C}$-path-homotopy classes, as defined in [25] and in Section 2.6.2.2. In particular, self-motion manifolds may break or join when the joint space solution crosses some coregular surface. For instance, with reference to Figure 2.6 and Figure 2.7, when the manipulator transits from $\mathcal{C}$-bundle 4 to $\mathcal{C}$-bundle 3, the adjoint manifold breaks into two disjoint manifolds that only meet at singular points along the coregular surface. If we imagined to trace two joint-space paths exiting $\mathcal{C}$-bundle 4 and entering two

different regions of $\mathcal{C}$-bundle 3, it would not be possible to continuously deform one into the other: the two paths would not be $\mathcal{C}$-path-homotopic. Hence, disjoint self-motion manifolds have the property of bifurcating the pre-image of the workspace path in different branches generating new $\mathcal{C}$-path-homotopy classes. Since this can only happen on a $\mathcal{C}$-bundle transition, such bifurcation points are identified to belong, for a given path, to the coregular value manifolds.

In going from the first to the last self-motion manifold (terminal points of a given workspace path), the joint space path may traverse several coregular value manifolds and, for the sub-paths laying in disjoint $\mathcal{C}$-bundles, take one of the branches. All the possible branches generate several routes. According to [26], for free boundary conditions, the number of $\mathcal{C}$-path-homotopy classes is exactly given by the number of such routes:

$$N_H = 2 \prod_{k=2}^{K-1} N_h(k) \tag{2.118}$$

where K is the number of $\mathcal{C}$-bundles that are traversed by the manipulator (counted as many times as traversed) and $N_h(k)$ is the number of $\mathcal{C}$-path-homotopy classes associated to the k-th $\mathcal{C}$-bundle. The latter is not, in general, equal to the number of disjoint regions in a $\mathcal{C}$-bundle since, for closed manifolds, the joint space path can wrap around for an arbitrary number of times. By looking at the equation above, it is clear that the number of homotopy classes rapidly increases as the number of coregular value crossings increases.

In [26], three typical shapes (or structures) of the pre-image of the workspace path are analyzed for a 3R planar manipulator with one degree of redundancy, showing that, already for this simple problem, $N_h(k)$ can be 1, 3 or 5. This result is obtained by assuming that the joint space path cannot wrap more than once for each direction (clockwise or counter-clockwise).

2.6.4 Aspects, extended aspects and multiple IK solutions

One more subject that comes in help for the discussions that will follow in Section 2.7 is the notion of *aspect* [37]. By using the same convention as Section 2.5.2.1, thus assuming, for the sake of simplyfing the notation, that $r = 1$, let $|\mathbf{J}_r^{(i)}(\mathbf{q})|$, with $i = 1..n$, be the determinants of the $(n-1) \times (n-1)$ minors of a redundant manipulator Jacobian $\mathbf{J}$. An aspect $\mathcal{D}$ can be defined as a connected set of points in $\mathcal{C}$ such that $|\mathbf{J}_r^{(i)}(\mathbf{q})| \neq 0 \ \forall \mathbf{q} \in \mathcal{D}$. Thus, by definition, the border of the aspect does not belong to the aspect [38].

Since $|\mathbf{J}_r^{(i)}(\mathbf{q})| = 0$ for any i makes a partition of $\mathcal{C}$ in two subspaces, characterized by conditions $|\mathbf{J}_r^{(i)}(\mathbf{q})| > 0$ and $|\mathbf{J}_r^{(i)}(\mathbf{q})| < 0$, one may think that 2^n aspects are generated from all the possible combinations positive/negative of the n minors determinants. However, already for the simple case of a 3R planar manipulator, more than 2^n aspects are generated, as shown in Figure 2.9. Aspects corresponding to the same combination of determinants signs are identified by the same color and are separated by a single point nullifying two minors together, as for the couples 1-5, 4-12, 7-11 and 6-10. Since the border of the aspect does not belong to the aspect, they are all separated aspects. On the other hand, if some combination of signs cannot happen for any configuration in $\mathcal{C}$, or joint limits exist, the number of admissible aspects can be less than 2^n.

It is worth noting that the condition $|\mathbf{J}_r^{(i)}(\mathbf{q})| = 0$ for some i physically corresponds to the singularity of the relating kinematic subchain, by which the manipulator has to transit in order to pass from one set of postures to another, e.g., from *elbow-down* to *elbow-up* for a planar manipulator. As a consequence, if a solution to (2.1) exists in aspect $\mathcal{D}$, obtained by fixing r joint variables, it is unique in the whole aspect. Being the number of aspects greater than the number of inverse kinematics solutions with r joint variables given, it follows that several aspects will not contain any of

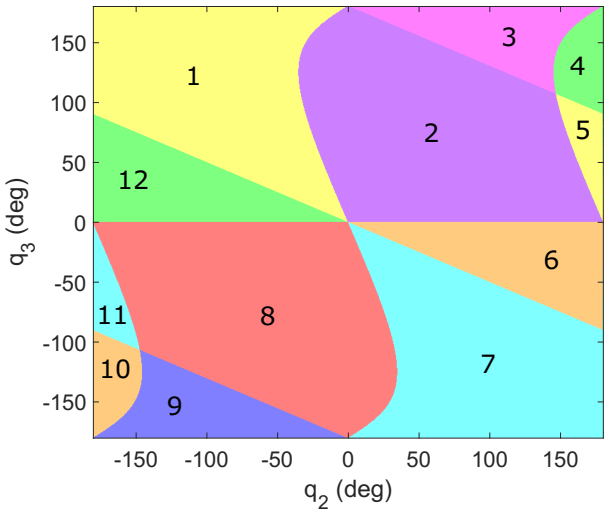


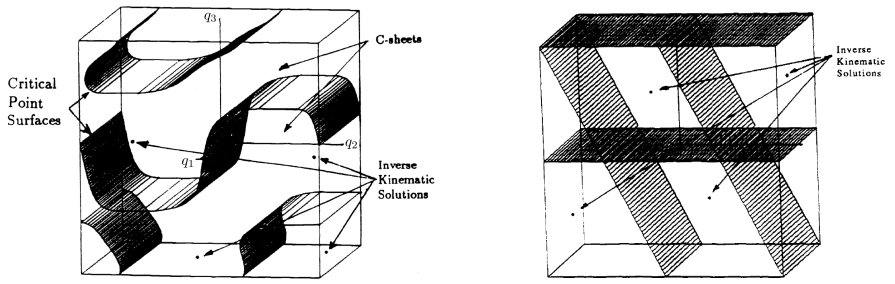
Figure 2.9: Aspects partition of $\mathcal{C}$ for a 3R planar manipulator with link lengths $l_1 = 7$, $l_2 = 7$, $l_3 = 4$, projected onto the q_2-q_3 plane

such solutions.

In Section 2.5.2.2, we have seen that redundancy parametrization through joint combination, which also includes the joint selection case, leads to the definition of an extended (or augmented) Jacobian $\mathbf{J}_a$, introducing algorithmic singularities. Like the singularities of non-redundant manipulators, in the general case, they constitute hypersurfaces in $\mathcal{C}$, generating a partition of $\mathcal{C}$ in sub-spaces. By extending the notion of aspect just mentioned to the augmented Jacobian, each of this sub-spaces is referred to as an *extended aspect* [39], and is identified by the signs of the factors of $|\mathbf{J}_a(\mathbf{q})|$ [40].

The notion of extended aspect is indeed equivalent to the notion of $\mathcal{C}$-*sheet*, applicable to non-redundant manipulators, that are defined as the disjoint regions into which the configuration space is partitioned by the hypersurfaces formed by the loci of singular configurations (or critical points). $\mathcal{C}$-sheets have been introduced

in [41] and therein thoroughly studied in relation to the *genericity* property of manipulators [42]. In brief, *generic* manipulators have non-intersecting critical point surfaces, making a smooth partition of $\mathcal{C}$, as in Figure 2.10(a), while *non-generic* manipulators have intersecting critical point surfaces, making a non-smooth partition of $\mathcal{C}$, as in Figure 2.10(b). It appears that non-genericity often arises from geometric simplification conditions, like two intersecting or parallel joint axes, and that most industrial manipulators are, in turn, non-generic. This also means that, when manufacturing an industrial manipulator, attention must be paid to the mechanical tolerances, which may turn a non-generic manipulator into a generic manipulator, drastically modifying its global kinematic properties [43].



(a) (b)

Figure 2.10: Inverse kinematic solutions, critical point surfaces and $\mathcal{C}$-sheets for a generic (left) and non-generic (right) 3R regional manipulator

Inverse kinematic solutions for a given $\mathbf{x}$ are superimposed on both images of Figure 2.10. The reader may recognize that, for the generic manipulator of Figure 2.10(a), there exists a one-to-many association between $\mathcal{C}$-sheets (aspects of a non-redundant manipulator) and inverse kinematic solutions (or postures). Rather, if the manipulator has to pass by a singularity to reconfigure its posture, each $\mathcal{C}$-sheet cannot contain more than one solution to the inverse kinematic problem, as for the non-generic manipulator of Figure 2.10(b). In [43], these kinematic structures are referred to

as *non-cuspidal* manipulators (or *type-1* manipulators [38]), and include all the common industrial manipulators. On the contrary, if the manipulator is *cuspidal* [43] (or with a *type-2* geometry [38]), more than one posture exists in the same aspect, separated by the so-called *characteristic surfaces*, theorized in [38]. Most generic manipulators are cuspidal, while most non-generic manipulators are non-cuspidal, but there are some non-generic manipulators which can change their posture without encountering a singularity [41], as well as some generic manipulators which must pass through a singularity to change their posture [43]. The belonging to the class of cuspidal manipulators is an important property, as it allows to distinguish between postures by using the theory of aspects.

As we will see in Section 2.7, extended aspects play an important role in the partition of the solutions space for dynamic programming algorithms and provide a partition criterion for any of the redundancy parametrization schemes discussed in Section 2.5.2. In facts, for a given end-effector location and orientation, the extended aspects contain a constant number of inverse kinematics solutions [38]. Thus, in the case of planar and spherical manipulators, where only two inverse kinematic solutions exist, the determinant of the augmented Jacobian provides an analytic solution to the problem of distinguishing between them. For some regional type 1 manipulators, when the explicit factorization of their Jacobian determinant is possible, an analytic solution could still be found. However, for more complex robots, this is no longer the case, and numeric techniques shall be adopted.

2.7 Dynamic programming

As remarked in Section 2.2, the traditional formulation of the global optimization problem lacks of the possibility to find performance indices which can easily accommodate a multitude of constraints that are common in real applications, such as limits

on joints positions, velocities, accelerations and torques. The technique based on Euler-Lagrange conditions was also demonstrated being weak in finding Pareto-optimal sets, as the optimization of multiple performance indices at the same time can only be done through the usage of weights [44, 12]. Also, a technique to consider joint limits has been studied in [45], where state space augmentation is used together with the Pontryagin's maximum principle. However, the latter results in a rather heavy mathematical formulation, whose scalability with respect to a higher number of redundant DOFs and additional constraints remains uncertain.

In order to overcome such issues and to then accommodate a multitude of constraints and objective functions, dynamic programming could be used instead, leading to problems that are usually demanding in terms of computational resources (mainly memory and time), but much more flexible in addressing the needs arising from real applications. Moreover, dynamic programming algorithms perform better as the number of constraints increases and adjustments can be made to achieve an acceptable compromise between computation time and accuracy of the solution.

In Section 2.4, we have seen that techniques aiming at finding the globally-optimal solutions only provide necessary conditions for optimality, as it is the case of the Euler-Lagrange conditions or Pontryagin's maximum principle. More complex algorithms [26] that are deeply based on the topological characteristics of the inverse kinematic mapping can be more accurate in finding the globally-optimal solution. However, in practice, they only work for systems with one degree of redundancy and are not flexible with respect to the accommodation of arbitrary constraints. Also, their computational complexity is dependent on the number of $\mathcal{C}$-path-homotopy classes. In [46] a numeric approach is proposed, where the unknown joint space position curves are discretized and optimized with the Newton method. The technique is general enough to accommodate cases where the constraint is not necessarily a workspace path, but, more generally, a sub-space of the workspace (e.g. a given surface). However, it is prone to sub-optimal

solutions, as the performances strictly depend on a starting guess, used to initialized the solver. With dynamic programming, we aim at designing an algorithm that guarantees the achievement of the global optimum, or at least, that is very likely to achieve it, according to a resolution-optimal paradigm. At the same time, the procedure should be simple enough to be easily applied to systems with multiple degrees of redundancy.

In Section 2.7.1, we recall the theoretical basics of dynamic programming. In Section 2.7.2, we argue about suitable parametrizations of the manipulator's kinematics, using some of the concepts presented in Section 2.5.2. Topological analyses of the solutions space will support the creation of the dynamic programming grids (see Section 2.7.3.2) which can be combined together with the algorithm presented in Section 2.7.4.1.

Again with the same use case as Section 2.7.5.1, it is demonstrated, in Section 2.7.5.2, that dynamic programming can efficiently seek the global optimum without solving the BVP. The use case is also analyzed from the topological point of view in Section 2.7.5.3. The conclusions of such analyses pave the way, in Section 2.7.5.4, to the possibility of using multiple DP grids to enable the online reconfiguration of the manipulator while the assigned trajectory is tracked.

Such a result is of primary importance as it considerably expands the flexibility of planning techniques for redundant manipulators operating in constrained environments, while preserving the global optimality of the solution.

Lastly, in Section 2.7.6, recalling the analogy between local and global solutions discussed in Section 2.4.6, we build a different use case where the manipulator is obliged to change its posture to keep the cost function at its minimum. We observe that solving the BVP does not lead to any acceptable solution, while the global minimum is achieved with dynamic programming.

2.7.1 Generic formulation

Although a continuous time formulation of the dynamic programming problem is possible, this dissertation is limited to the discrete time systems, as the objective here is to propose a solution that can be directly implemented on digital hardware. To this purpose, assume to discretize the interval $[0, \Lambda]$ such that $\lambda = i\tau$, where τ is the sampling interval, $i = 0, 1, 2..N_i$ and $N_i = \frac{\Lambda}{\tau}$.

Thus, the following discrete system is given with its initial conditions:

$$\mathbf{q}(i+1) = f\big(\mathbf{q}(i), \mathbf{u}(i)\big), \quad \mathbf{q}(0) = \mathbf{q}_0 \tag{2.119}$$

where $\mathbf{q}$ represents the state vector of the system, and $\mathbf{u}$ is the input vector. The objective is to find the optimal sequence of inputs that minimizes or maximizes a given cost function defined, in general, on both the state and input vectors and their derivatives.

Usually, $\mathbf{u}$ is not free, but constrained to belong to a certain domain $\mathcal{A}_i$, which, as its subscript suggests, may change at each value of the stage variable. In many real applications, where the input represents some physical quantity (e.g. velocities, torques), its derivative may also be limited to a given domain $\mathcal{B}_i(\mathbf{u}(i))$, which is, in principle, λ-variant as well as input-variant, such as, for instance, when the acceleration that an electric motor can produce is dependent on the motor velocity. We can then write:

$$\begin{aligned} \mathbf{u}(i) &\in \mathcal{A}_i \\ \mathbf{u}'(i) &\in \mathcal{B}_i(\mathbf{u}(i)) \end{aligned} \tag{2.120}$$

Since $\mathbf{u}$ is only defined at each i, its derivative can be defined using the Euler approximation, that is

$$\mathbf{u}'(i) = \frac{\mathbf{u}(i+1) - \mathbf{u}(i)}{\tau} \tag{2.121}$$

Thus, at each i, the set of admissible values of $\mathbf{u}(i)$, from which it is possible to reach $\mathbf{u}(i+1)$ is given by the intersection between $\mathcal{A}_i$

and the set of $\mathbf{u}$-values respecting the constraint on the derivative, that is

$$\mathcal{C}_i = \mathcal{A}_i \cap \left\{ \mathbf{u}(i) : \frac{\mathbf{u}(i+1) - \mathbf{u}(i)}{\tau} \in \mathcal{B}_i(\mathbf{u}(i)), \text{ with } \mathbf{u}(i+1) \in \mathcal{A}_{i+1} \right\} \tag{2.122}$$

Now that the dynamic system is fully defined, together with the domain of $\mathbf{u}$, we can introduce the objective function to optimize, that is

$$I(0) = \psi\big(\mathbf{q}(N_i)\big) + \sum_{j=0}^{N_i-1} l\big(\mathbf{q}(j), \mathbf{q}'(j), \mathbf{u}(j), \mathbf{u}'(j)\big) \tag{2.123}$$

where the assumption was made that the cost function computed locally l only depends on the states, on the inputs and on their first-order derivatives, but in general, more complex functions could be defined. If the Euler approximation is used for both $\mathbf{q}'$ and $\mathbf{u}'$, once τ is given, the cost function can be rewritten as:

$$I(0) = \psi\big(\mathbf{q}(N_i)\big) + \sum_{j=0}^{N_i-1} l\big(\mathbf{q}(j), \mathbf{q}(j+1), \mathbf{u}(j), \mathbf{u}(j+1)\big) \tag{2.124}$$

At a generic stage i, the objective function can be written as:

$$I(i) = \psi\big(\mathbf{q}(N_i)\big) + \sum_{j=i}^{N_i-1} l\big(\mathbf{q}(j), \mathbf{q}(j+1), \mathbf{u}(j), \mathbf{u}(j+1)\big) \tag{2.125}$$

or, otherwise, in a recursive form:

$$\begin{aligned} I(N_i) &= \psi\big(\mathbf{q}(N_i)\big) \\ I(i) &= I(i+1) + l\big(\mathbf{q}(i), \mathbf{q}(i+1), \mathbf{u}(i), \mathbf{u}(i+1)\big) \end{aligned} \tag{2.126}$$

Assume that the optimization criterion is to minimize $I(0)$. By using the *Bellman* principle, we could then write:

$$\begin{aligned} I(N_i) &= \psi\big(\mathbf{q}(N_i)\big) \\ I_{opt}(i) &= \min_{\mathbf{u} \in \mathcal{C}_i} \Big[l\big(\mathbf{q}(i), \mathbf{q}(i+1), \mathbf{u}(i), \mathbf{u}(i+1)\big) + I(i+1) \Big] \end{aligned} \tag{2.127}$$

While $I_{opt}(0)$ represents the optimized function, the same function $I_{opt}(i)$ at a generic stage i is also called *optimal return function* and corresponds to the minimum value of the objective function if the process started at the stage i. The first equation is necessary to initialize the recursion. If multiple final states are admissible, $\mathbf{q}(N_i)$ represents a variable configuration. In principle, a recursion should be instantiated for each possible final state. A common solution to manage, in practice, an undetermined final state, is to define a mock state at stage $N_i + 1$ and embed the cost $\psi\big(\mathbf{q}(N_i)\big)$ inside the local cost function l between stages N_i and $N_i + 1$. In this case, the recursion is initialized with $I(N_i + 1) = 0$.

2.7.2 Redundancy parametrization

The first issue to address when finding the optimal joint-space trajectory for a redundant manipulator by using dynamic programming is to establish to which quantity the input vector $\mathbf{u}$ in equation (2.119) corresponds. The choice has to take into account the algorithmic implementation of (2.127), which usually corresponds to a search on a multi-dimensional grid built with discrete values of λ and $\mathbf{u}$.

One possibility could be to use the discrete form of equation (2.6), given by

$$\mathbf{q}(i+1) = \mathbf{q}(i) + \tau\Big\{\mathbf{J}^{\dagger}(\mathbf{q})\mathbf{x}'(i) + \big[\mathbf{I} - \mathbf{J}^{\dagger}(\mathbf{q})\mathbf{J}(\mathbf{q})\big]\,\boldsymbol{\phi}'(i)\Big\} \quad (2.128)$$

and to set $\mathbf{u} = \boldsymbol{\phi}'$, as $\boldsymbol{\phi}'$ are the inputs that we could use to condition the motion in the null space of the Jacobian, without violating the kinematic constraints. However, this choice would imply that $\mathbf{u} \in \Re^n$, equivalent to seeking the optimal return function in an n-dimensional space, which is certainly not a viable solution from the implementation standpoint.

Alternatively, one could think of replacing the null space projector in (2.6) with a null space basis, reducing the search space to $n-m$

(i.e. number of redundancy degrees). In this case the problem becomes manageable in terms of local minimization/maximization over $\mathbf{u}$, but the space of the solutions will uncontrollably grow with the number of λ-steps. For instance, if we ran the dynamic programming algorithm for N_i steps and, at each step, p values of $\mathbf{u}$ were admissible, the algorithm should evaluate N_i^p solutions.

Indeed, this is a known issue in dynamic programming implementations, which usually suggests to discretize the state space, instead of the input space, if possible. Inputs are then selected from a continuous set which allow the system to evolve from one discrete state on the grid to the next.

The considerations about the minimum number of differential equations made in Section 2.5.2 suggest that the redundancy could also be parametrized in a way that the input $\mathbf{u}$ corresponds to one of the joints positions, on the basis of which the other joints position could be computed at each i. Alternatively, with a more generic formulation, $\mathbf{u}$ could correspond to a joint combination. As in Section 2.5.2, analyses have to be made on either the reduced Jacobian or the augmented Jacobian to ensure they do not become singular for any point on the path.

As far as joint selection method is concerned, all the considerations made in Section 2.5.2.1 apply here as well. Once the trajectory is assigned, this solution leads to the following dynamic system:

$$\mathbf{q}(i+1) = f\big(\mathbf{q}_r(i), \mathbf{q}_u(i)\big), \quad \mathbf{q}(0) = \mathbf{q}_0 \tag{2.129}$$

where $\mathbf{q}_u$ and $\mathbf{q}_r$ are the vector of joints selected to parametrize redundancy and the vector of the remaining joints respectively.

Rather, if a joint combination is chosen, the dynamic system (2.119) becomes:

$$\mathbf{q}(i+1) = f\big(\mathbf{q}(i), \mathbf{v}(i)\big), \quad \mathbf{q}(0) = \mathbf{q}_0 \tag{2.130}$$

Although initial conditions are given in both (2.129) and (2.130), it will be shown that they can be computed as part of the solution, accounting for the case in which natural boundary conditions apply.

In the remainder of this dissertation, we will assume to choose a parametrization based on joint selection, so that $\mathbf{q}_u$ and $\mathbf{u}$ can be used interchangeably.

2.7.3 Considerations on implementation

2.7.3.1 *Forward* vs *backward* implementation

As far as the implementation is concerned, anytime the grid contains the discrete states of the system, the algorithm is limited to a search over such a grid. This makes it possible to choose between a recursive and an iterative approach with the usage of a return function or without and between a *forward* (i.e. from $\lambda = 0$ to $\lambda = \Lambda$) and a *backward* (i.e. from $\lambda = \Lambda$ to $\lambda = 0$) implementation. It is worth remarking that, in this case, a solution can be found regardless of the Bellman optimality principle, which would rather be necessary if the states set was continuous. In fact, it is possible to give a formulation of the problem in terms of graph theory [47], from which it is evident that the search space is an acyclic directed graph and the so-called dynamic programming algorithm is, in truth, an optimal path search algorithm.

It has to be noted that if a $forward$ implementation was chosen, equation (2.127) has to be rewritten as

$$\begin{aligned} I(0) =& \psi\big(\mathbf{q}(0)\big) \\ I_{opt}(i) =& \min_{\mathbf{u}_{i-1} \in \mathcal{C}_{i-1}} \Big[I(i-1) + l\big(\mathbf{q}(i), \mathbf{q}(i-1), \mathbf{u}(i), \mathbf{u}(i-1)\big) \Big] \end{aligned} \tag{2.131}$$

In this case it is convenient to redefine $\mathbf{u}'(i)$ as:

$$\mathbf{u}'(i) = \frac{\mathbf{u}(i) - \mathbf{u}(i-1)}{\tau} \tag{2.132}$$

and $\mathcal{C}_{i-1}$ as:

$$\mathcal{C}_{i-1} = \mathcal{A}_{i-1} \cap \left\{ \mathbf{u}(i-1) : \frac{\mathbf{u}(i) - \mathbf{u}(i-1)}{\tau} \in \mathcal{B}_i(\mathbf{u}(i)), \right. \\ \left. \text{with } \mathbf{u}(i) \in \mathcal{A}_i \right\} \tag{2.133}$$

The choice between *forward* and *backward* implementation is not, in general, arbitrary, as it often depends on considerations about performance and on the hardware architecture used, as well as on the boundary conditions of the problem.

In order to better understand how boundary conditions affect the choice, we could highlight that the optimum cost function $I_{opt}(0)$ (*backward*) or $I_{opt}(N_i)$ (*forward*) are conditioned by the sequence of inputs enabled $\mathcal{A}_i$ at each i. In many practical cases, unless the environment in which the robot moves is particularly constrained, applications require that either the initial joints positions or the final ones or both are assigned or constrained to belong to a certain subset of the domain. The optimal solution and the value of the cost function will then vary together with the initial or final set of inputs. So we may write I_{opt} as a function of such sets [44], having $I_{opt}(0, \mathcal{A}_{N_i})$ or $I_{opt}(N_i, \mathcal{A}_0)$.

Assume to run our *forward* dynamic programming algorithm once, starting with inputs in $\mathcal{A}_0$ and ending with inputs in $\mathcal{A}_{N_i}$. One execution of the algorithm is sufficient to provide the solution together with its cost for the optimum joint-space paths ending in each single element of $\mathcal{A}_{N_i}$. From the practical standpoint, the upside is that one may decide to select a sub-optimal solution if its cost does not vary too much from the optimal cost, while the final joints position is much more favorable for the particular task the robot has to execute.

On the other hand, if one asked for a solution starting from a specific $\mathbf{u}(0)$, this may require an additional execution of the algorithm either proceeding *backward* or by explicitly forcing the

initial condition at the moment $\mathcal{A}_0$ is defined. In other words, one execution of the *forward* algorithm with free initial conditions does not guarantee the computation of a solution for each input in $\mathcal{A}_0$, as well as one execution of the *backward* algorithm with free final conditions does not guarantee the computation of a solution for each input in $\mathcal{A}_{N_i}$.

2.7.3.2 Grid computation

As discussed above, dynamic programming algorithms for redundant manipulators are particularly efficient when $\mathbf{u}$ is taken from the state vector or when, like in the case of a joint combination, directly determines the state vector. In both cases, the efficiency can be further improved by pre-computing the grid, as once a point on the path and the "redundant" joints positions $\mathbf{q}_u$ (or joint combination parameter $\mathbf{v}$) are given, all joints positions can be computed from either equation (2.85) or equation (2.106). For the sake of simplifying the notation, the assumption that $\mathbf{u}(i) = u(i) \in \mathcal{A}_i \subset \Re$ is made, implying that the grid will be two-dimensional.

First, one needs to discretize the input domain, by selecting, for instance, N_u equally spaced values of u, such that $\mathcal{A}_i$ is a discrete set for each i and $u_j(i)$ is the j-th element of $\mathcal{A}_i$, with $j = 1..N_u$ and $i = 1..N_i$.

Then, the joints positions can be computed from $u_j(i)$. If an analytic form of the inverse kinematics exists, parametrized with respect to the redundancy parameter, the joints positions can be computed from the algebraic expression. If an analytic solution does not exist or is hard to obtain, numeric solvers can be used.

One thing to keep in mind though, is that, in general, equations (2.85) and (2.106) admit a finite set of solutions, but not a unique one. While this is immediate to verify when an analytic expression of the inverse kinematics is available, it might not be as evident when using numeric solvers, which do not invert the kinematics

relations explicitly. Even for non-redundant manipulators, as it is the case of a simple two-link planar manipulator, the inverse kinematics solution of a Cartesian point in its workspace is notoriously made of two configurations, commonly known as *elbow-up* and *elbow-down*. This suggests that multiple dynamic programming grids need to be considered at the same time if the solution has to be found across the whole configurations space.

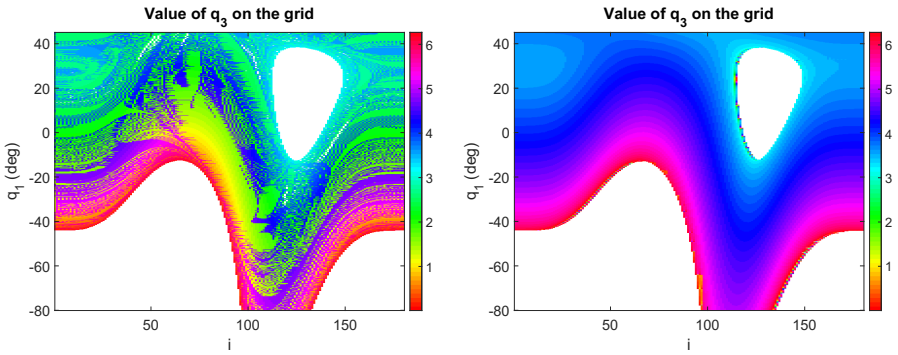
In the following, we will use the term *posture* to refer to subspaces of the joints space into which it is possible to find one and only one solution to the inverse kinematics problem when $\mathbf{q}_u$ (or $\mathbf{v}$) is given. If the manipulator is of type 1 [38], the terms *posture* and *extended aspect* can be used interchangeably. The reader must be aware that there is a slight difference between the *posture* just defined and the concept of *self-motion manifold* recalled in Section 2.6. In facts, two or more postures may close in one manifold if all configurations in such spaces can be reached with a continuous motion of the joints, which, as discussed in Section 2.6, is a condition depending on where the end-effector is in the workspace.

However, regardless of the self-motion manifold they belong to, the farther the manipulator or its subchains are from a singularity, the longer is the distance, in the configuration space, between solutions belonging to different postures. An arbitrary choice between one solution or the other may cause *jumps* in the grid, which are likely to violate the derivative constraints, affecting the search of the optimal solution.

We then say that the grids have to be homogeneous (i.e. solutions have to be continuous) for the dynamic programming algorithm to provide the optimum in that specific posture.

Some grids examples are provided in Figure 2.11. On the x-axis is $i = 1..N_i$, with $N_i = 180$. On the y-axis are the values $u_j = q_{1,j}$, obtained by discretizing the input space in the interval $[-80, 45]$ degrees to the purpose of displaying the features of interest. The color bars placed next to the figures associate colors to joint positions in $[0, 2\pi]$. The colors in the cells refer to one of the joint

positions (q_3 in this case) computed through inverse kinematics, given, for each cell, $q_{1,j}$ and $\mathbf{x}(i)$. Pronounced colors differences between adjacent cells suggest that the joints positions associated with them are far in the configuration space and are likely to belong to different postures.



(a) (b)

Figure 2.11: Examples of non-homogeneous (a) and homogeneous (b) grids in q_3 where $u = q_1$. White areas represent regions where no solution exists for the given $(u_j, \mathbf{x}(i))$ pair

In the grid of Figure 2.11(a), solutions have been obtained through numerical solving procedures, which usually stop when the first solution is found. As there is no guarantee that solutions always belong to the same posture, non-homogeneous grids, as in Figure 2.11(a) are likely to be generated.

Numeric solvers also suffer from bad performances when no solution exists. Depending on actual implementations, they may try to seek the solution starting from different initial conditions, which is time consuming. It is the case of the white areas in Figure 2.11(a).

One way to improve efficiency and to ensure obtaining homogeneous solutions is to build the grid row by row (i.e. fixing one of the joints at $u_j(i)$) and to compute the initial condition in the desired posture by the means of a numeric solver, starting from which all the other values on the same row are computed by nu-

merical integration using the inverse of the Jacobian. The grid of Figure 2.11(b) has been computed with this method. As the reader may notice, the colors smoothly change from light blue, i.e. about 3 radians, to red, i.e. 2π radians, which is an indication that continuity (in the joint space) is guaranteed between adjacent cells of the grid. In several previous works, such as [39] and [40], homogeneous grids have also been referred to as *feasibility maps*, as they can be exploited to study the feasibility of trajectories in environments with obstacles or in presence of joint limits.

Using the terminology adopted in [34], the number of grids to be generated N_g only depends on the mechanical characteristics of the manipulator and is equivalent to the maximum number of self-motion manifolds generated in the configuration space as pre-images of a generic point in the task space. This means that, regardless of where the end-effector is and regardless of how many degrees of redundancy the manipulator has, the maximum number of grids is constant and is 2,2,4 and 16 for planar, spherical, regional and spatial manipulators respectively [34]. However, it is worth remarking that, for some specific kinematic structures, as well as for some specific trajectories [38], the actual number of grids could be less than the maximum theoretical value. For instance, for most six-axis industrial manipulators, the number of different configurations is equal to 8 [47]. An example of two homogeneous grids, representing two different postures for the same manipulator and trajectory is provided with the case study of Section 2.7.5.2.

Homogeneous grids may also be obtained by using the topological notion of extended aspects discussed in Section 2.6.4. If the manipulator is planar or spherical, meaning that only two inverse kinematic solutions exist for a given $\mathbf{x}$, once the redundancy parameter $\mathbf{v}$ is defined, the determinant of the augmented Jacobian can be tested, such that one grid contains all the solutions satisfying $|\mathbf{J}_a| > 0$ and the other all the solutions satisfying $|\mathbf{J}_a| < 0$. If a numeric solver is employed, one condition or the other could be used to reduce the search space and guarantee that the solu-

tion returned by the solver is in the desired posture. In case the manipulator is more complex, but still of type 1 [38], a possibility exists to create homogeneous grids if the augmented Jacobian can be easily factorized. In all the other cases, the augmented Jacobian singularities only provide one condition to distinguish between solutions, thus additional criteria have to be found by testing either the joint variables (e.g. shoulder, elbow, wrist), or the minors determinants. An example is provided in [48], with application to parallel robots, and with the case study of Section 2.8, with a 7-DOF manipulator.

2.7.4 Dynamic programming algorithm

2.7.4.1 Single grid and multi-grid algorithms

As discussed in Section 2.7.3.1, several implementations are possible. In this section, for the sake of brevity, only a *forward* iterative algorithm is presented for the case $n - m = 1$.

First assume to work with each of the grids separately, which means that as many algorithm executions as the number of grids are necessary to find the global minimum. Also, if grids are not used together, the motion is kept confined within a certain posture, i.e. passes from one configuration to the other are not possible. Under this assumption, the pseudo-code of the algorithm is as in Algorithm 1.

Sets $\mathcal{A}_i$ at step 1 could be given in terms of inequalities, e.g. $-1.5 < u < 1.5$ or as boolean maps when it is necessary to select specific values from the discrete set. When the knowledge of the input implies the knowledge of the state, as shown in Section 2.7.2, the definition of $\mathcal{A}_i$ could be extended to include all the constraints given on the state. This way, $\mathcal{A}_i$ could be used to accommodate any joint limit or more complex, usually application dependent configuration constraints, such as the distance from obstacles or singularities [35]. Similar considerations can be done for the initialization of $\mathcal{B}_i$ at step 2 to accommodate joint rates limits or

Algorithm 1 *Forward* iterative dynamic programming algorithm with single grid

Initialize $\mathcal{A}_i, \forall i = 0..N_i$
Initialize $\mathcal{B}_i, \forall i = 1..N_i$
Initialize $\mathcal{C}_i = \varnothing, \forall i = 0..N_i$
Initialize cost map $I_{i,j} = +\infty, \forall i = 1..N_i, \forall j = 0..(N_u - 1)$
Initialize cost map $I_{0,j}, \forall j = 0..(N_u - 1)$ *with the initial cost*
$\mathcal{C}_0 \leftarrow \mathcal{A}_0$
for $i \leftarrow 0$ to $N_i - 1$ **do**
    for each $u_j \in \mathcal{C}_i$ **do**
        for each $u_k \in \mathcal{A}_{i+1}$ **do**
            $u' \leftarrow \frac{u_k - u_j}{\tau}$
            if $u' \in \mathcal{B}_{i+1}$ **then**
                $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_{i+1} + \{u_k\}$
                Compute instantaneous cost function l
                if $I_{i,j} + l < I_{i+1,k}$ **then**
                $I_{i+1,k} = I_{i,j} + l$
                Let u_j *at stage* i *be the predecessor of* u_k *at stage* $i + 1$
$I_{opt}(N_i) = \min_j I_{N_i,j}$
Build function $u(i)$ *of optimal inputs by screening the predecessors map backward*

additional velocity-dependent constraints. It is worth remarking that $\mathcal{A}_i$ and $\mathcal{B}_i$ are stage-dependent to account for the most general case, but, if only joint position and velocity limits are given, they are both constant along the motion and their dependence on i can be omitted.

A more flexible algorithm could be designed to visit all the grids at the same time. This way, it is possible for the manipulator to smoothly pass from one posture to the other, increasing dexterity for particularly complex tasks. It has to be noted that the passage from one posture to the other can only happen in those regions of the workspace ($\mathcal{W}$-covers, according to terminology in [34]) where the self-motion manifolds form a continuous curve in the joint space. At most, they can happen on the borders of such $\mathcal{W}$-covers, in which case though the manipulator will encounter a singularity. Rather, when the manipulator reconfigures its posture inside a $\mathcal{W}$-cover where it can, only a subset of its joints will align, bringing the relating kinematic subchain to a singularity. However, from the practical standpoint, this does not represent an issue for the majority of real applications. An example of us-

age of multiple grids, although not related to optimization and dynamic programming, can be found in [48], where an algorithm is designed for parallel manipulators to visit multiple extended aspects, therein termed *working modes*.

Even in the case of multiple grids, several implementations are possible. Which one to adopt is usually dependent on a time-memory trade-off. In facts, some implementations may increase the size of the maps kept in memory but be faster when it comes to execution time. Rather, others may just compute some constraints in-line, saving memory to the detriment of time. Assuming that, for instance, one would like to privilege memory over time, a possible implementation could be as in Algorithm 2.

Algorithm 2 *Forward* iterative dynamic programming algorithm with multiple grids

Initialize $\mathcal{A}_i, \forall i = 0..N_i$
Initialize $\mathcal{B}_i, \forall i = 1..N_i$
Initialize $\mathcal{C}_i = \varnothing, \forall i = 0..N_i$
Initialize cost map $I_{i,j,g} = +\infty, \forall i = 1..N_i, \forall j = 0..(N_u - 1), \forall g = 1..N_g$
Initialize cost map $I_{0,j,g}, \forall j = 0..(N_u - 1), \forall g = 1..N_g$ *with the initial cost*
$\mathcal{C}_0 \leftarrow \mathcal{A}_0$
for $i \leftarrow 0$ to $N_i - 1$ **do**
    for each $u_j \in \mathcal{C}_i$ **do**
        for each $\mathbf{q}_g$ within joints positions limits **do**
            for each $u_k \in \mathcal{A}_{i+1}$ **do**
                for each $\mathbf{q}_h$ within joints positions limits **do**
                    $u' \leftarrow \frac{u_k - u_j}{\tau}$
                    if $u' \in \mathcal{B}_{i+1}$ **then**
                        $\mathbf{q}' \leftarrow \frac{\mathbf{q}_h - \mathbf{q}_g}{\tau}$
                        if $\mathbf{q}'$ is such that $\dot{\mathbf{q}}$ is within joints velocity limits **then**
                            $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_{i+1} + \{u_k\}$
                            Compute instantaneous cost function l
                            if $I_{i,j,g} + l < I_{i+1,k,h}$ **then**
                                $I_{i+1,k,h} = I_{i,j,g} + l$
                                Let u_j, $\mathbf{q}_g$ *at* i *be the predecessors of* u_k, $\mathbf{q}_h$ *at* $i + 1$
$I_{opt}(N_i) = \min_{j,g} I_{N_i,j,g}$
Build functions $u(i)$ *and* $\mathbf{q}(i)$ *of optimal inputs and states by screening the predecessors map backward*

For the algorithm above, the following assumptions have been made:

- $\mathcal{A}_i$ and $\mathcal{B}_i$ only account for constraints defined on the input

variable, not for those defined on state variables. However, if u is taken from the state vector, the checks on $\mathcal{A}_i$, $\mathcal{B}_i$ and $\mathcal{C}_i$ can be skipped.

- Constraints on the joints positions and velocities are only given in terms of constant upper and lower bounds.

By removing the assumptions above, more complex algorithms could be designed, which are not discussed here for the sake of brevity.

Anyhow, whichever the implementation details are, if all grids are visited at the same time, all the possible $\mathcal{C}$-path-homotopy classes will be available to the algorithm, which can find the optimal solution across them. At any time, the manipulator will be able to get close to and pass through singularities of any of its kinematic subchains, overcoming the limits of the solutions based on the Euler-Lagrange formulation or Pontryagin's maximum principle. We will clarify these aspects with the example of Section 2.7.5.

2.7.4.2 Additional performance indices

If the algorithm is implemented as described in the previous section, it is clear that a multitude of performance indices can be accommodated in addition to those mentioned in Section 2.4.5, such as [35]

- *maximum energy*

$$G = \max_t \left\{ \dot{\mathbf{q}}^T \mathbf{W} \dot{\mathbf{q}} \right\} \tag{2.134}$$

- *maximum of inverse manipulability*

$$G = \max_t \left\{ \frac{1}{|\mathbf{J}\mathbf{J}^T|} \right\} \tag{2.135}$$

Moreover, vector performance indices of the form $\mathbf{G} = [G_1, G_2, ..., G_n]^T$ can be considered. For instance, in laser cutting applications, some common indices of this kind are [35] (k is the joint index in the joint position vector):

- *joint coordinate range*

$$G_k = \max_t \{q_k(t)\} - \min_t \{q_k(t)\} \tag{2.136}$$

- *joint coordinate displacement*

$$G_k = \int_0^\tau |\dot{q}_k(t)| dt \tag{2.137}$$

- *joint maximum speed*

$$G_k = \max_t \{\dot{q}_k(t)\} \tag{2.138}$$

A discrete version of these performance indices can be found in [47], where they are employed as measures of the trajectory smoothness, as discussed in Section 2.7.4.3. It is clear that, in such cases, the dynamic programming algorithm will find a Pareto-optimal solution [12].

Lastly, it is also worth mentioning the performance index proposed in [49], which, by neglecting the robot dynamics, allows to obtain the best kinematic approximation of the time-optimal solution. In a time-optimal problem, the distance in time between two stages on a DP grid can be approximated as the travel time of the slowest joint, commanded at the maximum speed, that is

$$\Delta t_i = \max_{k=1..n} \left\{ \frac{|q_k(i+1) - q_k(i)|}{\dot{q}_{k,max}} \right\} \tag{2.139}$$

Thus, the total trajectory time can be formalized as the sum of such contributions

$$T = \sum_{i=1}^{n-1} \Delta t_i \tag{2.140}$$

The reader may verify that, in this case, the velocity constraints are automatically respected, while the others have to be explicitly checked.

2.7.4.3 Feasibility and smoothness of solutions

Because the state space is discretized, it might be possible that the solution resulting from the application of the algorithms described in Section 2.7.4.1 is not feasible on real hardware. Rather, in other circumstances, it might happen that the trajectory is feasible, but it is not smooth enough to be repeated over and over again without damaging the mechanical parts on the long run. Thus, it is interesting to look at the factors compromising either the fasibility or the smoothness of the solution and understand what can be done to address such issues.

At the beginning of Section 2.7, we have argued that dynamic programming is much more flexible in accommodating additional constraints than those we can manage with a reasonably simple mathematical formulation based on calculus of variations. In Section 2.7.1, we introduced a rather generic formalism to handle position and velocity constraints through the sets $\mathcal{A}_i$ and $\mathcal{B}_i$, and, in Section 2.7.4.1, we discussed about the usage of such sets. The proposed formulation is straightforward, but, in practice, is not enough to ensure that the motion is always feasible and smooth. In fact, on one hand, the output joint trajectory could exceed joint torque capacities and, on the other, could result in oscillations of the joints because of its non-smoothness.

Hence, in order to achieve both feasible and smooth motions, the joint torques should be taken into account. This requires a rather accurate dynamic model, which considerably complicates the implementation of the dynamic programming algorithm and is not always available in real industrial applications. Thus, a practical solution consists in transforming torque constraints into acceleration constraints, which are usually provided in the robots datasheets [47]. This allows to keep the same framework as Sec-

tion 2.7.4.1, ensuring that the complexity of the algorithm does not grow too much, with evident consequences on the execution time.

On the other hand, while the mentioned approximation allows for smooth joint position functions, it might not be enough to guarantee smoothness at velocity level. In such cases, it might be suitable considering additional constraints on the derivative of the acceleration, which could also be provided in the robots datasheets [49].

An example of λ-acceleration and λ-jerk computed on the multi-grid is reported in Figure 2.12: g_1, g_2, g_3 and g_4 are four generic grid indices, i is the current waypoint index and j is the current redundancy parameter index. As indicated in Algorithm 1 and Algorithm 2, a predecessor map is created while the dynamic programming algorithm is executed, so that each reachable cell from $\lambda(0)$ to $\lambda(i)$, i.e. the current stage, has a predecessor on such a map, making up a chain starting with $\mathbf{q}_{i,j}^{(g_3)}$ and ending with $\mathbf{q}_{1,k}^{(h)}$. Thus, when a comparison is made with a cell of the next waypoint, e.g. $\mathbf{q}_{i+1,j-1}^{(g_4)}$, the λ-acceleration and λ-jerk, together with λ-velocity (step 14 of Algorithm 2), can be computed as

$$\begin{aligned} \mathbf{q}_i'' &= \frac{\mathbf{q}_{i+1}' - \mathbf{q}_i'}{\tau} \\ \mathbf{q}_i''' &= \frac{\mathbf{q}_{i+1}'' - \mathbf{q}_i''}{\tau} \end{aligned} \tag{2.141}$$

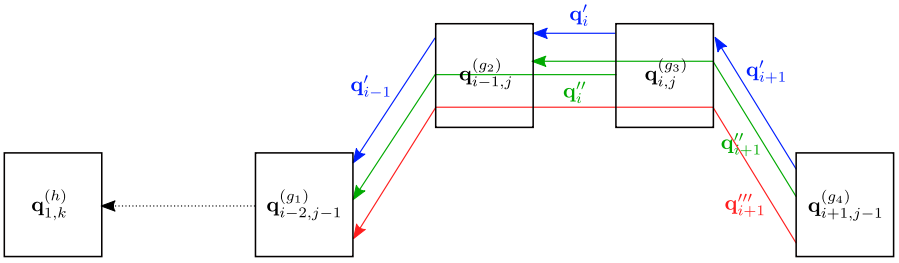


Figure 2.12: Discrete λ-acceleration and λ-jerk on a DP multi-grid

If a time law is assigned, equations (2.141) directly determine the acceleration and the jerk to pass from configuration $\mathbf{q}_{i,j}^{(g_3)}$ to configuration $\mathbf{q}_{i+1,j-1}^{(g_4)}$, which can be compared, together with joint velocity (line 15 of Algorithm 2), with the limits from the datasheet. In Figure 2.12, the colored arrows represent the cells involved in the computation of the discrete derivatives (note that line 14 in Algorithm 2 can be generalized to compute any derivative of interest): blue lines only involve two nodes for the computation of the λ-velocity, green lines involve three nodes for the computation of the λ-acceleration and the red line involves four nodes for the computation of the λ-jerk.

Together with the imposed constraints, the discretization step of the redundancy parameter also plays an important role in the generation of smooth joint space trajectories. It is clear that the finer the discretization is, the smoother the trajectory can be, but this comes to the detriment of time. Indeed, some redundancy parameters have a higher sensitivity with respect to the motion to be performed, meaning that for large changes of their value, all the other variables in play, such as the joint position variables, change less. If this is the case, a coarser discretization can be used for the redundancy parameter, as it is very representative of the motion, resulting in a smooth trajectory, still at a reasonable computation time. Alternatively, an iterative approach can be used, where a finer discretization is performed in the neighborhood of a solution obtained with a coarser discretization at the previous iteration [49]. This technique yields satisfactory results, but may compromise the optimality of the solution if the first discretization is too coarse.

In some other works, such as in [35] and [47], the trajectory smoothness has also been explicitly included in the performance index to optimize. For instance, the performance indices (2.136)-(2.138) can be used as indirect measures of smoothness. They can be suitably combined with other performance indices of interest for the specific application, but the result will always be a sub-optimal solution with respect to each of the indices.

In [49], a different approach is considered, which is based on the post-processing of the solution. In particular, the redundancy parameter curve is smoothened by applying a fifth-order polynomial approximation. Then, in order to guarantee that the trajectory is exactly tracked, inverse kinematics is solved again with the new values of the redundancy parameter. In the numerical approach of [46], the proposal also is to interpolate in post-processing, but all joint position curves are interpolated at the same time in an iterative Newton-Rapshon-like root-finding algorithm that aims at minimizing the distance with the assigned workspace path. Every time the control input is post-processed before being sent to the robot controller, the constraints that have been enforced by the dynamic programming algorithm might be violated between two interpolation points. Although applied to the domain of time-optimal control that we will address in Chapter 3, the solution proposed in [14] foresees to perform the mentioned interpolation between the grid nodes as opposed to post-processing. The interpolation is embedded in the constraints checking step of the dynamic programming algorithm so that unfeasible profiles can be directly excluded, thus ensuring that the generated trajectory will be eventually smooth and feasible on real hardware without further modifications.

As far as the feasibility is concerned, together with velocity and torque constraints, the assigned end-effector trajectory also plays an important role. Assuming that the requested end-effector velocity and acceleration are feasible at each point of the trajectory, the way the trajectory is discretized may compromise the final result. In fact, according to [35] and [47], a lower bound exists in the distance between waypoints, which is determined by the capabilities of the robot controller. In particular, for controllers using a trapezoidal velocity profile, the joining of very close waypoints yields an undesirable velocity reduction [47], thus, said $\Delta\mathbf{x}$ the distance between waypoints, $\dot{\mathbf{x}}$ the assigned velocity at the end-effector and τ_{min} the minimal duration of the acceleration/deceleration sections

in a trapezoidal profile, as per controller characteristics, it is

$$\Delta \mathbf{x} \geq |\dot{\mathbf{x}}| \tau_{min} \tag{2.142}$$

2.7.5 Example: 4R planar manipulator

The example reported here was first proposed in [24], where a 4R planar manipulator centered in the origin, with links $l_1 = 7$, $l_2 = 7$, $l_3 = 4$ and $l_4 = 1.5$, is asked to track a circular path of radius $R = 4$ centered in $(6, 4)$ at a fixed end-effector orientation of $-60°$ with respect to the x-axis. The initial and final end-effector position is $(10, 4)$, as shown in Figure 2.13, while its initial and final velocities are zero.

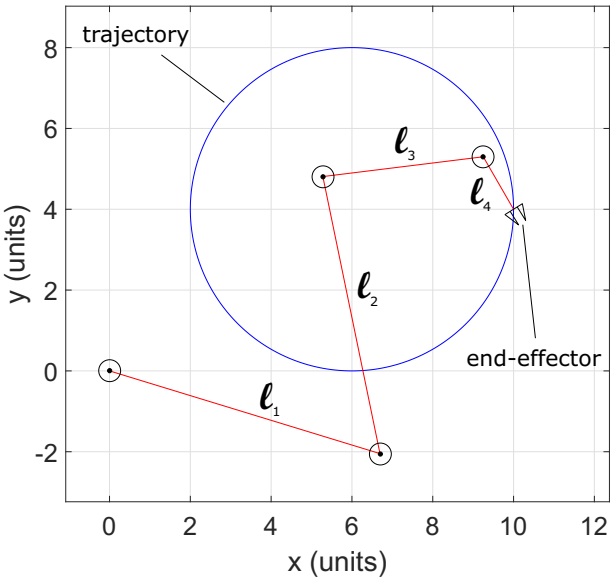


Figure 2.13: 4R planar manipulator with end-effector at its initial/final position

Initial and final joints positions are not given, so assume that the configuration represented in Figure 2.13 is arbitrary. Under this

hypothesis, the natural boundary conditions apply, as explained in Section 2.4.2.1. The objective function to minimize is the square norm of joints velocities, thus the solution of Section 2.4.5.1 applies. Also assume that $\lambda = t$ and that $\Lambda = 9$, thus the trajectory has to be completed in 9 seconds starting from $t = 0\ s$.

End-effector acceleration profile is not given in [24], but accelerations $\ddot{\mathbf{x}} = \mathbf{x}''$ are required in (2.50). For this reason, assume they are shaped as in Figure 2.14. It is a smooth trapezoidal trajectory with a peak velocity of 1.05 rad/s. Cubic polynomials have been used for the jerk in the first and the last third of the trajectory. The profile has been selected in order to have results comparable to those from [24].

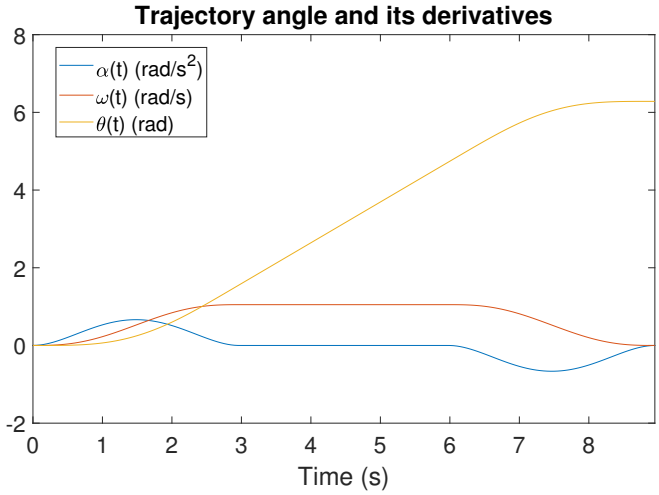


Figure 2.14: Trajectory in polar coordinates and its derivatives

As boundary conditions (2.52) apply and the boundary end-effector velocities shall be zero, boundary conditions become:

$$\dot{\mathbf{q}}(0) = 0, \quad \dot{\mathbf{q}}(9) = 0 \tag{2.143}$$

The problem made of equations (2.50) subject to boundary conditions (2.143) is a BVP in 8 equations, 8 variables ($\mathbf{q}$ and $\dot{\mathbf{q}}$) and 8

boundary conditions constraining 4 ($\dot{\mathbf{q}}$) of such variables. Half of the conditions are given at the initial time, half at the final time.

In Section 2.7.5.1, the BVP is first solved with classical numerical integration techniques. Some preliminary considerations are made about the nature of the optimal solution with respect to the whole configuration space. In Section 2.7.5.2, the same problem is solved with dynamic programming using Algorithm 1 as many times as the number of available DP grids. In Section 2.7.5.3, the optimal solutions obtained in different postures are analyzed from the topological point of view, allowing to come to some important conclusions which apply to generic manipulators. Finally, in Section 2.7.5.4, by bringing all the pieces together, we demonstrate that, with the employment of multiple grids as per Algorithm 2, it is possible to find globally optimal solutions with one scan of the DP grids. Also, we demonstrate that the solution will not be confined within some specific $\mathcal{C}$-path-homotopy class, but the manipulator will be able to smoothly reconfigure itself along the path, overcoming the limits of previous techniques.

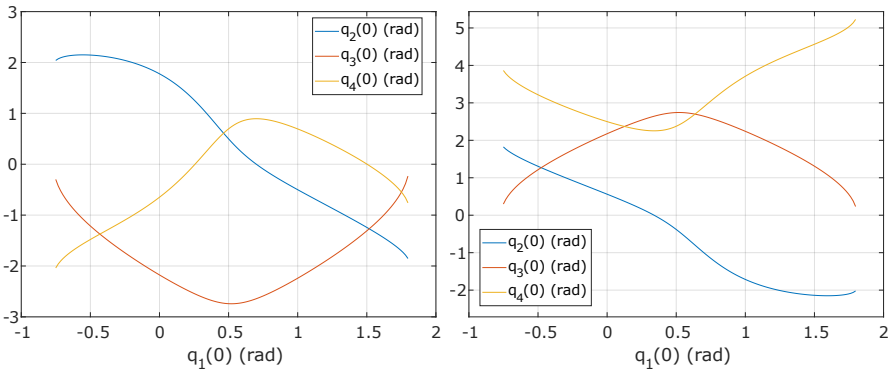
2.7.5.1 Solving as an initial value problem

As suggested in [24], one method to easily solve the problem without solving the BVP is to make a guess on $n - m$ of the joints positions ($n - m = 1$ in this case) at the initial time, such that initial conditions are given for both $\mathbf{q}$ and $\dot{\mathbf{q}}$ and (2.50) is solved as an initial value problem. The objective is then to find the initial joints positions leading to $\dot{\mathbf{q}}(9) = 0$ and returning the minimum cost.

While in [24] the authors assume to make several initial guesses and, from each of them, to use the Powel's optimization routine to reach the minimum, here we propose to discretize one of the joint position domains and to find the solution(s) for each value in the discrete set. Although this method results to be heavier from the computational standpoint, it allows us to draw some important conclusions about the nature of the solutions.

First of all, it is worth noting that making several guesses on $n - m$ of the initial joints positions is not in general enough to achieve the global minimum, regardless of the number of the initial guesses. In facts, for each of the guesses made on one of the joints, multiple solutions for the remaining joints, corresponding to different postures, exist. In the case of a planar manipulator, two solutions exist for non-singular points when one of the joint positions is given.

Thus, in order to ensure that the whole configuration space is explored, assume to create two 4-dimensional sets of initial joints positions. $q_1(0)$ is first fixed, then the two configurations are computed by inverting the kinematic equations. Call such configurations *elbow-left* and *elbow-right* for the sake of symmetry with the actual path. The "elbow" corresponds to the third joint. The two sets are reported in Figure 2.15, where $q_1(0)$ is used as x-axis. Notice that no solution to the kinematics equations exists when $q_1(0)$ is lower than about $-45°$ (-0.79 rad) as well as when it is greater than about $100°$ (1.75 rad).

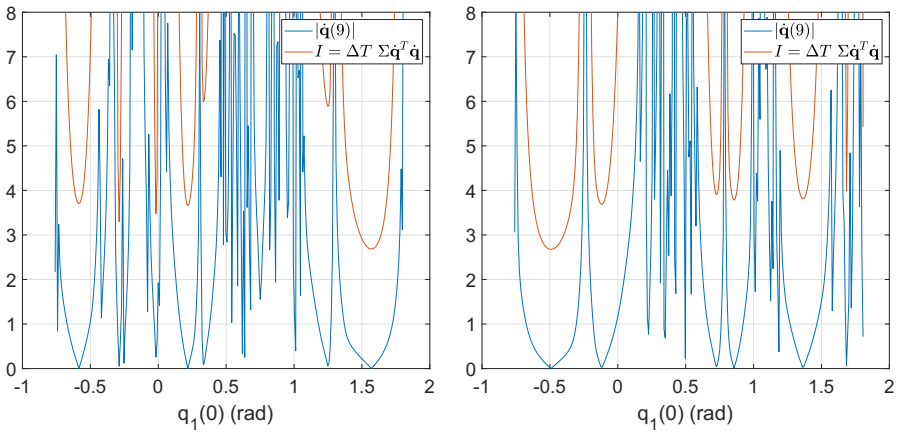


(a) (b)

Figure 2.15: Initial angles sets of 300 samples each, corresponding to elbow-left (a), i.e. negative values of q_3, and elbow-right (b) configurations, i.e. positive values of q_3

From each initial position in the sets and $\dot{\mathbf{q}} = 0$, the initial value problems are solved. The values of the final joint rates and of

the cost function, for each possible initial position, $q_1(0)$, are reported in Figure 2.16. Only the trajectories for which $\dot{\mathbf{q}}(9) = \mathbf{0}$ are solutions of the TPBVP.



(a) (b)

Figure 2.16: Final joint rates and cost function for elbow-left (a) and elbow-right (b) configurations

It is immediate to notice that several solutions exist which satisfy the Euler-Lagrange equations and boundary conditions. Each of them corresponds to a local minimum, but only one is the actual global minimum. According to [25, 26], such solutions belong to different $\mathcal{C}$-path-homotopy classes and the Euler-Lagrange conditions fail in distinguishing between them.

Thanks to this numerical procedure, it is possible to spot the global minimum for each of the configurations, occurring when $q_1 = 1.5708$ rad and $q_1 = -0.4922$ rad for the *elbow-left* and *elbow-right* configurations respectively. The values of the cost function are $I = 2.68$ and $I = 2.67$ respectively. Such results are not immediately comparable with those from [24], as no information about the trajectory profile is available. For instance, if the same trajectory profile proposed here was used, but lowering the peak velocity to 1.00 rad/s, the final cost would be $I = 2.58$, which is lower than the global minimum found in [24]. One more source of mismatch

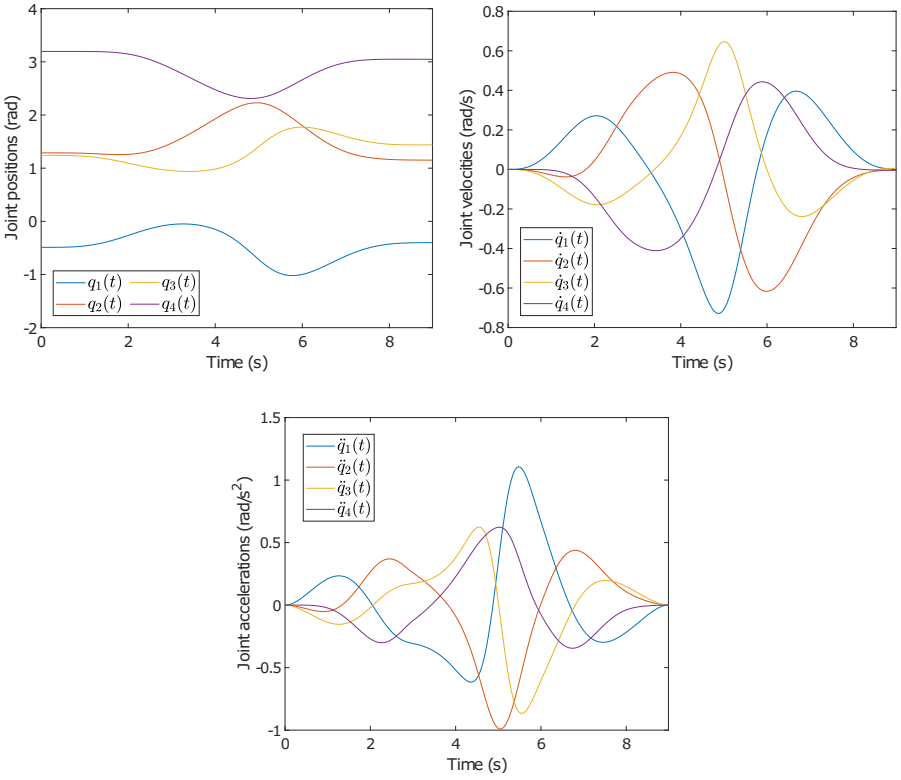


Figure 2.17: Joints positions, velocities and accelerations relating to the global minimum solution

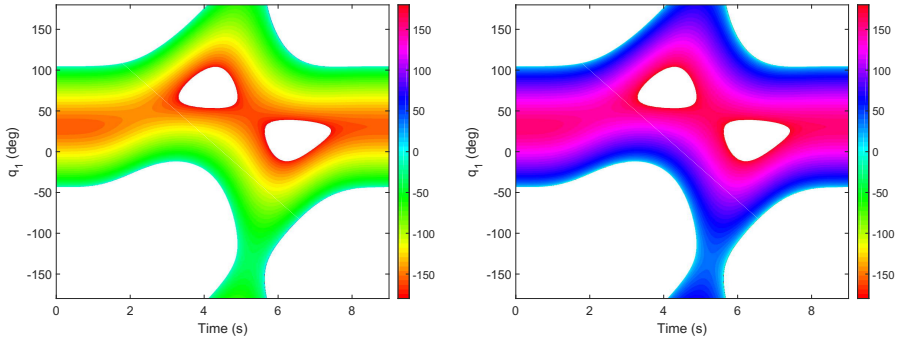
certainly is the approximation caused by the discretization of the initial state domain, which implies the computation of solutions in a neighborhood of the actual minimum.

However, from a qualitative point of view, the joints position, velocity and acceleration profiles obtained with the *elbow-right* configuration do not differ from those in [24], as Figure 2.17 demonstrates.

2.7.5.2 Solving with dynamic programming

The dynamic system (2.129) is considered, but initial conditions are not assigned, so that they can be found as part of the solution. The first joint is chosen to parametrize the redundancy.

Homogeneous state grids for both elbow configurations (postures, extended aspects) are computed as in Section 2.7.3.2 for all values of q_1 and all values of $\mathbf{x}$. Each cell in the grid contains values for q_2, q_3 and q_4, while q_1 represents the input and therefore constitutes one of the grid's axes. As an example, the "slices" of the grids relating to q_3, expressed in degrees, are reported in Figure 2.18.

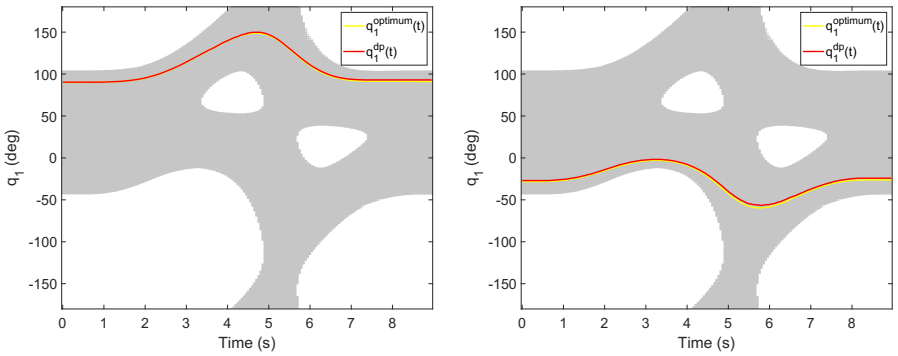


(a) (b)

Figure 2.18: State grids in q_3 (in degrees) relating to elbow-left (a) and elbow-right (b) configurations

Algorithm 1 is run on each of the grids separately to find the optimal solution for the given manipulator posture. The solutions found are reported in Figure 2.19 where they overlap to the optimal solutions found by resolving the IVPs, as discussed in Section 2.7.5.1.

In order to make the comparison with the IVP solutions, that are defined on the continuous state domain, they have been discretized using the same state grid as the dynamic programming problem. This means that, at each time step, the actual state $\mathbf{q}$ is replaced by the closest solution available on the grid. It is worth



(a) (b)

Figure 2.19: Discretized optimal solution (yellow) and dynamic programming optimal solution (red) for elbow-left (a) and elbow-right (b) configurations

remarking that such a discretization makes the IVP solutions lose their property of being optimal, affecting their cost. In particular, the cost increases from $I = 2.67$ to $I = 2.75$ for the *elbow-right* configuration and from $I = 2.68$ to $I = 2.76$ for the *elbow-left* one.

As Algorithm 1 is designed to provide the optimal solution given a certain grid, the costs mentioned above are upper bounds for the dynamic programming solution. In facts, the cost relating to the red curves in Figure 2.19 is $I = 2.73$ and $I = 2.72$ for *elbow-right* and *elbow-left* configurations respectively. According to the dynamic programming algorithm, with the given grid resolution of 0.1 degrees along the q_1 axis, the *elbow-left* configuration is slightly advantageous over the *elbow-right* one. However, given the proximity in cost between the solutions, we could conclude that the IVP and DP methods are coherent and both highlight that the two solution are equivalent from a qualitative viewpoint.

The cost is notoriously linked to the grid resolution. For instance, if we downscaled the q_1 set from 0.1 to 0.2 degrees resolution for the *elbow-left* grid, the cost would increase to $I = 2.76$. On the other hand, if the distance between samples tended to zero, both the DP and IVP-discretized solutions would tend to the optimal

solutions found in Section 2.7.5.1.

2.7.5.3 Topological analysis of solutions

Now that the optimal solutions have been found, it would be interesting to investigate their relationship from the topological point of view. The objective is threefold:

- to investigate the relationship between the solutions in terms of $\mathcal{C}$-path-homotopy classes, recalling the observations made in Section 2.6;
- to understand what happens in the joint space when a decision about the posture is taken and when, on the other hand, a passage from one posture to the other is required because of obstacles or other workspace constraints or because of certain boundary conditions that could be assigned in real scenarios;
- to identify the sub-structures of the pre-image of the workspace path with respect to the typical sub-structures described in [26].

The upper-right quadrant of the workspace is reported in Figure 2.20, and it is where the manipulator's end-effector operates. By considering the singular configurations, i.e. all links aligned in the case of a planar manipulator, $\mathcal{W}$-covers are drawn and separated by black arcs. In blue (dotted), the reader may recognize the path of Figure 2.13. In order to make the matter more tractable, assume to simplify the problem of the 4R planar manipulator with assigned position and orientation into the equivalent problem of a 3R planar manipulator with assigned position and free orientation. Therefore, the kinematic subchain made of joints 1, 2 and 3 is considered, with the end-effector assumed to be at the end of link 3. Its path in the workspace is drawn in red and corresponds to the former joint 4's path. A few points are highlighted on

such a curve. The green points in $\mathcal{W}$-covers 1 and 3, $\mathbf{x}_5$ and $\mathbf{x}_1$, are the ones where the distance between the end-effector and the manipulator's base is minimum and maximum respectively. The one in $\mathcal{W}$-cover 2 is chosen arbitrarily. The points highlighted in purple, i.e. $\mathbf{x}_2$ and $\mathbf{x}_4$, are *coregular* values (see Section 2.6) by which the end-effector transits when moving toward the base of the manipulator from its initial location.

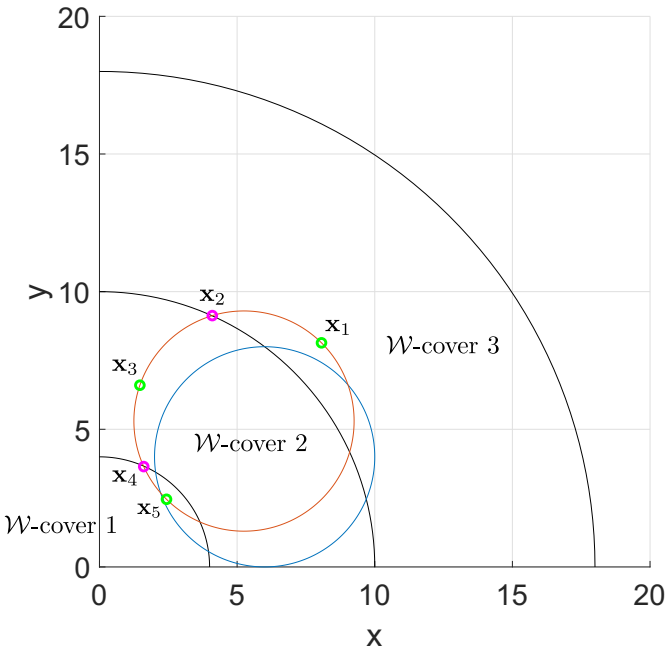
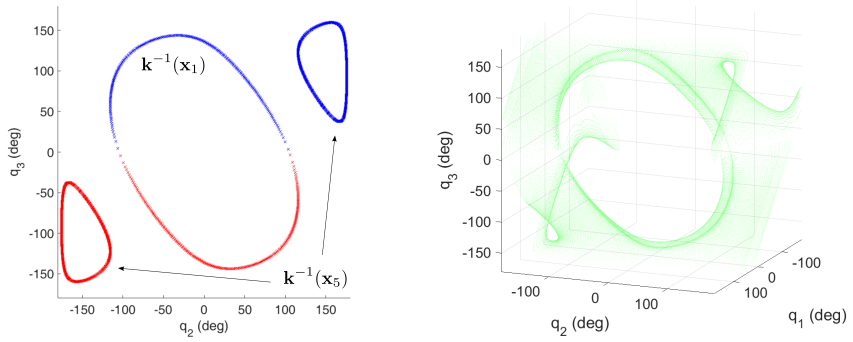


Figure 2.20: 3R manipulator's workspace divided in $\mathcal{W}$-covers; solid red line: 3R manipulator end-effector's path, dotted blue line: 4R manipulator end-effector's path

Each of the 5 points are pre-mapped onto the joint space in one or more self-motion manifolds. By definition, pre-images of *coregular* values contain singularities.

All the elements are in place to address our first objective: to investigate about the $\mathcal{C}$-path-homotopy relationship between solutions.

First, as done in [25] and [28] for similar analyses, we would like to outline the curves bounding the path in the joint space. Unlike examples reported in [25] and [28] though, where bounding manifolds close in circles (see Figure 2.8), our path crosses all the coregular value manifolds, jumping from one self-motion manifold to the other, eventually between self-motions belonging to different $\mathcal{C}$-homotopy classes (see Section 2.6.2). Our bounding manifolds will then belong to different $\mathcal{C}$-bundles and will then have different "shapes". Figure 2.21(a) shows the bounding manifolds.



(a) (b)

Figure 2.21: (a) Bounding manifolds for the 3R end-effector path in Figure 2.20 (solid line), projected onto the $q_2 - q_3$ plane, obtained from the composition of elbow-left (red) and elbow-right (blue) inverse kinematic solutions; (b) pre-image of the workspace path obtained from the union of all the self-motion manifolds for $t \in [0, 9]$

As a consequence, the surface constituting the pre-image of the workspace path is not a deformed torus, but the much more complex surface of Figure 2.21(b), that is also a composition of the typical manifold structures identified in [26].

As Figure 2.22 demonstrates, the path curve in the joint space begins between the large and the small boundary manifolds, either in the *elbow-right* (top-right curve) or in the *elbow-left* (bottom-left

curve) configuration. It touches the large boundary manifold at $\mathbf{x}_1$, then leaves it to cross the *coregular* surfaces at $\mathbf{x}_2$ and $\mathbf{x}_4$. It touches one of the small manifolds at $\mathbf{x}_5$. It ends in the same workspace point as where it started, but in a different configuration, as the cyclicity requirement is not imposed.

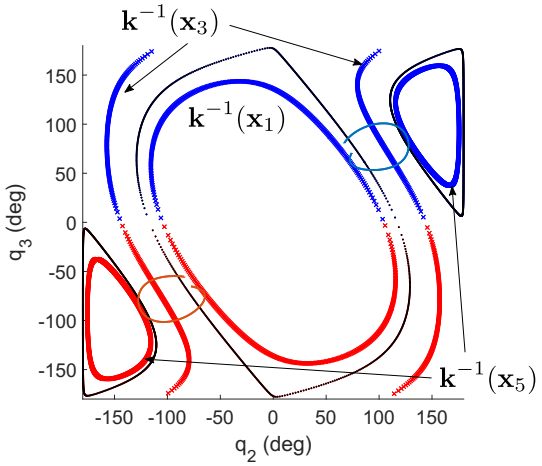


Figure 2.22: Optimal solutions for elbow-left (red) and elbow-right (blue) configurations; coregular surfaces projections (black) are also shown, together with bounding manifolds; the pre-image of regular value $\mathbf{x}_3$ is also shown for completeness

The first thing that one may notice by looking at Figure 2.22 is the symmetry of the solutions. Indeed, the line passing through points $\mathbf{x}_1$ and $\mathbf{x}_5$ and through the base of the manipulator has gradient 1.01, which means that it almost coincides with the bisector of the quadrant. However, the symmetry is not exact, as the velocity and acceleration profiles are not symmetric with respect to the same line, but respect to the horizontal line passing through the center of the 3R manipulator's path. As velocity is minimized, configurations are also affected. This is also confirmed by the fact that the optimal solutions are not symmetric when plotted versus time, as in Figure 2.19.

The three-dimensional Cartesian representation of the solutions is reported in Figure 2.23, where two different viewpoints are given

to ease the comprehension of the geometry. The reader may verify that the two curves are not $\mathcal{C}$-path-homotopic and so they would be even if they were closed under the cyclicity requirement.

Here we have considered the optimal solutions in both the elbow configurations, but from Figure 2.16, we know that other solutions exist satisfying the Euler-Lagrange necessary conditions. Each of them could be $\mathcal{C}$-path-homotopic to one or the other of the optimal solutions, or, potentially, they could belong to a different $\mathcal{C}$-path-homotopy class. According to [25], the BVP (that has not been solved here, because the problem was transformed in multiple IVPs) is unable to distinguish between such solutions. As remarked at the end of Section 2.7.4.1, if the dynamic programming algorithm is implemented in a way that the whole configuration space can be explored, i.e. multiple grids are visited together, it would be able to overcome such a limit. For the example considered here, Algorithm 2 will only find the elbow-right curve (blue) which we know to be the global optimum.

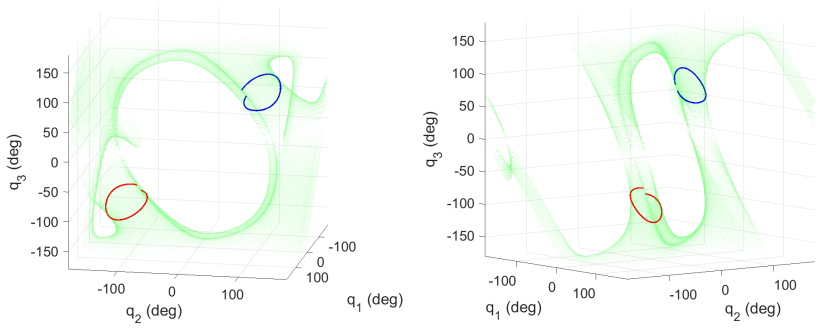


Figure 2.23: Three-dimensional Cartesian visualization of solutions in the joint space, plotted onto the surface corresponding to the pre-image of the workspace path

We are now ready to address the second objective of this section: to investigate about the capabilities of the manipulator to change its posture "on the way". In real scenarios, this could happen

because of certain constraints in the workspace (e.g. obstacles) or because of the imposition of certain boundary conditions or cost functions, as we will show in the next sections.

With reference to Figure 2.22, the topological analysis shows that the manipulator would be able to fully reconfigure itself while the end-effector operates in $\mathcal{W}$-cover 3, as the self-motion manifolds close in circles within the same $\mathcal{C}$-bundle, e.g. $\mathbf{k}^{-1}(\mathbf{x}_1)$. As it crosses the first coregular value manifold, getting into $\mathcal{W}$-cover 2, it can still reconfigure joint 3 between *elbow-left* and *elbow-right* configurations, but q_2 assumes either positive or negative values, in two different self-motion manifolds, e.g. $\mathbf{k}^{-1}(\mathbf{x}_3)$. In other words, the capability of the manipulator to reconfigure itself is limited. In fact, the pre-image of $\mathcal{W}$-cover 2 is made up of two disjoint $\mathcal{C}$-bundles, separated by singularities $[q_1, 0, \pm\pi]$, $[q_1, \pm\pi, 0]$ and $[q_1, \pm\pi, \pm\pi]$. Last, when the end-effector crosses the second coregular value manifold, getting into $\mathcal{W}$-cover 1, the manipulator has no possibility to reconfigure itself at all, e.g. $\mathbf{k}^{-1}(\mathbf{x}_5)$. Here, the pre-image of $\mathcal{W}$-cover 1 is made up of two disjoint $\mathcal{C}$-bundles, separated by singularities $[q_1, \pm\pi, 0]$ and $[q_1, \pm\pi, \pm\pi]$.

In Figure 2.24, an alternative visualization of the $\mathcal{C}$-bundles is given with relation to the trajectory. Vertical black lines correspond to coregular surfaces, hence they contain singularities, in particular at the points of tangency with the white areas. The regions of the elbow-left and elbow-right grids where the colors are the same for both q_2 and q_3 correspond to the configurations where a passage from one grid to the other is possible. Such regions adjoin white areas, where no solution to the inverse kinematic problem exists. As reconfiguration is only possible by passing by a singularity of the subchain made of links 2 and 3, one may also look for the light blue (0 deg) and red ($\pm$180 deg) regions on the q_3 grids. The reader may realize that such regions exist in $\mathcal{C}$-bundles 2 and 3, while they do not in $\mathcal{C}$-bundle 1, where, if the manipulator approached a white region, it would also approach the boundaries of $\mathcal{C}$-bundle 1, ending up in a singularity.

The same can also be verified in Figure 2.22 again. If the manip-

ulator was in $\mathcal{C}$-bundle 1, that is where $\mathbf{k}^{-1}(\mathbf{x}_5)$ is, and tried to change its elbow configuration from right to left or left to right without crossing the coregular surface, it should pass through either $\mathbf{q} = [q_1, \pm\pi, \pm\pi]^T$ or $\mathbf{q} = [q_1, \pm\pi, 0]^T$, which are both singularities of the whole kinematic chain.

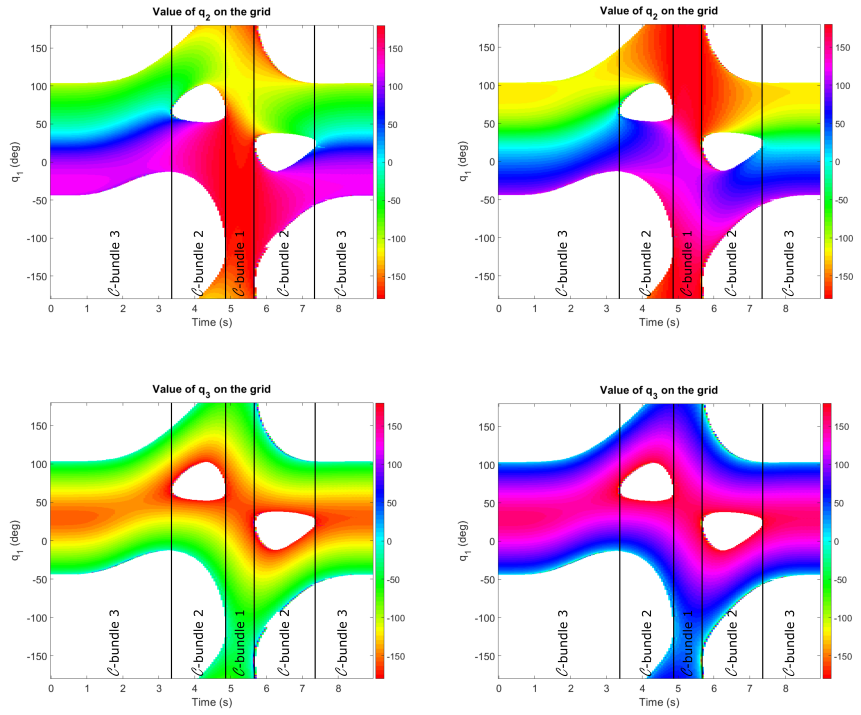


Figure 2.24: Grids in q_2 (top) and q_3 (bottom) for elbow-left (left) and elbow-right (right) configurations with overlapped the $\mathcal{C}$-bundles boundaries

The transition map of Figure 2.25 is built by computing the distance (i.e. the norm of the difference) between joints configurations in different grids. The darker the color is, the closer the configurations are, making the jump possible from one grid to the other. The reader may notice that such regions correspond to the contours of the white areas and that in $\mathcal{C}$-bundle 1, the jump is not

possible without hitting a singularity.

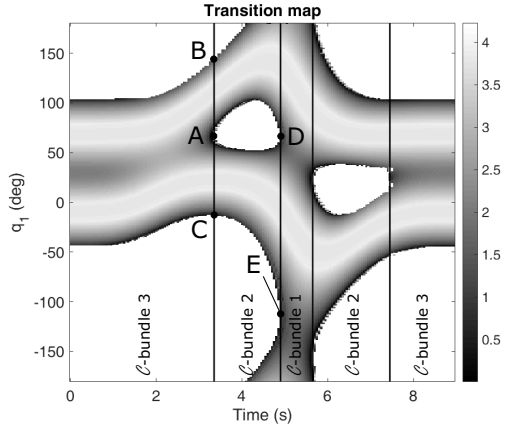


Figure 2.25: Transition map showing areas (black) where the jump from one elbow configuration to the other is possible; difference is in radians

Referring again to Figure 2.24, it is also interesting noticing how in $\mathcal{C}$-bundles 2 it is not possible to pass from positive to negative values of q_2, as noted above about Figure 2.22.

The transition map of Figure 2.25 is also a useful tool to provide a 2D representation of a complex multi-dimensional manifold. In [26], several sub-structures as cylinders with "pinched" ends and bifurcating manifolds have been identified. The transition map can be used to understand these shapes. The elbow-left and elbow-right grids presented in this section can be seen as two surfaces that meet at the black borders of the transition map and close in manifolds. In this sense, the elbow-left and elbow-right grids can be seen as projections of the lower and upper faces of the same three-dimensional surface. In $\mathcal{C}$-bundle 3, there only is one manifold. On one side, for $t = 0$ and $t = 9$, its cross-section is a circle, while at the borders with $\mathcal{C}$-bundle 2, its cross-section is an eight-shaped line whose center is the singularity A. Regular points B and C are where the grids are connected forming one manifold. In $\mathcal{C}$-bundle 2, each of the circles that make up

the eight-shaped border constitutes the beginning of two disjoint manifolds that are separated by an empty region (white area between A and D in Figure 2.25). This is the same as Figure 6 in [26]. In $\mathcal{C}$-bundle 1, each grid contains a full manifold, in fact there are no junction points between the two grids, except for singularities that, however, only exist at the borders of the bundle. A cross-section of the border between $\mathcal{C}$-bundle 2 and $\mathcal{C}$-bundle 1 is given in Figure 2.26. The dotted line corresponds to the upper manifold of $\mathcal{C}$-bundle 2 ($q_2 < 0$), while the solid line corresponds to the lower manifold of $\mathcal{C}$-bundle 2 ($q_2 > 0$). Both manifolds contain both elbow-left and elbow-right configurations, coherently to Figure 2.22. Such manifolds intersect at singularities D and E, where they generate, in $\mathcal{C}$-bundle 1, an inner (blue, elbow-right) and an outer (red, elbow-left) manifold that concentrically run together along $\mathcal{C}$-bundle 1 without meeting. This type of transition is a combination of junction (of the upper and lower manifolds of $\mathcal{C}$-bundle 2) and bifurcation (of the elbow-left and elbow-right manifolds of $\mathcal{C}$-bundle 1), that is not present in the examples from [26].

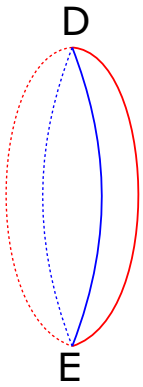


Figure 2.26: Cross-section of the border between $\mathcal{C}$-bundle 2 and $\mathcal{C}$-bundle 1, where D and E are the singularities of Figure 2.25

Anytime that manifolds bifurcate, as remarked in Section 2.6.3, new $\mathcal{C}$-path-homotopy classes are generated [26]. Once again, if the state space grids fully represent the pre-image of the workspace

path and transitions are allowed between them, i.e. the joint-space solutions can travel around the manifold, the globally-optimal solution will be found, across all $\mathcal{C}$-path-homotopy classes. With respect to the algorithm presented in [26], the complexity of the dynamic programming algorithm is independent of the number of $\mathcal{C}$-path-homotopy classes, that can be in the order of millions already for a simple trajectory traversing a few coregular surfaces.

2.7.5.4 Online reconfiguration

In light of the topological analysis above, it is interesting to show how, by using Algorithm 2, the manipulator is able to reconfigure itself, when required by the specific application, jumping from one grid to the other.

In order to do so, assume the manipulator has to track the trajectory of Figure 2.14 and the square norm of velocities has to be minimized (same as before), but, this time, the robot is forced to begin its motion in an elbow-left configuration and end it in an elbow-right configuration. The specific initial and final joints positions are left as results of the problem. One could call them *free elbow-constrained boundary conditions*. In practical terms, this can be done by imposing ad-hoc punctual position limits on q_3, i.e.

$$\begin{aligned} q_3 &< 0 \text{ for } t = 0 \ s \\ q_3 &> 0 \text{ for } t = 9 \ s \end{aligned} \tag{2.144}$$

Anywhere in between $t = 0$ and $t = 9$, q_3 is free, which means that the solution is always free to "jump" from one grid to the other. Figure 2.27 shows the solution for q_1 obtained from the execution of Algorithm 2, overlapping the transition map of Figure 2.25.

At time $t = 5.8$ s, soon after the manipulator passes from $\mathcal{C}$-bundle 1 to $\mathcal{C}$-bundle 2, the solution approaches the black region therein and jumps from the elbow-left to the elbow-right grid. As expected, because of constraints (2.144), the cost increases to

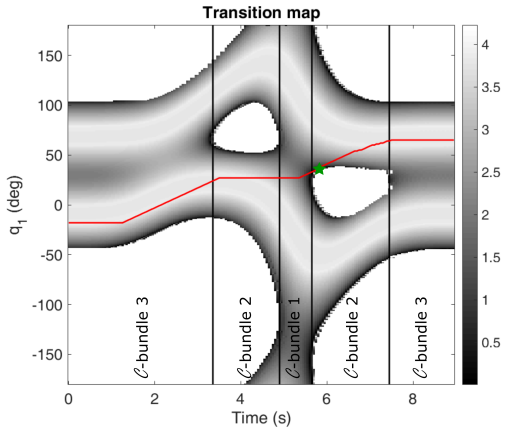


Figure 2.27: Solution for q_1 obtained by minimizing the square norm of velocities subject to constraints (2.144); the star indicates the time instant at which the jump happens

$I = 4.18$, obtained with a u-set of 1-degree resolution and a t-set of 0.05 seconds resolution.

Figure 2.28 reports a sequence of four views of the manipulator in its workspace while tracking the assigned trajectory, showing

- the manipulator in $\mathcal{C}$-bundle 2, before entering $\mathcal{C}$-bundle 1 (a);
- the manipulator in $\mathcal{C}$-bundle 1, prior to the reconfiguration (b);
- the manipulator at the reconfiguration point, in $\mathcal{C}$-bundle 2 (c);
- the manipulator after the reconfiguration, in $\mathcal{C}$-bundle 2 (d).

To complete the analysis of this case study, the optimal trajectory is also drawn in the joint space in Figure 2.29, where the projection onto the $q_2 - q_3$ plane is represented. The reader may recognize that the configuration space represented is the same as Figure 2.22,

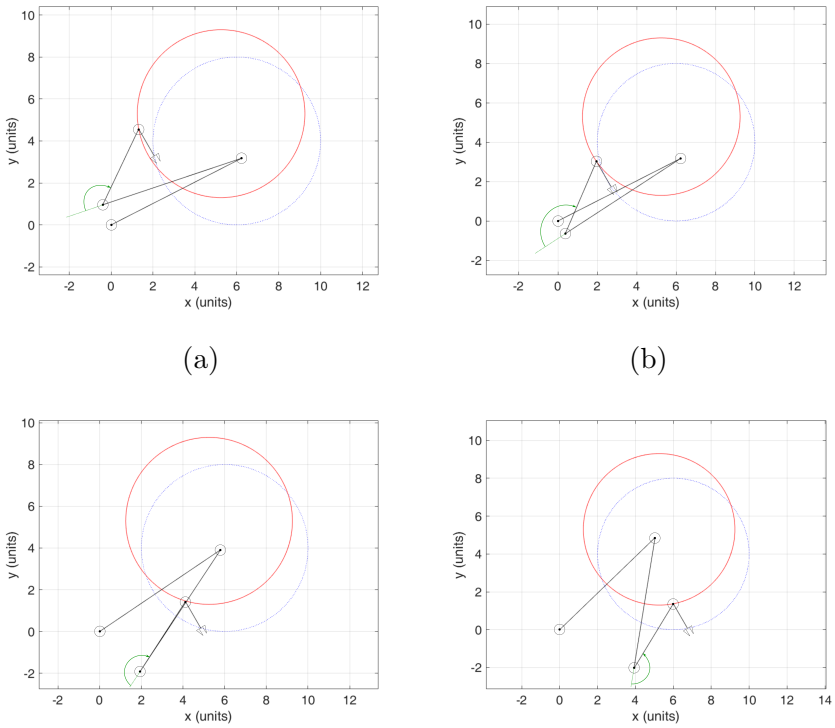


Figure 2.28: Sequence of four consecutive frames showing the joints configuration in the proximity of the reconfiguration point; for paths, the same color code as Figure 2.20 has been used

but both q_2 and q_3 now take values in the interval $[0, 2\pi]$ instead of $[-\pi, \pi]$. From this representation, it is evident that the motion starts in the elbow-left configuration and ends in the elbow-right configuration, as requested by constraints (2.144). In order to switch from one configuration to the other, the manipulator has to approach the boundary between the red and the blue regions in a bundle where they are connected (i.e. where they form one self-motion manifold). In this case, the chosen boundary is at $q_3 = 180°$ and the bundle where the jump happens is $\mathcal{C}$-bundle 2.

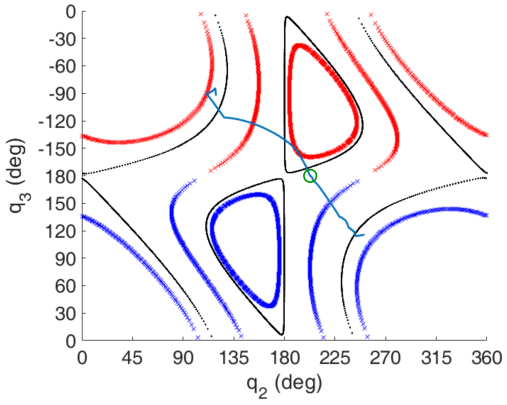


Figure 2.29: Optimal trajectory drawn in the joint space and $\mathcal{C}$-bundles. Highlighted by the green circle, the point at which the reconfiguration happens, i.e. $q_3 = 180°$

With respect to previous works, the latter example demonstrated that, by working with multiple dynamic programming grids (e.g. by using Algorithm 2), it is possible to find the global minimum across all manifolds and configurations. Recalling the limits of the Euler-Lagrange conditions expressed in [25], the optimal solution across all $\mathcal{C}$-path-homotopy classes can be found as well.

At the same time, it was shown that this method keeps the flexibility of a typical dynamic programming redundancy resolution scheme, where arbitrarily complex constraints and cost functions can be given as inputs to the problem.

2.7.6 Comparison with BVPs

Now that we have learnt that algorithms based on the multiple grids search are capable to find solutions spanning the whole joints space, it would be interesting to make a comparison with the capabilities of a classical redundancy resolution scheme based on calculus of variations.

In order to do so, it is necessary to remove all the constraints, such as (2.144), which cannot be easily handled with calculus of variations. Moreover, we would like the boundary conditions to be free, such that the "true" optimal solutions can be compared. The objective is then to define a performance index which, in order to be optimal, clearly requires the manipulator to reconfigure its posture without the imposition of specific constraints.

The possibility to use configuration-dependent terms, as shown in Section 2.4.5.4, comes in help in this case. One possibility is to use the distance from an obstacle (or multiple obstacles), or, for the simple manipulator and path of Figure 2.13, one could think to minimize the distance between the elbow, i.e. the third joint, and the center of the circular path. The latter results in a simpler expression to be handled with calculus of variations and certainly requires the manipulator to reconfigure its posture. From the practical standpoint, this distance could be an approximate measure of the manipulator's footprint while the task is accomplished, as the expected result is that the elbow will be kept, as much as possible, inside the circular path.

Thus, consider the following performance index:

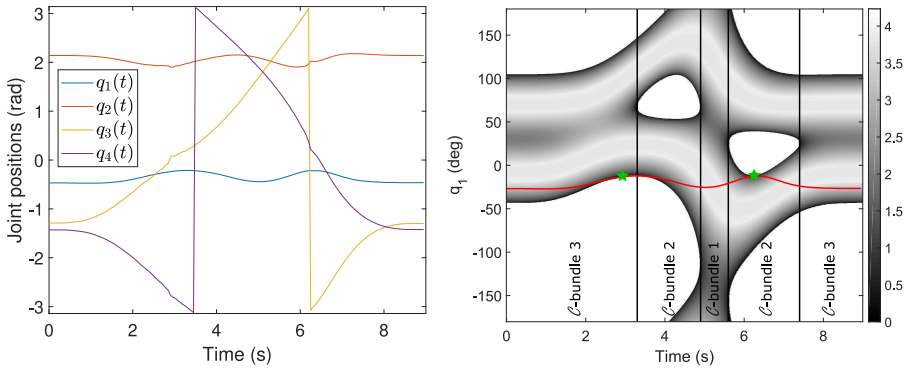
$$G = w_v \dot{\mathbf{q}}^T \dot{\mathbf{q}} + w_c d(\mathbf{q}) \tag{2.145}$$

where w_v and w_c are scalars summing to one, representing the weights associated to the velocity-dependent and the configuration-dependent performance indices respectively. $d(\mathbf{q})$ represents the square distance between the elbow position ($\mathbf{x}_{j3}$) and the center of the circular path ($\mathbf{x}_C$):

$$d(\mathbf{q}) = (\mathbf{x}_{j3}(\mathbf{q}) - \mathbf{x}_C)^T (\mathbf{x}_{j3}(\mathbf{q}) - \mathbf{x}_C) \tag{2.146}$$

By using the same parametrization as Section 2.7.5.2, the optimal solution can be found with Algorithm 2. For $w_v = 0.1$ and $w_c = 0.9$, the optimal joint positions are reported in Figure 2.30 (a). The cost associated to such a solution is $I = 7.90$, obtained with a grid resolution of 0.25 deg along the input axis and 0.05 s along the

time axis. Velocity and acceleration limits are set to infinity, so as to make the comparison with the BVP straightforward, where constraints of this kind cannot be easily accommodated.



(a) (b)

Figure 2.30: Optimal solution for the performance index (2.145) computed with Algorithm 2; (a) optimal joint positions, (b) transition points and optimal joint 1 trajectory

As we may have expected, there is a very little variation of the first two joint positions, whereas the trajectory is tracked by mainly exploiting the motion of joints 3 and 4, which accomplish a 360 degrees travel, in order to keep the elbow close to the center of the circular path. The manipulator jumps from one elbow configuration to the other twice: at $t = 2.9$ s and $t = 6.25$ s, highlighted by the star markers in Figure 2.30 (b). Four snapshots picturing the manipulator tracking the optimal trajectory are reported in Figure 2.31, showing

- the manipulator in $\mathcal{C}$-bundle 3, prior to the first reconfiguration (a);
- the manipulator in $\mathcal{C}$-bundle 2, after the first reconfiguration (b);
- the manipulator in $\mathcal{C}$-bundle 2, prior to the second reconfiguration (c);

- the manipulator in $\mathcal{C}$-bundle 2, after the second reconfiguration (d).

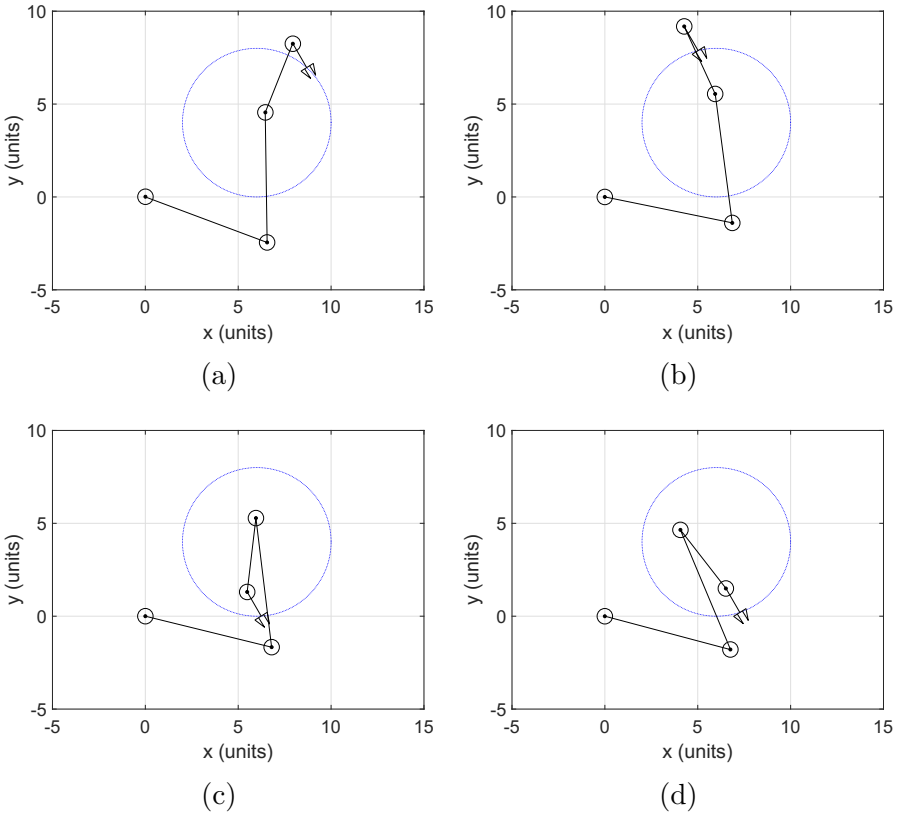


Figure 2.31: Four snapshots showing the joints configuration in the proximity of the two reconfiguration points

The joint trajectories of Figure 2.30 (a) are not everywhere smooth. If necessary, one may improve its smoothness by employing one of the techniques discussed in Section 2.7.4.3. For instance, the usage of a spline interpolation yields the result of Figure 2.32. The interpolation is only performed on the curve q_1 and the others are obtained by inverse kinematics, so as to guarantee that the path constraint is respected.

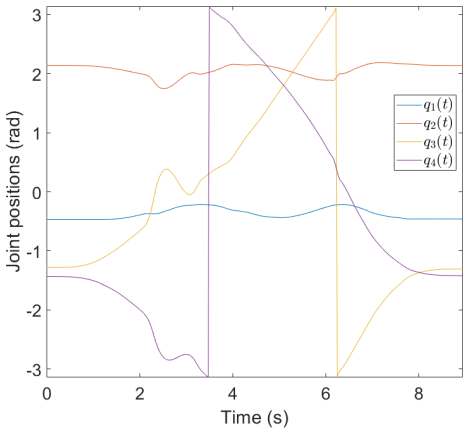


Figure 2.32: Optimal solution of Figure 2.30(a) smoothed with spline interpolation applied to q_1

Now consider again the cost function (2.145). By following the same steps as any of the typical performance indices of Section 2.4.5, the solution must satisfy

$$\ddot{\mathbf{q}} = \mathbf{J}^{\dagger}(\ddot{\mathbf{x}} - \dot{\mathbf{J}}\dot{\mathbf{q}}) + \frac{w_c}{w_v}(\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\mathbf{J}_{j3}^{T}(\mathbf{x}_{j3} - \mathbf{x}_C) \tag{2.147}$$

where $\mathbf{J}_{j3}$ is the Jacobian associated to the direct kinematics of joint 3:

$$\mathbf{J}_{j3} = \frac{\partial \mathbf{x}_{j3}}{\partial \mathbf{q}} \tag{2.148}$$

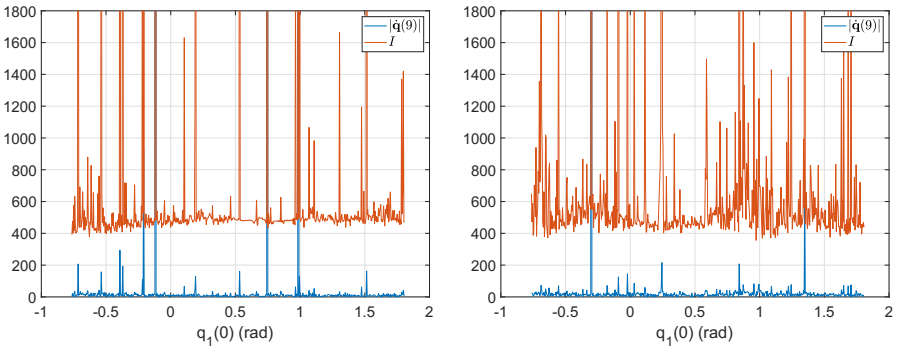
In addition, the solution must satisfy the following boundary conditions:

$$\begin{aligned} \dot{\mathbf{q}}(t_i) &= \mathbf{J}^{\dagger}(\mathbf{q}(t_i))\dot{\mathbf{x}}(t_i) \\ \dot{\mathbf{q}}(t_f) &= \mathbf{J}^{\dagger}(\mathbf{q}(t_f))\dot{\mathbf{x}}(t_f) \end{aligned} \tag{2.149}$$

with $t_i = 0\ s$, $t_f = 9\ s$ and $\dot{\mathbf{x}}(t_i) = \dot{\mathbf{x}}(t_f) = \mathbf{0}$ for the example considered here.

Again, like all the other performance indices mentioned in Section 2.4.5, the solution can be found by solving a *Two-Point Boundary*

Value Problem (TPBVP). As done in Section 2.7.5.1, the initial states can be discretized and as many IVPs as the number of states in such a discrete set can be solved to seek the solution. The same graph as Figure 2.16, showing the norm of the final joints velocities and cost, is reported in Figure 2.33.



(a) (b)

Figure 2.33: Final joint rates and cost function for elbow-left (a) and elbow-right (b) configurations

This time though, the difference of a few orders of magnitude between the cost function and the final joints rates does not allow to make a proper analysis by only looking at the graphs. However, the norm of joints velocities at the final time is never zero (or sufficiently close to zero), regardless of the initial joints configuration, which means that none of the solutions is a solution to the BVP and, in turn, none of them respects the necessary conditions for optimality. It is important noticing that the absence of a solution is not due to the BVP itself, but to the fact that the proximity to a singularity makes the joint velocities grow almost instantaneously. As a consequence, the numerical integration induces errors bringing the joints to velocities other than zero at the final time. However, even though a CLIK implementation [22] was able to recover from the "instability", the high joints velocities would affect the cost function and the global optimality of the solution [23].

In this specific example, the value of the cost function is never lower than about $I = 400$, a order of magnitude higher than $I = 7.90$, obtained with dynamic programming. Recalling the considerations made in Section 2.4.6, links 2 and 3 must align twice to keep the elbow joint inside the circular path. As said, this leads the smallest singular value to zero and, in turn, makes joint velocities rapidly grow. The result is the same anomalous behavior that previous authors have experimented, which explains why the cost is extremely high compared to dynamic programming.

Once again, this confirms that no difference exists, from such a point of view, between local and global solutions and that, indeed, under certain circumstances, calculus of variations may not be able to provide a practical solution at all.

On the other hand, as already observed in Section 2.7.5.4, because no Jacobian inversion is performed, the multiple grid algorithm based on dynamic programming is capable to explore the whole configurations space and find a solution across different manifolds and $\mathcal{C}$-path-homotopy classes, whenever this is necessary to optimize the given performance index. Practically speaking, this means that tasks that would have been unfeasible with calculus of variation (e.g. to minimize the manipulator's footprint while following a circular path, starting and ending with zero velocity) could actually be feasible and tasks that would have been performed sub-optimally (i.e. optimally in a $\mathcal{C}$-path-homotopy class) could actually be performed optimally in a global sense.

2.7.7 Computational complexity and efficiency

Although the globally-optimal redundancy resolution is a planning task to be carried out off-line, there are certain applications that require to execute such an optimization process in a bounded time frame.

A first example is provided in [35] and concerns the optimization of a robotized workcell layout. In fact, because the work has to

be repeated several times and each execution comes with a certain cost (execution time, energy consumption, mechanical wear), it is of interest to find the best positioning of the robot minimizing such a cost. The employment of redundant robots provides more room for reducing such a cost, but it is still a function of the workcell layout. Thus, the redundancy resolution has to be performed as fast as possible, so that more layouts can be tested by the operator, who usually interacts with a CAD-based software simulator.

In space applications of planetary exploration and, more in general, in all the mission scenarios envisaging relayed communications between the ground control and the space asset, a bounded time frame is assigned on ground to the planning stage, which may vary from some minutes to a few hours [50]. In such cases, it is clear that the redundancy resolution process has to be completed in due time, so as to allow for a smooth progressing of the control center activities.

These scenarios provide a motivation to discuss about the computational complexity of dynamic programming and relating graph search algorithms applied to redundancy resolution.

2.7.7.1 Efficiency of the dynamic programming algorithm

Assume that the multi-grid dynamic programming algorithm, as presented in Section 2.7.4.1, is employed and that the underlying state space is discretized as described in Section 2.7.3.2 so as to have multiple grids (either homogeneous or non-homogeneous). In the terminology used in [47], each column of the grid, obtained for fixed k, i.e. the grid-index, and i, i.e. the time index, and variable j, i.e. the redundancy parameter index, is termed *cluster*. Two clusters are adjacent if they correspond to adjacent time samples, regardless of the grid they belong to. This means that cluster $\{\mathbf{q}_{i,j}^{(k)}, j = 1..N_u\}$ has $2N_g$ adjacent clusters that are $\{\mathbf{q}_{i-1,j}^{(g)}, j = 1..N_u\}$, with $g = 1..N_g$, and $\{\mathbf{q}_{i+1,j}^{(g)}, j = 1..N_u\}$, with $g = 1..N_g$. As discussed in Section 2.7.4.1, dynamic programming solvers are based on the comparison between adjacent clusters, say $\{\mathbf{q}_{i,j}^{(k)}, j =$

$1..N_u\}$ and $\{\mathbf{q}_{i+1,j}^{(g)}, j = 1..N_u\}$, in order to find the minimal cost predecessor for each $\mathbf{q} \in \{\mathbf{q}_{i+1,j}^{(g)}, j = 1..N_u\}$, with $g = 1..N_g$.

This means that each single cell of the current cluster (identified by indices i and k) is compared to all the cells (of cardinality N_u) of all the next clusters (of cardinality N_g). Thus, the computational complexity of this algorithm is $O(N_i N_u^2 N_g^2)$. Because N_g is never greater than 16, $N_g << N_u$ and is a constant of the system, while N_i and N_u can be modified to increase precision. For this reason, for a given application, the complexity can be considered of $O(N_i N_u^2)$ [47]. For typical values of N_i and N_u, in the case of real industrial manipulators, finding the globally-optimal solution may require up to several hours on modern architectures. In turn, setting up a workcell, as in the process described above, may require one or more working days. In the example of ground planning of robotic assets performing planetary exploration on extra-terrestrial celestial bodies, such optimization time may lead not to meet the planning deadline, decreasing the utilization of the expensive space asset.

Thus it is of interest to investigate about techniques to reduce the computational complexity, without affecting the global-optimality of the solution, as discussed next.

2.7.7.2 Homogeneous grids

One criterion that can be adopted to reduce the computation time is to limit the number of comparisons between adjacent clusters. This is actually possible if grids are homogeneous, since, if a velocity constraint is not respected for the couple $\{\mathbf{q}_{i,j}^{(k)}, \mathbf{q}_{i+1,h}^{(g)}\}$, it will not be respected for any other redundancy parameter index beyond h. The same idea, applied to the problem of time-optimal control of non-redundant robots is described in [14]. A pictorial view of this optimization is represented in Figure 2.34. The modified procedure is reported in Algorithm 3, where unnecessary details have been removed for the sake of clarity.

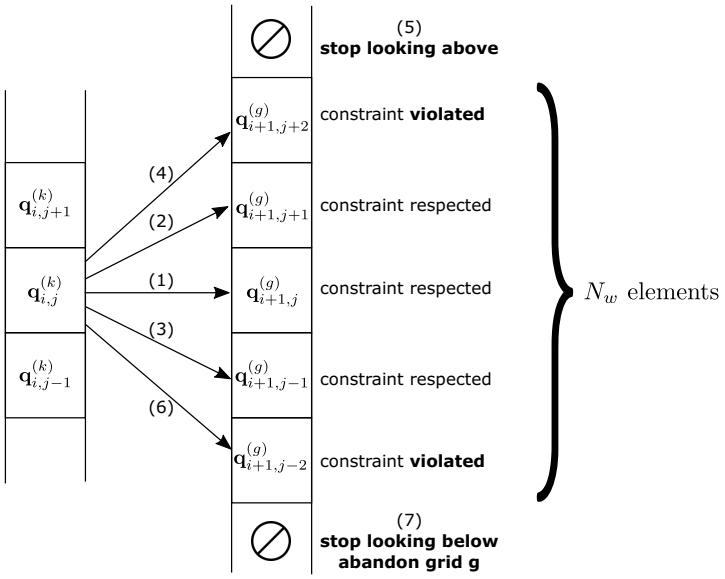


Figure 2.34: Algorithm optimization exploiting homogeneous grids

Algorithm 3 *Forward* iterative dynamic programming algorithm with multiple grids and optimization based on homogeneous grids

Initialize cost map $I_{i,j,k} = +\infty, \forall i = 1..N_i, \forall j = 1..N_u, \forall k = 1..N_g$
Initialize cost map $I_{1,j,k}, \forall j = 1..N_u, \forall k = 1..N_g$ *with the initial cost*
for $i \leftarrow 1$ to $N_i - 1$ **do**
  for each grid k **do**
    for each $\mathbf{q}_j$ in $\{\mathbf{q}^{(k)}_{i,j}\}$ within limits **do**
      for each grid g **do**
        $w \leftarrow 0$
        while $h \leftarrow j \pm w$ are admissible indices in $\{\mathbf{q}^{(g)}_{i+1,h}\}$ **do**
          $\mathbf{q}_h \leftarrow \{\mathbf{q}^{(g)}_{i+1,h}\}$
          if derivative constraints for $\mathbf{q}_j$ and $\mathbf{q}_h$ are respected **then**
            Compute instantaneous cost function l
            if $I_{i,j,k} + l < I_{i+1,h,g}$ **then**
              $I_{i+1,h,g} = I_{i,j,k} + l$
              Let u_j, $\mathbf{q}_j$ *at* i *be the predecessors of* u_h, $\mathbf{q}_h$ *at* $i+1$
            Increase w *of* 1
          else
            Abandon grid g
$I_{opt}(N_i) = \min_{j,k} I_{N_i,j,k}$
Build functions $\mathbf{q}(i)$ *of optimal states by screening the predecessors map backwards*

The set of nodes of the next cluster visited from each single node

of the current cluster is termed *lookup window*, as shown in Figure 2.34 and has a cardinality of N_w. The discussed implementation allows to reduce the computational complexity of the algorithm to $O(N_i N_u N_w)$, as the number of constraint tests is no longer N_u^2, but $N_u N_w$, with $N_w << N_u$. For typical applications, N_w is one to three orders of magnitude lower than N_u, yielding a drastic reduction of the execution time. Experiments show that a solution can be found in a few minutes on modern architectures, even for very fine discretizations of the redundancy parameters, as summarized in Table 2.2. The algorithm has been implemented in C++ in ROS and executed on a 64-bit Ubuntu 16.04 LTS OS running on an Intel® Xeon(R) CPU E5530@2.40GHz ×8, using the example of Section 2.7.5, with $N_i = 180$. No multi-core execution model has been used in the tests.

$\mathbf{N_u}$	**Redundancy parameter resolution**	**Algorithm 2**	**Algorithm 3**
360	1 deg	25 seconds	1 second
720	0.5 deg	4 minutes	9 seconds
1440	0.25 deg	20 minutes	33 seconds
2880	0.125 deg	103 minutes	2 minutes

Table 2.2: Comparison of execution times between unoptimized and optimized multi-grid DP redundancy resolution algorithms

It is worth mentioning that this kind of optimization can only be used for velocity constraints, as the principle at the basis of the optimization, and represented in Figure 2.34, is not true for other quantities. With an example, given the pair $\{\mathbf{q}_{i,j}^{(k)}, \mathbf{q}_{i+1,h}^{(g)}\}$, with $h > j$, if the acceleration constraint is not satisfied, it could be satisfied for the pair $\{\mathbf{q}_{i,j}^{(k)}, \mathbf{q}_{i+1,h+1}^{(g)}\}$, as it depends on the predecessor of $\mathbf{q}_{i,j}^{(k)}$, that has been determined at the stage before.

Algorithm 3 also allows abandoning, at the first comparison, the

grids to which it is not possible to transit from the current state. In this sense, this is the same as tracking transitional points by using maps such as the one of Figure 2.25 and test the feasibility of the transition only at those points, as proposed in [39].

2.7.7.3 Repeated searches

As anticipated in Section 2.7.4.3, repeated searches can be performed to improve the smoothness of the solution, but, on the other hand, they can also improve the algorithm performances, as demonstrated in [49]. The idea is to perform a first search with a rough discretization of the redundancy parameter, so that it terminates in a short time, returning a first solution. Then, the redundancy parameter is discretized more finely, but only in the neighborhood of the previous solution, so that the number of samples on the redundancy parameter axis does not change. In this way, it is demonstrated that a solution can be found up to 50 times faster than performing only one search on a very fine grid.

However, the reader must be aware that adopting an excessively coarse discretization at the first iteration may yield a sub-optimal solution at the end of the process. For instance, consider the example of Section 2.7.5 and the globally-optimal solution of Figure 2.17, also obtained with dynamic programming in Section 2.7.5.2, with $N_u = 3600$, i.e. 0.1 deg resolution. Assume to downscale the resolution up to 1 deg, so that $N_u = 360$ and to execute either Algorithm 2 or Algorithm 3 using both grids. The result is reported in Figure 2.35.

The reader may recognize that the solution is very far from that of Figure 2.17, implying that any search in its neighborhood will not be able to find the globally-optimal solution. Since the redundancy parameter resolution at which the solution begins resembling the globally-optimal one varies on a case-by-case basis, those interested in picking the globally-optimal solution would be tempted to use quite a fine resolution already at the first stage. This implies losing the advantages of the two-stage optimization in favor

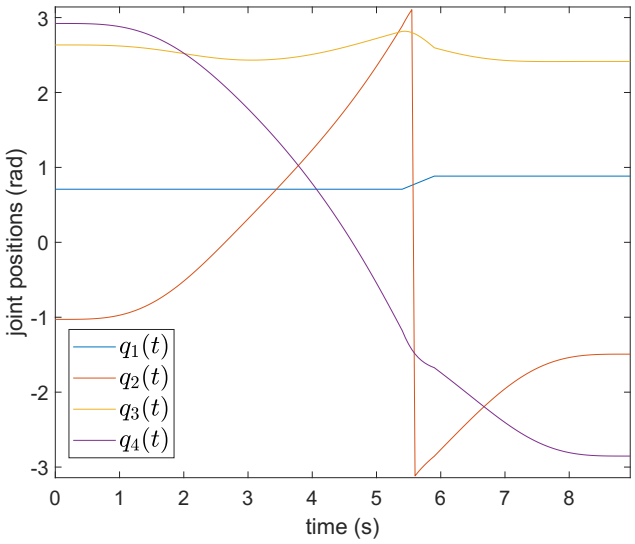


Figure 2.35: Joint positions resulting from the execution of Algorithm 3 on 180×360 grids

of the global optimality.

2.8 Application to a 7-DOF robotic arm

2.8.1 Use case description

In order to demonstrate that the methodology developed in the previous sections can be effectively applied to a real scenario, let us consider a real robotic arm with 7 degrees of freedom, to which a task constrained in position and orientation is assigned, with a time law. The objective is to reduce the energy consumption indirectly through the global minimization of the square norm of joint velocities. Sub-optimal solutions are not of interest, hence the globally-optimal solution must be found. In addition, the computation has to be performed as fast as possible, so as to maximize the number of planning sessions fitting in a given time window. Thus, Algorithm 2 and Algorithm 3 are developed in C++ in ROS

(*Robot Operating System*), by extending the *MoveIt!* framework [51], so as to benefit from the communication and visualization tools already available. Since the Panda robot by Franka Emika [52] has 7 degrees of freedom and is the flagship robot of *MoveIt!*, it is a convenient choice for the experiment at hand.

The reference frames of the Panda are reported in Figure 2.36 and the relating modified Denavit-Hartenberg parameters [53] in Table 2.3 [54]. The configuration represented in Figure 2.36 is $\mathbf{q} = \mathbf{0}^T$. Let us set the joint position, velocity and acceleration limits according to the datasheet. Limits on the jerk are not imposed, but, as discussed in Section 2.7.4.3, it might be convenient to consider them in case the resulting trajectory has to be executed on real hardware.

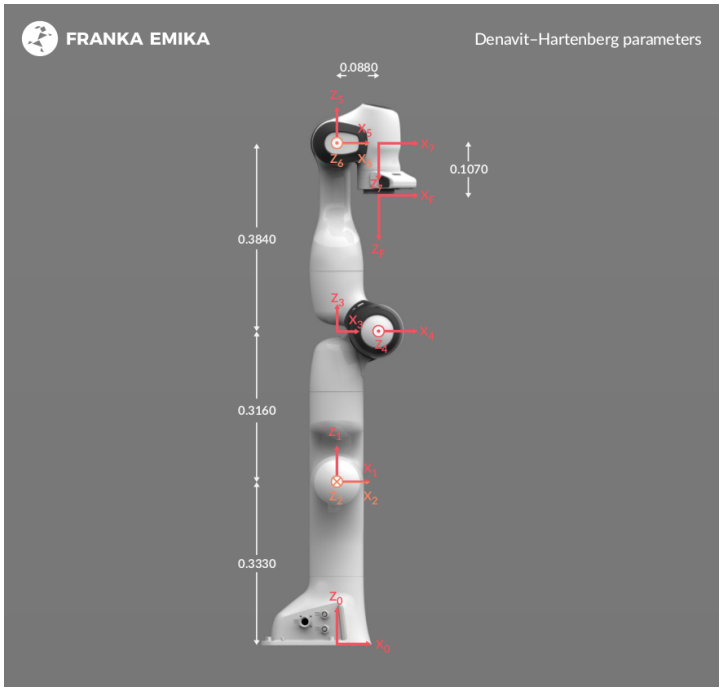


Figure 2.36: Panda reference frames related to the modified Denavit-Hartenberg parameters

The workspace path is defined in terms of position and orientation

	$\mathbf{d_i}$	$\boldsymbol{\theta}_i$	$\mathbf{a_i}$	$\boldsymbol{\alpha}_i$
J1	0.333	q_1	0	0
J2	0	q_2	0	$-\frac{\pi}{2}$
J3	0.316	q_3	0	$\frac{\pi}{2}$
J4	0	q_4	0.0825	$\frac{\pi}{2}$
J5	0.384	q_5	-0.0825	$-\frac{\pi}{2}$
J6	0	q_6	0	$\frac{\pi}{2}$
J7	0	q_7	0.088	$\frac{\pi}{2}$
Flange	0.107	0	0	0

Table 2.3: Panda modified Denavit-Hartenberg parameters

and is depicted in Figure 2.37, together with the base reference frame. The axes x, y and z are in red, green and blue respectively. The planning is performed for the end-effector's flange that has to visit five waypoints, in the order $\mathbf{x}_A$, $\mathbf{x}_B$, $\mathbf{x}_C$, $\mathbf{x}_D$ and $\mathbf{x}_E$, describing the corners of a rectangle in the y-z plane, with variable orientation. Their values with respect to the base reference frame, considering a roll-pitch-yaw representation for the orientation, are

$$\begin{aligned}
\mathbf{x}_A &= \begin{bmatrix} 0.3 & -0.3 & 0.8 & 0 & -\pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_B &= \begin{bmatrix} 0.3 & -0.3 & 0.4 & 0 & -\pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_C &= \begin{bmatrix} 0.3 & 0.3 & 0.4 & 0 & -\pi & 0 \end{bmatrix}^T \\
\mathbf{x}_D &= \begin{bmatrix} 0.3 & 0.3 & 0.8 & 0 & \pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_E &= \begin{bmatrix} 0.3 & -0.3 & 0.8 & \pi/2 & \pi/2 & 0 \end{bmatrix}^T
\end{aligned} \tag{2.150}$$

All the points in between each pair of waypoints are obtained by linear interpolation, with a linear resolution not exceeding 0.01 m.

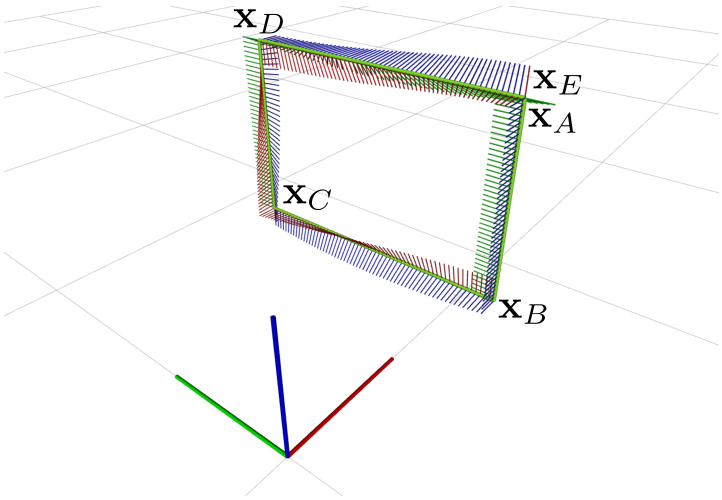


Figure 2.37: Workspace path assigned to the Panda arm, together with the base reference frame

A time law is defined so as to complete the whole trajectory in 60 seconds, with a constant time offset between consecutive points. The total number of points is $N_i = 203$.

2.8.2 Grids computation

In order to speed up the calculation of the dynamic programming grids, as described in Section 2.7.3.2, it is convenient to adopt an analytic inverse kinematic solver, that is several orders of magnitude faster than numeric solvers. In the ROS framework, a possibility is given by *IKFast*, which can find all the IK solutions on the order of 6 microseconds, while most numeric solvers may require even 10 milliseconds or longer, and convergence is not certain [55]. *IKFast* performs an off-line analytic kinematic inversion and generates a C++ library containing the algebraic IK solver, able to return all the solutions for given end-effector position and orientation. The off-line process may require several minutes, but is independent from the assigned trajectory and, thus, needs to be executed only once for a given kinematic chain. Currently, *IKFast*

is able to manage open kinematic chains with one degree of redundancy. The value of the redundancy parameter has to be provided at the time the algebraic solver is called, which is the case of the DP grids considered in this dissertation.

In order to simplify the off-line process of generating the solver library, we choose the redundancy parameter $u = q_4$. In fact, since joint 4 is in the middle of the kinematic chain and its axis does not intersect any other joint axis, we minimize the chances of encountering degenerate cases and of handling more complicated expressions [55]. The redundancy parameter can be discretized so that $N_u = 2880$, either between -180 deg and 180 deg, which yields a resolution of 0.125 deg, or between its physical limits, i.e. -176 deg and -4 deg, which yields a resolution of about 0.06 deg. The Panda manipulator has 8 IK solutions, i.e. $N_g = 8$, for all the points on the trajectory, but in practice, because of joint limits, some points have less. The "slices" corresponding to q_1 of the grids computed with *IKFast* are reported in Figure 2.38, while those computed neglecting joint limits, for comparison purposes, are reported in Figure 2.39.

The first interesting thing to notice about these grids is that they are homogeneous, as evident from those of Figure 2.39. The extended Jacobian $\mathbf{J}_a$, obtained from the 6×7 rectangular Jacobian by adding the row [0 0 0 1 0 0 0], cannot be easily factorized, implying that we are not provided with analytic conditions to classify the solutions of *IKFast*. For this reason, the following three conditions are used, obtained from an a-posteriori analysis of the solution sets:

- $|\mathbf{J}_r^{(4)}| > 0$
- $q_2 > 0$
- $q_5 > 0$

Each of the grids in Figure 2.38 and Figure 2.39 corresponds to a different combination of the conditions above, providing an homogeneous classification of the solutions. It is possible to demonstrate

that both q_2 and q_5 are factors of $\mathbf{J}_a$ and, being the "augmented" Panda manipulator of type 1, according to [38], they are sufficient conditions for classifying the solutions.

The second trait of interest is that q_4 is not so representative of the null space for the trajectory assigned. In fact, by looking at the grids of Figure 2.39 (without joint limits), a large portion of the joint domain does not contain any solution. This means that large variations of the other joints shall be expected for little variations of the redundancy parameter. According to the discussion of Section 2.7.4.3, a fine discretization of the redundancy parameter is needed for the dynamic programming algorithm to provide a smooth solution.

Lastly, it is worth noting that joint limits, in real scenarios, notably reduce the search space, giving a chance to the dynamic programming algorithm to find the globally-optimal solution in a short time. Also, because of joint limits, the Panda is not able to track the assigned trajectory remaining in the same extended aspect, as none of the grids admits a feasible joint-space path from $\mathbf{x}_A$ (corresponding to $i = 0$) to $\mathbf{x}_E$ (corresponding to $i = 203$). Hence, the robot will need to reconfigure its posture on the way by passing through singularities of its kinematic subchains.

2.8.3 Globally-optimal solution

Since grids are homogeneous, the optimization discussed in Section 2.7.7.2 applies. Thus both Algorithm 2 and Algorithm 3 can be executed to find the globally-optimal solution on the grids of Figure 2.38. Table 2.4 reports the execution time of both algorithms and different discretization steps of the redundancy parameter, together with the associated cost function. Tests have been executed on a 64-bit Ubuntu 16.04 LTS OS running on an Intel® Core™ i7-2600K CPU @ 3.40GHz × 8. No multi-core execution model has been used in the tests.

It is interesting to notice that there is not any considerable im-

$\mathbf{N_u}$	**Redundancy parameter resolution**	**Algorithm 2**	**Algorithm 3**	**Cost**
360	0.48 deg	11 seconds	11 seconds	4.27
720	0.24 deg	54 seconds	54 seconds	2.76
1440	0.12 deg	4 minutes	4 minutes	2.44
2880	0.06 deg	14 minutes	13 minutes	2.16
4000	0.04 deg	27 minutes	26 minutes	2.04

Table 2.4: Cost function and performance of DP redundancy resolution algorithm for the Panda example

provement in the performance by using Algorithm 3 in place of Algorithm 2. This means that either position or acceleration limits are almost everywhere stricter than velocity limits for the assigned trajectory.

The convergence rate that we may estimate from the values of the cost function is a confirmation that q_4 is very sensitive for the considered trajectory, meaning that small variations of q_4 yield considerable changes in the solution for the other joints and, as consequence, in the final value of the cost function.

The solution obtained for $N_u = 4000$ is reported in Figure 2.40. It starts from grid 5, then, at $t = 3.3$ s ($i = 12$), it jumps to grid 6 and, at $t = 14.6$ s ($i = 50$), to grid 1. For the majority of the trajectory, up to $t = 48.7$ s ($i = 165$), the solution lies on grid 1. Afterwards, it transits to grid 2 and terminates, achieving 3 posture reconfigurations in total and visiting 4 different extended aspects. As commented in Section 2.7.5.4, posture reconfigurations always happen on the boundaries of the feasible (non-white) regions, where two or more of the maps have the same color for all the joints (only q_1 is shown in Figure 2.38). It is easy to verify that this is the case for the sequence of grids visited by the algorithm

and transitions at the stages mentioned above.

As far as the solution of Figure 2.40 is concerned, the reader may clearly notice the discontinuities in the derivative of the joint positions at each of the three intermediate corners of the trajectory. In between these points the curves are not everywhere smooth. Either a post-processing step or the introduction of jerk constraints would be desirable to allow for the execution on real hardware, as commented in Section 2.7.4.3. Five snapshots from the simulation, corresponding to waypoints $\mathbf{x}_A$, $\mathbf{x}_B$, $\mathbf{x}_C$, $\mathbf{x}_D$ and $\mathbf{x}_E$ are reported in Figure 2.41.

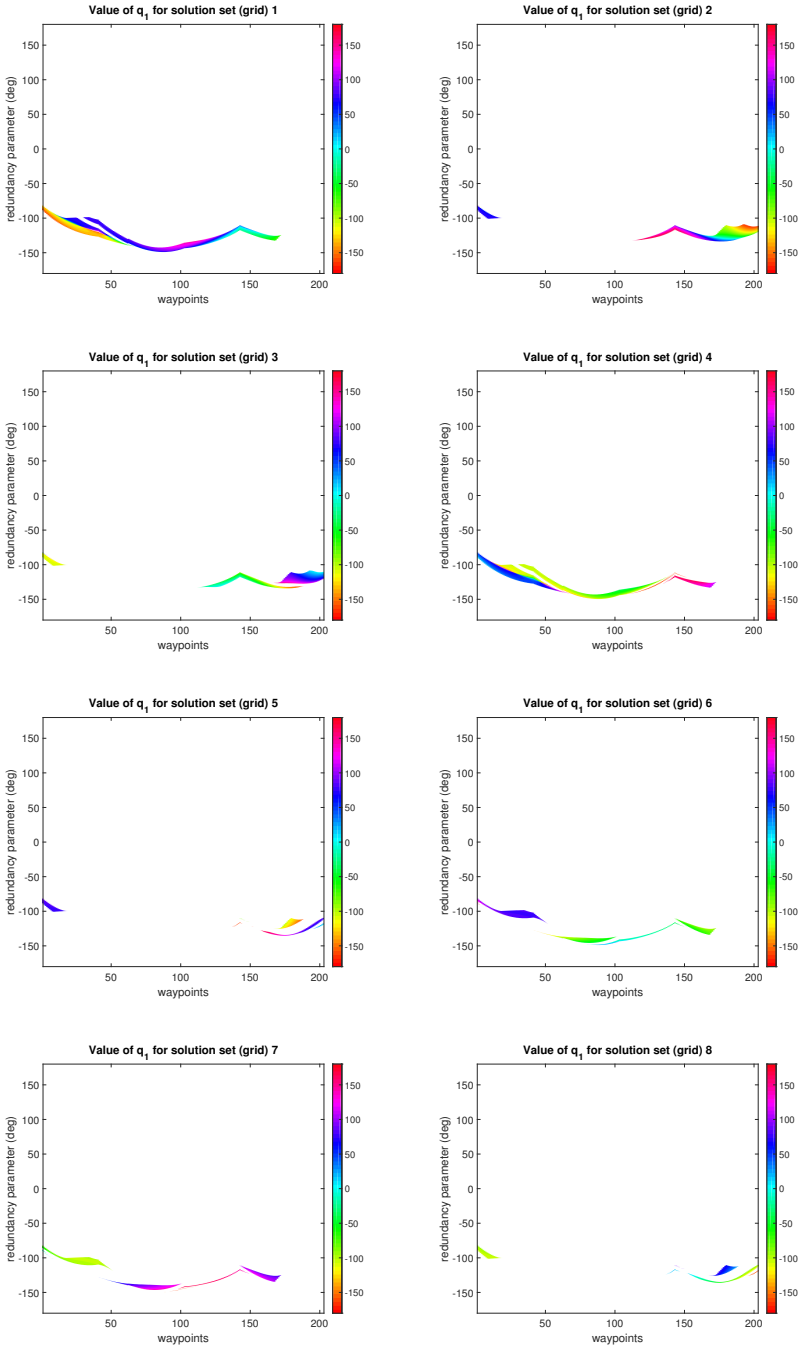


Figure 2.38: Panda grids representing q_1 for the trajectory described in Section 2.8.1 considering joint limits

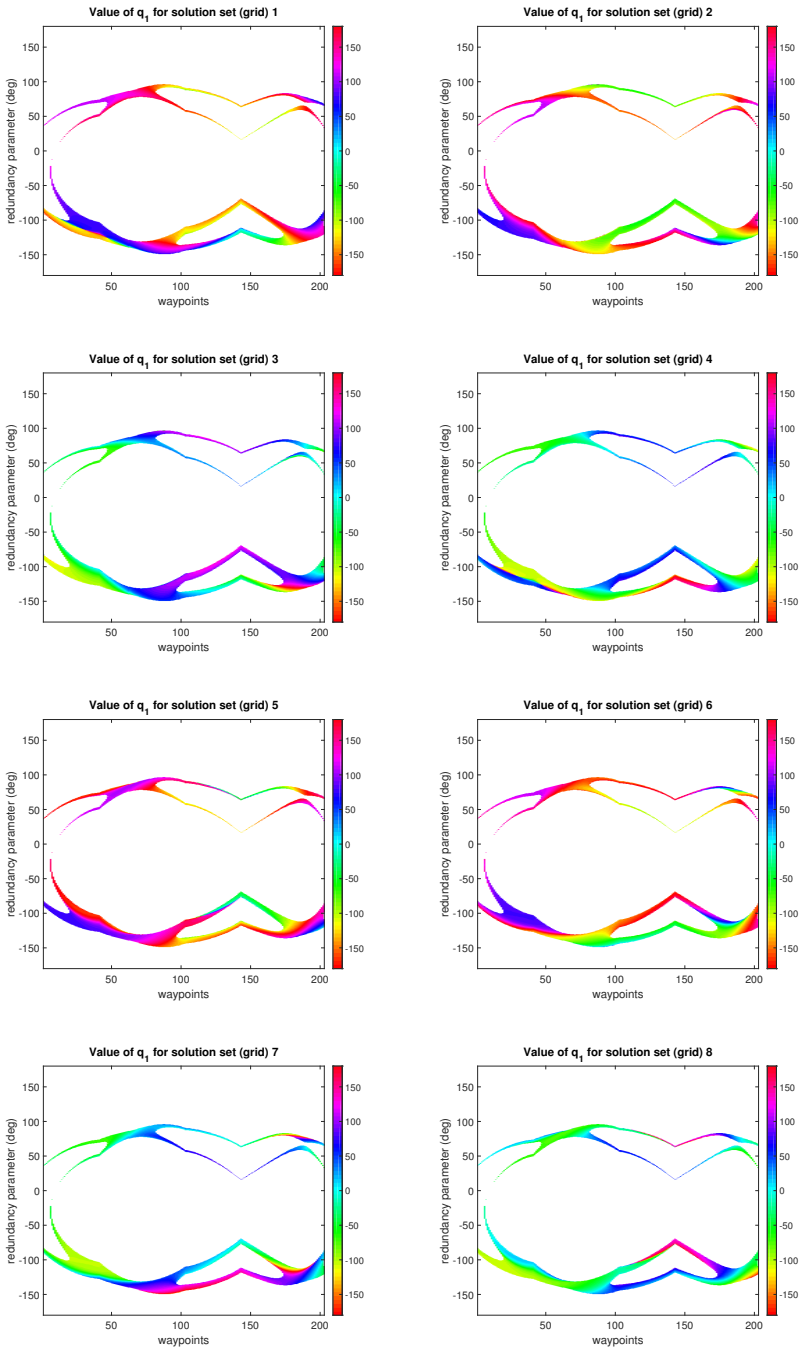


Figure 2.39: Panda grids representing q_1 for the trajectory described in Section 2.8.1 neglecting joint limits

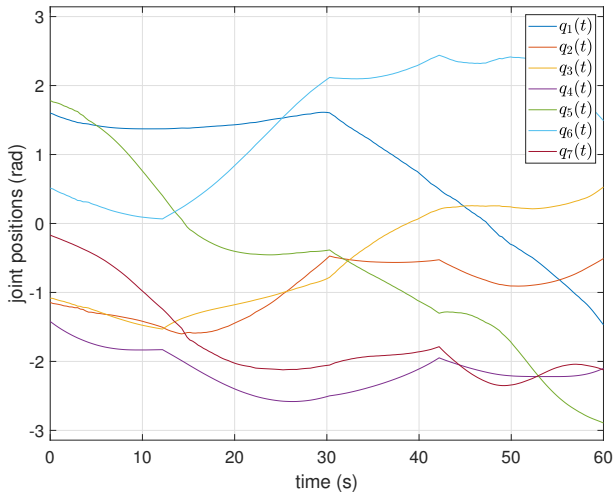


Figure 2.40: Globally-optimal solution for the Panda example

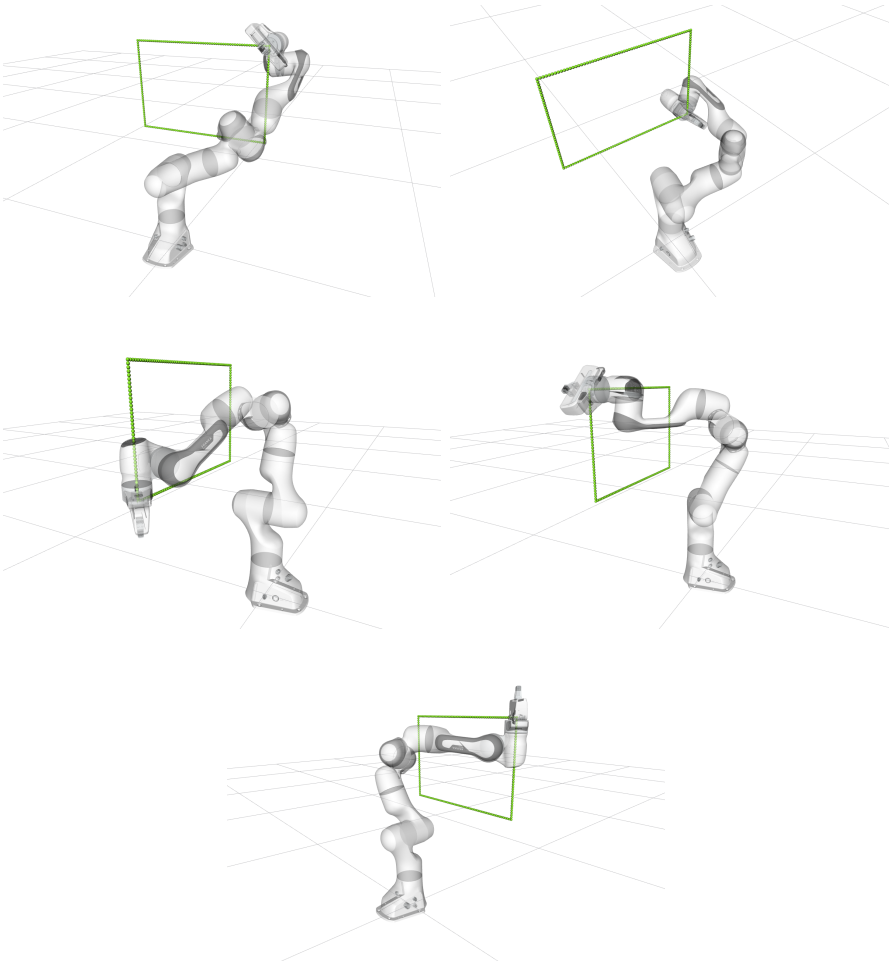


Figure 2.41: Snapshots of the solution of Figure 2.40, corresponding, from left to right and top to bottom, to waypoints $\mathbf{x}_A$, $\mathbf{x}_B$, $\mathbf{x}_C$, $\mathbf{x}_D$ and $\mathbf{x}_E$

Chapter 3

Time-optimal planning of non-redundant robots

3.1 Introduction

A multitude of manufacturing tasks, as well as many other time-critical applications, such as aerial and space ones, require robotic manipulation tasks to be executed in minimum time. In automated workcells or plants, this naturally leads to an increment of the throughput, while in space applications it allows certain tasks to fit within a tight schedule of operations, maximizing the employment of the space asset, whose usage is notably expensive.

Time optimal robotic manipulation tasks can be classified in three categories [56]:

- unconstrained motion between two endpoints;
- partially-constrained motion between two endpoints with the presence of obstacles in the workspace;
- fully-constrained motion when the exact path to be tracked is assigned.

This dissertation focuses on the third category, covering the cases where either a dense set of points in the workspace or some geometric curve are given to describe the path. If the manipulator is not redundant, the relating joint space path can be chosen from a finite set of postures and be optimized with respect to the time optimality criterion, which is to find the optimal actuators control torques allowing to track the path in the shortest time. If the manipulator is redundant, infinite joint space paths exist. The time-optimal control torques can be chosen after the joint space path has been determined, or, the two tasks can be accomplished together, meaning that the choice of the joint space path becomes part of the time-optimal planning process.

In Section 3.2, the time optimization problem is described for non-redundant manipulators. In Section 3.3 some analysis techniques are presented, focusing, in particular, on the properties in the so-called phase plane of the curvilinear coordinate. Several resolution techniques are summarized in Section 3.4, each one based on a different formulation of the problem. In Section 3.5, we introduce a simple use case that allows us to compare some of the methodologies that, in this dissertation, we are mostly interested in, that are the shooting method in the phase plane, addressed in the same section, genetic algorithms, presented in Section 3.6 and dynamic programming, discussed in Section 3.7.

3.2 Problem formulation

3.2.1 Path parametrization

A *path* is given either in the m-dimensional task space or in the n-dimensional joint space, as either $\mathbf{x}(\lambda)$ or $\mathbf{q}(\lambda)$ respectively. $\lambda(t)$ is a parameter varying monotonically with respect to time in $[0, \Lambda]$, i.e. $\lambda(t) : \mathbb{R} \to [0, \Lambda]$ and $\dot{\lambda} = \frac{d\lambda}{dt} > 0 \ \forall t$. The relationship $\lambda(t)$, associating each point of the path to a specific time instant, is unknown and has to be found as the result of the time optimization

problem. Once $\lambda(t)$, representing the *time law*, is known, the *trajectory* $\mathbf{x}(\lambda(t)) = \mathbf{x}(t)$ is uniquely determined.

Let $(\bullet)' = \frac{d(\bullet)}{d\lambda}$ be the λ-derivative of a certain quantity $(\bullet)$. If Λ is chosen as the length of the task space path $\mathbf{x}(\lambda)$, $\mathbf{x}'(\lambda)$ and $\mathbf{x}''(\lambda)$ correspond to the path λ-*velocity* (or *parametric velocity*) and λ-*acceleration* (or *parametric acceleration*).

If the path is directly parametrized with respect to $\lambda(t)$ in the joint space, then the first and second order time derivatives of $\mathbf{q}(t) = \mathbf{q}\big(\lambda(t)\big)$ are given by

$$\dot{\mathbf{q}} = \mathbf{q}'\dot{\lambda} \tag{3.1}$$
$$\ddot{\mathbf{q}} = \mathbf{q}'\ddot{\lambda} + \mathbf{q}''\dot{\lambda}^2 \tag{3.2}$$

which link the joints' velocity and acceleration $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ with their parametric velocity and acceleration $\mathbf{q}'$ and $\mathbf{q}''$.

Rather, if the path is more commonly parametrized in the task space, the inverse kinematics mappings can be used to obtain the joint space path [57]. It is reasonable to assume that the given path is such that the non-redundant manipulator is far from singularities. Under this hypothesis, said $\mathbf{J}(\mathbf{q})$ the Jacobian matrix associated to the manipulator's kinematics, the first and second order inverse kinematic mappings are given by

$$\mathbf{q}' = \mathbf{J}^{-1}\mathbf{x}' \tag{3.3}$$
$$\mathbf{q}'' = \mathbf{J}^{-1}(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') \tag{3.4}$$

The time derivatives of $\mathbf{q}$ can then be obtained by inserting equations (3.3) and (3.4) in (3.1) and (3.2):

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}\mathbf{x}'\dot{\lambda} \tag{3.5}$$
$$\ddot{\mathbf{q}} = \mathbf{J}^{-1}\left[\mathbf{x}'\ddot{\lambda} + \mathbf{x}''\dot{\lambda}^2 - (\mathbf{J}^{-1}\mathbf{x}')^T\frac{\partial \mathbf{J}}{\partial \mathbf{q}}(\mathbf{J}^{-1}\mathbf{x}')\dot{\lambda}^2\right] \tag{3.6}$$

where the λ-derivative $\mathbf{J}'$ in (3.4) has been written as $\mathbf{J}' = \frac{\partial \mathbf{J}}{\partial \mathbf{q}}\mathbf{q}'$ and $\frac{\partial \mathbf{J}}{\partial \mathbf{q}}$ is the Hessian of the direct kinematic function [57].

3.2.2 Robot dynamics parametrization

In the absence of contact and friction forces, the following manipulator dynamic model is given:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{3.7}$$

where $\mathbf{H}(\mathbf{q})$ is the $n \times n$ symmetric positive-definite inertia matrix, $\mathbf{C}(\mathbf{q})$ is the $n \times n \times n$ matrix of centrifugal and Coriolis forces coefficients, $\mathbf{g}(\mathbf{q})$ is the $n \times 1$ vector of gravitational torques and $\boldsymbol{\tau}$ is the $n \times 1$ vector of control torques applied to the manipulator's actuators. Equation (3.7) is a rigid dynamic model, that is commonly employed for a broad range of applications. Nevertheless, for time-optimal motion of real industrial mechanisms, it might not be accurate enough, depending on the required performances. In fact, at fast transitions, especially when high-order derivatives are unconstrained, the elasticity that characterizes the transmission components could no longer be negligible [58]. In the following, for the sake of simplicity, we will assume that (3.7) well describes the system at hand, but the reader should be aware that joint elasticity is an important aspect to consider in order to achieve superior performances with real systems.

By plugging equations (3.1) and (3.2) into (3.7) and recalling that $\mathbf{q}$, $\mathbf{q}'$ and $\mathbf{q}''$ are all functions of λ, the manipulator's dynamic model is parametrized as follows:

$$\mathbf{a}(\lambda)\ddot{\lambda} + \mathbf{b}(\lambda)\dot{\lambda}^2 + \mathbf{g}(\lambda) = \boldsymbol{\tau} \tag{3.8}$$

where

$$\mathbf{a}(\lambda) = \mathbf{H}\big(\mathbf{q}(\lambda)\big)\mathbf{q}'(\lambda) \tag{3.9}$$

$$\mathbf{b}(\lambda) = \mathbf{H}\big(\mathbf{q}(\lambda)\big)\mathbf{q}''(\lambda) + \big(\mathbf{q}'(\lambda)\big)^T\mathbf{C}\big(\mathbf{q}(\lambda)\big)\mathbf{q}'(\lambda) \tag{3.10}$$

If the path is specified in the task space, we can use equations (3.5) and (3.6) in place of (3.1) and (3.2): the parametrization in

(3.8) is unchanged, but $\mathbf{a}$ and $\mathbf{b}$ take the values

$$\mathbf{a}(\lambda) = \mathbf{H}(\mathbf{x})\mathbf{J}^{-1}\mathbf{x}' \tag{3.11}$$

$$\mathbf{b}(\lambda) = \mathbf{H}(\mathbf{x})\mathbf{J}^{-1}\left[\mathbf{x}'' - \left(\mathbf{J}^{-1}\mathbf{x}'\right)^T \frac{\partial \mathbf{J}}{\partial \mathbf{q}}\left(\mathbf{J}^{-1}\mathbf{x}'\right)\right] + \left(\mathbf{J}^{-1}\mathbf{x}'\right)^T \mathbf{C}(\mathbf{x})\left(\mathbf{J}^{-1}\mathbf{x}'\right) \tag{3.12}$$

where the dependence on λ has been omitted to the right-hand side of equations for the sake of clarity of notation.

Alternatively, it could be convenient, especially when adaptive control schemes are adopted, to parametrize the robot dynamics starting from the standard dynamic parameters linear form, as suggested in [59]. Forms of the dynamic model other than (3.7) could also be used, as in [60] and [61], but they always lead to the same parametrization (3.8).

3.2.3 Constraints parametrization

3.2.3.1 Torque limits

Consider each of the rows of the matrix equation (3.8) separately, such that the right-hand term of the equation corresponds to the torque applied to the i-th actuator:

$$a_i(\lambda)\ddot{\lambda} + b_i(\lambda)\dot{\lambda}^2 + g_i(\lambda) = \tau_i \tag{3.13}$$

In real applications, torques are usually characterized by lower and upper bounds $\tau_{i,min}$ and $\tau_{i,max}$, determining the joints maximum deceleration and acceleration respectively. Thus, for each joint, the following inequalities hold:

$$\tau_{i,min} \leq \tau_i \leq \tau_{i,max} \tag{3.14}$$

Assuming that $a_i(\lambda) \neq 0$ and plugging equation (3.13) into (3.14) and solving with respect to $\ddot{\lambda}$, we obtain

$$L_i(\lambda, \dot{\lambda}) \leq \ddot{\lambda} \leq U_i(\lambda, \dot{\lambda}) \tag{3.15}$$

where, by omitting the dependence on λ of the terms on the right-hand side

$$L_i(\lambda, \dot{\lambda}) = \frac{\tau_{i,min}\delta_i + \tau_{i,max}(1-\delta_i) - b_i\dot{\lambda}^2 - g_i}{a_i} \tag{3.16}$$

$$U_i(\lambda, \dot{\lambda}) = \frac{\tau_{i,max}\delta_i + \tau_{i,min}(1-\delta_i) - b_i\dot{\lambda}^2 - g_i}{a_i} \tag{3.17}$$

with

$$\delta_i(\lambda) = \begin{cases} 1 & \text{if} \quad a_i(\lambda) > 0 \\ 0 & \text{if} \quad a_i(\lambda) < 0 \end{cases} \tag{3.18}$$

Since (3.15) has to hold for each i, we can write

$$L(\lambda, \dot{\lambda}) \leq \ddot{\lambda} \leq U(\lambda, \dot{\lambda}) \tag{3.19}$$

where

$$\begin{aligned} L(\lambda, \dot{\lambda}) &= \max_i \{L_i(\lambda, \dot{\lambda})\} \\ U(\lambda, \dot{\lambda}) &= \min_i \{U_i(\lambda, \dot{\lambda})\} \end{aligned} \tag{3.20}$$

In case $a_i(\lambda) = 0$, the i-th actuator does not contribute to $\ddot{\lambda}$ bounds as $L_i(\lambda, \dot{\lambda})$ and $U_i(\lambda, \dot{\lambda})$ are not defined and, thus, do not appear in equations (3.20). However, it contributes bounding $\dot{\lambda}^2$, as will be shown later.

From the practical standpoint, rated torques can be chosen as torque limits in (3.14) and this would be a rather natural choice. However, as pointed out in [62], if the manipulator has to execute a repetitive task continuously, rated torques are likely to exceed the maximum power of the motors and, because of overheating, damage the electrical parts. Rather, if the motor's electric model is considered, it is possible to define more conservative heat-dependent limits, which would allow the manipulator to operate in a safer (and more reliable) condition.

It is also worth remarking that constraints in the form of (3.19) are valid not only for actual motor torques, but, in general, cover any

generalized force constraint. For example, in the case of parallel cable-based manipulators, the same parametrization can be used to ensure that the cables are always tight during the motion [63].

Lastly, equation (3.14) considers fixed torque bounds. However, in most real cases, the torque is dependent on the joint velocity, according to the motor torque characteristic. This condition can be easily accommodated in the framework described in this section, without any major modification [14].

3.2.3.2 Torque rate and jerk limits

In real manipulators, it might be necessary, under certain circumstances that will be discussed later, that limits on joint torques derivatives are considered together with torque limits [64]:

$$\dot{\boldsymbol{\tau}}_{min} \leq \dot{\boldsymbol{\tau}} \leq \dot{\boldsymbol{\tau}}_{max} \tag{3.21}$$

In order for such constraints to be usable, it is necessary to consider the first-order derivative of the manipulator's dynamic model, that is

$$\mathbf{H}(\mathbf{q})\dddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\ddot{\mathbf{q}} + \ddot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \dot{\mathbf{q}}^T\dot{\mathbf{C}}(\mathbf{q})\dot{\mathbf{q}} + \dot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{g}}(\mathbf{q}) = \dot{\boldsymbol{\tau}} \tag{3.22}$$

where

$$\dddot{\mathbf{q}} = \mathbf{q}'''\dot{\lambda}^3 + 3\mathbf{q}''\dot{\lambda}\ddot{\lambda} + \mathbf{q}'\dddot{\lambda} \tag{3.23}$$

and

$$\mathbf{q}''' = \mathbf{J}^{-1}(\mathbf{x}''' - \mathbf{J}''\mathbf{q}' - 2\mathbf{J}'\mathbf{q}'') \tag{3.24}$$

By plugging all the equations in (3.21), it can be demonstrated that the following constraints are obtained [64]:

$$j_{\tau,min}(\lambda, \dot{\lambda}, \ddot{\lambda}) \leq \dddot{\lambda} \leq j_{\tau,max}(\lambda, \dot{\lambda}, \ddot{\lambda}) \tag{3.25}$$

$$\alpha_{\tau,min}(\lambda, \dot{\lambda}) \leq \ddot{\lambda} \leq \alpha_{\tau,max}(\lambda, \dot{\lambda}) \tag{3.26}$$

$$\dot{\lambda} \leq v_\tau(\lambda) \tag{3.27}$$

where j_{min}, j_{max}, $\alpha_{\tau,min}$, $\alpha_{\tau,max}$ and v_τ are the most restrictive bounds obtained from the constraints vector (3.21).

Joint jerk limits of the form

$$\dddot{q}_{i,min} \leq \dddot{q}_i \leq \dddot{q}_{i,max} \tag{3.28}$$

also yield similar constraints on the pseudo-jerk $\dddot{\lambda}$. In fact, by plugging equation (3.24) into (3.23), then (3.23) into (3.28) and solving for $\dddot{\lambda}$, we have

$$j_{q,min}(\lambda, \dot{\lambda}, \ddot{\lambda}) \leq \dddot{\lambda} \leq j_{q,max}(\lambda, \dot{\lambda}, \ddot{\lambda}) \tag{3.29}$$

where $j_{q,min}$ and $j_{q,max}$ represent the strictest bounds for all i.

3.2.3.3 Joint acceleration and velocity limits

Joint acceleration and velocity limits can also be included in a similar manner [65]. In particular, acceleration limits like

$$\ddot{q}_{i,min} \leq \ddot{q}_i \leq \ddot{q}_{i,max} \tag{3.30}$$

lead, as for torques, to constraints like

$$\alpha_{q,min}(\lambda, \dot{\lambda}) \leq \ddot{\lambda} \leq \alpha_{q,max}(\lambda, \dot{\lambda}) \tag{3.31}$$

where $\alpha_{q,min}$ and $\alpha_{q,max}$ are the strictest bounds computed by inserting, for each i, equations (3.2) or (3.6) in (3.30) and solving for $\ddot{\lambda}$.

On the other hand, velocity limits like

$$-\dot{q}_{i,max} \leq \dot{q}_i \leq \dot{q}_{i,max} \tag{3.32}$$

lead to constraints in the form

$$\dot{\lambda} \leq v_q(\lambda) \tag{3.33}$$

where v_q is the minimum between all the quantities computed by inserting, for each i, equations (3.1) or (3.5) in (3.32) and solving for $\dot{\lambda}$.

In case of a discrete-time implementation, as noted in [66], a joint velocity limit can be easily transformed into an acceleration limit. In fact, said Δt the sampling interval and k the discrete-time, by using the Euler approximation, by which

$$\Delta t \ddot{q}_i(k) = \dot{q}_i(k+1) - \dot{q}_i(k) \tag{3.34}$$

constraints (3.32) become

$$\frac{-\dot{q}_{i,max} - \dot{q}_i(k)}{\Delta t} \leq \ddot{q}(k) \leq \frac{\dot{q}_{i,max} - \dot{q}_i(k)}{\Delta t} \tag{3.35}$$

Such a formulation is equivalent to (3.30), yielding to additional constraints in the form of (3.31). Its usefulness must be found in its applicability to real-time resolution of minimum-time trajectories, as mentioned in Section 3.4.1.

3.2.3.4 Task acceleration and velocity limits

Certain tasks, notably welding, cutting and gluing, require high accuracy and, as a consequence, cannot be usually executed at the maximum of the manipulator's capabilities. For this reason, the velocity in the task space is usually bounded in order to meet the accuracy requirements. In pick and place operations, the payload acceleration is usually bounded so as not to exceed the gripper force, which would cause the payload loss before reaching the target destination. For particularly fragile payloads, this is a common issue [67].

Although, in these cases, task velocity and acceleration need to be limited, there still is the need to execute such operations as fast as possible, in order to maximize the throughput of the plant. This leads to time minimization, subject to task space constraints.

For the sake of simplicity, without loss of generality, assume that the path is parametrized in the task space, i.e. $\mathbf{x}(\lambda)$ is available, and that Λ is its length. Under these assumptions, $\mathbf{x}'(\lambda)$ is the unit vector tangent to the path and $\mathbf{x}''(\lambda)$ is a vector normal to

the path and whose magnitude depends on the path curvature. The following properties hold [67]:

$$\|\mathbf{x}'\| = 1 \tag{3.36}$$
$$\mathbf{x}'^T\mathbf{x}'' = 0 \tag{3.37}$$

As for joint velocities and accelerations, we have

$$\dot{\mathbf{x}} = \mathbf{x}'\dot{\lambda} \tag{3.38}$$
$$\ddot{\mathbf{x}} = \mathbf{x}'\ddot{\lambda} + \mathbf{x}''\dot{\lambda}^2 \tag{3.39}$$

Acceleration constraints are usually given as [67]

$$\|\ddot{\mathbf{x}}\| \leq a_{max} = \frac{F_{max}}{m_p} \tag{3.40}$$

where a_{max} is the task-space acceleration limit, obtained from F_{max}, i.e. the maximum gripping force and m_p, i.e. the payload mass.

By inserting (3.39) into (3.40), we find additional limits on $\ddot{\lambda}$ in the form

$$\alpha_{x,min}(\lambda, \dot{\lambda}) \leq \ddot{\lambda} \leq \alpha_{x,max}(\lambda, \dot{\lambda}) \tag{3.41}$$

where

$$\alpha_{x,min}(\lambda, \dot{\lambda}) = \frac{-\mathbf{x}'^T\mathbf{x}''\dot{\lambda}^2 - \sqrt{(\mathbf{x}'^T\mathbf{x}'')^2\dot{\lambda}^4 + \|\mathbf{x}'\|^2 (a_{max}^2 - \|\mathbf{x}''\|^2 \dot{\lambda}^4)}}{\|\mathbf{x}'\|^2}$$
$$\alpha_{x,max}(\lambda, \dot{\lambda}) = \frac{-\mathbf{x}'^T\mathbf{x}''\dot{\lambda}^2 + \sqrt{(\mathbf{x}'^T\mathbf{x}'')^2\dot{\lambda}^4 + \|\mathbf{x}'\|^2 (a_{max}^2 - \|\mathbf{x}''\|^2 \dot{\lambda}^4)}}{\|\mathbf{x}'\|^2} \tag{3.42}$$

By using (3.36) and (3.37), the limits above become [67]

$$\alpha_{x,min}(\lambda, \dot{\lambda}) = -\sqrt{a_{max}^2 - \|\mathbf{x}''\|^2 \dot{\lambda}^4} \tag{3.43}$$
$$\alpha_{x,max}(\lambda, \dot{\lambda}) = +\sqrt{a_{max}^2 - \|\mathbf{x}''\|^2 \dot{\lambda}^4} \tag{3.44}$$

Since the term under square root must be non-negative, the inequalities above also impose a limit on $\dot{\lambda}$:

$$\dot{\lambda} \leq \sqrt{\frac{a_{max}}{\|\mathbf{x}''\|}} \tag{3.45}$$

As far as task velocity constraints are concerned, they can be written as [67]

$$\|\dot{\mathbf{x}}\| \leq v_{x,max} \tag{3.46}$$

which, by using (3.38) and (3.36), can be transformed into

$$\dot{\lambda} \leq v_{x,max} \tag{3.47}$$

3.2.3.5 Process-specific constraints

The constraints analyzed in the previous sections can be classified as generic or cross-application constraints, in that they are dependent on the physical characteristics of the manipulator and its actuation system (3.2.3.1, 3.2.3.2, 3.2.3.3) or on the task velocity and acceleration requirements (3.2.3.4).

However, depending on the specific application, additional constraints may be required. A generic set of constraint can be expressed in the form [68]

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T\mathbf{B}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{c}(\mathbf{q}) \leq 0 \tag{3.48}$$

where, said n_c the number of constraints, $\mathbf{A}(\mathbf{q})$ is an $n_c \times n$ matrix, $\mathbf{B}(\mathbf{q})$ is an $n \times n_c \times n$ tensor and $\mathbf{c}(\mathbf{q})$ is an n_c-dimensional vector. It is easy to show that, by setting

$$\mathbf{A}(\mathbf{q}) = \begin{bmatrix} \mathbf{H}(\mathbf{q}) \\ -\mathbf{H}(\mathbf{q}) \end{bmatrix}, \ \mathbf{B}(\mathbf{q}) = \begin{bmatrix} \mathbf{C}(\mathbf{q}) \\ -\mathbf{C}(\mathbf{q}) \end{bmatrix}, \ \mathbf{c}(\mathbf{q}) = \begin{bmatrix} \mathbf{g}(\mathbf{q}) - \boldsymbol{\tau}_{max} \\ -\mathbf{g}(\mathbf{q}) + \boldsymbol{\tau}_{min} \end{bmatrix} \tag{3.49}$$

one can obtain the torque constraints of Section 3.2.3.1.

Any generic constraint in the form (3.48) yields a generic constraint in λ and its derivatives that can be posed as

$$C(\lambda, \dot{\lambda}, \ddot{\lambda}) \leq 0 \tag{3.50}$$

It is clear that it can be easily accommodated in the problem definition and managed in the same manner as the others [69]. In fact, as long as the constraint can be defined in terms of λ and its derivatives, it always results in limiting either $\dot{\lambda}$ or $\ddot{\lambda}$ or both.

Some examples of process-specific constraints are provided in [69], and many others can be defined as needed. As instance, the contour error prediction can enforce a limited error when describing contours in the workspace. When circular estimation of contour error is used, the model based contour error constraint is defined as

$$\left\| \hat{\mathbf{x}}(\lambda, \dot{\lambda}, \ddot{\lambda}) - \left(\mathbf{x}(\lambda) - \frac{\mathbf{N}(\lambda)}{\kappa(\lambda)} \right) \right\| - \frac{1}{\kappa(\lambda)} \leq E_{max} \tag{3.51}$$

where $\hat{\mathbf{x}}(\lambda, \dot{\lambda}, \ddot{\lambda}) = \mathbf{x}(\lambda) - \hat{\mathbf{e}}(\lambda, \dot{\lambda}, \ddot{\lambda})$, $\hat{\mathbf{e}}(\lambda, \dot{\lambda}, \ddot{\lambda}) \in \mathbb{R}^m$ is the vector of error estimates for each of the task variables, $\kappa(\lambda)$ is the path curvature, $\mathbf{N}(\lambda)$ is the path normal and E_{max} is the maximum allowed error.

Another example falling in this category is the bandwidth limitation of each axis, because of which the command signals are constrained to stay below the frequency tracking ability of individual axes. For each axis, a dominant frequency is observed, which is a function of both the path curvature and the tangential velocity, thus the constraint will be in the form

$$|\dot{\mathbf{x}}(\lambda, \dot{\lambda})| \kappa(\lambda) \leq \omega_{max} \tag{3.52}$$

where ω_{max} is the limit frequency.

An additional example is provided in [14], where the mechanical power limit is considered, i.e.

$$P_{m,min} \leq P_m(\lambda, \dot{\lambda}, \ddot{\lambda}) \leq P_{m,max} \tag{3.53}$$

where $P_m(\lambda, \dot{\lambda}, \ddot{\lambda}) = \boldsymbol{\tau}^T(\lambda, \dot{\lambda}, \ddot{\lambda})\dot{\mathbf{q}}(\lambda, \dot{\lambda})$.

The reader may verify that, by solving the inequalities (3.51), (3.52) and (3.53) with respect to $\ddot{\lambda}$ or $\dot{\lambda}$, the constraints above are equivalent to those analyzed in the previous sections.

More recently, in [70], the same kind of parametrization has been applied in a cooperative grasping scenario to ensure that the normal force applied to a commonly manipulated object remains inside the friction cone in order to ensure stable grasping. Although, in this case, the formulation is slightly more complex, once again, there is no difference with the parametric constraints analyzed here.

3.2.4 Problem formulation with calculus of variations

Let $\mathbf{s}(t) = [\lambda(t), \dot{\lambda}(t)]^T$ be the *state* and $u(t) = \ddot{\lambda}(t)$ the *control input* of a dynamic system. Under the hypothesis that only torque constraints are enforced, the problem of finding the minimum time trajectory can be formalized as follows:

$$\begin{aligned} \min_{u(t)\in[L(\mathbf{s}),U(\mathbf{s})]} \quad & \int_0^{t_f} dt \\ \text{s.t.} \quad & \dot{\mathbf{s}}(t) = \mathbf{f}(\mathbf{s}, u) = [s_2(t)\ u(t)]^T \\ \text{w/ b.c.} \quad & \mathbf{s}(0) = \mathbf{s}_0,\ \mathbf{s}(t_f) = \mathbf{s}_f \end{aligned} \tag{3.54}$$

Necessary conditions for optimality can be found by using the Extended Pontryagin's Maximum Principle (EPMP) [56] which, in addition to the classic PMP formulation, allows to consider the presence of state-dependent constraints on the input in the form of equalities and inequalities. In the case of constraints (3.19), they are

$$u(t) - L\big(\mathbf{s}(t)\big) \geq 0 \tag{3.55}$$

$$U\big(\mathbf{s}(t)\big) - u(t) \geq 0 \tag{3.56}$$

As demonstrated in [56], the optimal control input $u(t) = \ddot{\lambda}(t)$ always takes either its maximum (U) or minimum (L) value, except for a few cases discussed later.

In the case the problem is complicated with additional constraints like (3.31) and/or (3.41), the control may no longer be bang-bang in the torques, but stricter limits may exist preventing the actuators torques from saturating [65, 67].

As demonstrated in [64], the fact that the solution to the time-optimal problem is bang-bang in the torques is of utmost practical relevance. In fact, several drawbacks exist when implementing the bang-bang torque control in real manipulators, such as joint vibrations due to finite joint stiffness and overshoot of the nominal torque limits due to unmodeled actuator (and structure) dynamics. Also, since the trajectory is then executed in closed loop, the controller is likely to have no margin to reduce the errors on the reference values [71]. As a consequence, bang-bang control often leads to poor performances, in terms of both trajectory tracking and execution time. An example of this phenomenon is given in [72].

In adaptive control, torque saturation typically yields tracking errors and significant parameters estimation error. In these cases, trajectories are often planned which do not exploit all the available torque. However, on one hand, the robot performances are conservative while, on the other, the efficiency of the adaption law is limited as the estimation quality of the dynamic parameters increases with growing excitation of the modeled dynamics [59].

In all the real cases, it is then convenient to impose constraints on the torque rates, such as (3.21), or on the jerk, as in [59].

3.3 Analysis in the λ domain

Although using the EPMP necessary conditions may seem attractive to seek after the solution, more efficient techniques exist which

also provide further insights into the analysis of the time-optimal control problem. Such techniques are based upon the analysis of the curves in the $\lambda - \dot{\lambda}$ plane (also referred to as *phase* plane), and in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane.

3.3.1 Properties in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane

In the $\dot{\lambda}^2 - \ddot{\lambda}$ plane, the constraints (3.15), at each λ, are regions limited by a pair of parallel lines representing the torque limits. The intersection of such regions is a polygon, as Figure 3.1 shows, which varies as λ varies. All the pseudo-accelerations inside the polygon are considered *feasible*, in that they guarantee the manipulator to stay on the assigned path. On the contrary, those outside the polygon are *unfeasible*, not because they cannot be physically achieved by the actuators, but because, if commanded, they would drive the end-effector off the path.

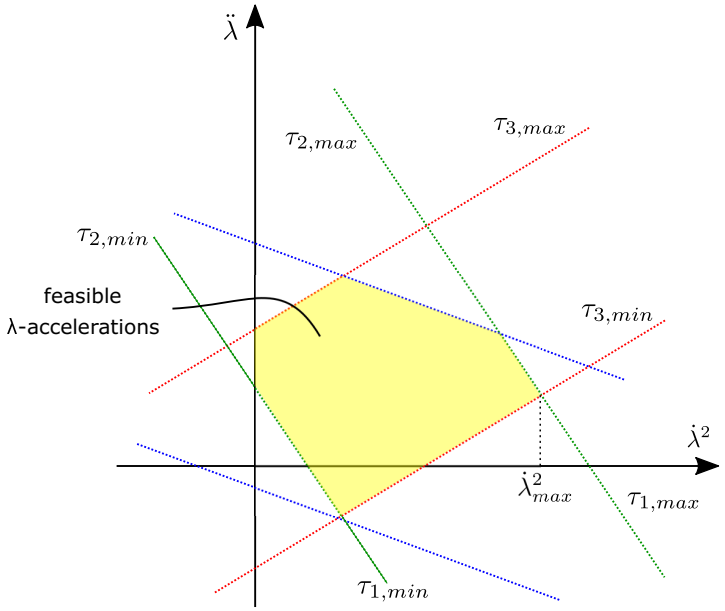


Figure 3.1: Feasible pseudo-accelerations in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane with $L(\lambda, \dot{\lambda}^2_{max}) = U(\lambda, \dot{\lambda}^2_{max})$ and constant λ

The following statements hold:

- as a consequence of the EPMP necessary conditions, if the time has to be minimized, $\ddot{\lambda}$ will always be at either the upper or lower bound of the feasible region;
- $\dot{\lambda}^2$ is limited, to the left, by the $\ddot{\lambda}$ axis, as it cannot be negative;
- $\dot{\lambda}^2$ has a maximum, i.e. $\dot{\lambda}^2_{max}$, at which the condition $L(\lambda, \dot{\lambda}) = \ddot{\lambda} = U(\lambda, \dot{\lambda})$ is satisfied, which means that, when the pseudo-velocity is maximum, only one pseudo-acceleration is allowed.

It is worth remarking that the latter only holds if either $a_i(\lambda) \neq 0 \; \forall i$ or $a_i(\lambda) = 0$, but i is not the bounding actuator, i.e. $\min_j\{U_j\} \neq U_i$. Figure 3.2 shows the case in which $a_i(\lambda) = 0$ and $i = 1$ is the bounding actuator.

In this case, the i-th pair of constraints (3.14) no longer affects the pseudo-acceleration, but only provides the bounds for the pseudo-velocity. Assuming $b_i(\lambda) \neq 0$, we obtain:

$$\frac{\tau_{i,min}\gamma_i + \tau_{i,max}(1-\gamma_i) - g_i}{b_i} \leq \dot{\lambda}^2 \leq \frac{\tau_{i,max}\gamma_i + \tau_{i,min}(1-\gamma_i) - g_i}{b_i} \tag{3.57}$$

where

$$\gamma_i(\lambda) = \begin{cases} 1 & \text{if} \quad b_i(\lambda) > 0 \\ 0 & \text{if} \quad b_i(\lambda) < 0 \end{cases} \tag{3.58}$$

At the maximum pseudo-velocity, i.e. $\dot{\lambda}^2_{max}$, the pseudo-acceleration is no longer unique, which may lead to singular conditions when searching for the optimal control input, as will be discussed later. Also, the condition represented in Figure 3.2 can only happen if the torque constraints constitute straight lines, as in (3.15), or when the constraint is made up of a set of straight segments. If not, the reader may easily verify that there always exists one value of $\ddot{\lambda}$ at $\dot{\lambda}^2_{max}$.

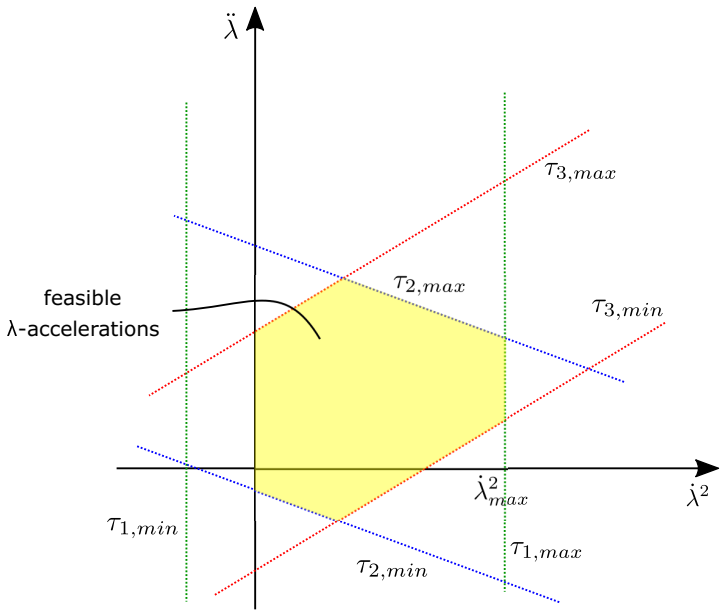


Figure 3.2: Feasible pseudo-accelerations in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane with $L(\lambda, \dot{\lambda}^2_{max}) \neq U(\lambda, \dot{\lambda}^2_{max})$ and constant λ

Lastly, the case $a_i(\lambda) = 0 \wedge b_i(\lambda) = 0$ represents a degenerate case where the inequality (3.14) reduces to

$$\tau_{i,min} \leq g_i(\lambda) \leq \tau_{i,max} \tag{3.59}$$

meaning that no constraint is imposed on $\ddot{\lambda}$ and $\dot{\lambda}^2$ in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane. However it only implies that the available torques at the actuators must be at least sufficient to balance the gravity and does not provide any useful information in the process of seeking the optimal trajectory.

3.3.1.1 Redundantly actuated robots

Computing the feasibility polygon from inequalities (3.15), or, more generally, from (3.14), is interesting for analysis purposes, as will be clear next. However, in certain situations, it is also useful from the practical standpoint. As instance, in the case of

redundantly actuated robots, the dynamic model does not respect equation (3.7), thereby not allowing to proceed with the same parametrization as Section 3.2.2. Nevertheless, by following the developments of [73], the *tree structure*, resulting from cutting at some joints the closed kinematic chain to obtain an open kinematic chain, can still be handled according to Section 3.2.2 [74]. Its kinematic model is:

$$\mathbf{H}(\mathbf{q}_O)\ddot{\mathbf{q}}_O + \dot{\mathbf{q}}_O^T\mathbf{C}(\mathbf{q}_O)\dot{\mathbf{q}}_O + \mathbf{g}(\mathbf{q}_O) = \boldsymbol{\tau}_O \tag{3.60}$$

where the subscript O indicates that the quantities are related to joints belonging to the tree structure (or open kinematic chain). The dynamic model above can then be parametrized as:

$$\mathbf{a}_O(\lambda)\ddot{\lambda} + \mathbf{b}_O(\lambda)\dot{\lambda}^2 + \mathbf{g}_O(\lambda) = \boldsymbol{\tau}_O \tag{3.61}$$

Torque limits in (3.14) are clearly expressed, for redundantly actuated robots, only on actuated joints. Therefore, we extract vector $\boldsymbol{\tau}^A$ of the actuated joint torques from the torque vector $\boldsymbol{\tau}$ and set the limits

$$\tau_{i,min}^A \leq \tau_i^A \leq \tau_{i,max}^A \tag{3.62}$$

The connection between torques in the open chain and those in closed chain is given by

$$\mathbf{S}^T\boldsymbol{\tau}^A = \mathbf{W}^T\boldsymbol{\tau}_O \tag{3.63}$$

where $\mathbf{S}$ and $\mathbf{W}$ are *sensitivity* matrices of the appropriate size, as defined in [73]. By plugging (3.61) into (3.63), we get:

$$\mathbf{S}^T\boldsymbol{\tau}^A = \mathbf{W}^T\mathbf{a}_O(\lambda)\ddot{\lambda} + \mathbf{W}^T\mathbf{b}_O(\lambda)\dot{\lambda}^2 + \mathbf{W}^T\mathbf{g}_O(\lambda) \tag{3.64}$$

In order to define the parametrized dynamics of the closed chain, represented by vectors $\mathbf{a}(\lambda)$, $\mathbf{b}(\lambda)$ and $\mathbf{g}(\lambda)$, and to consequently use the constraints in (3.62), it would be necessary that $\mathbf{S}^T$ was invertible, which is not the case for redundantly actuated robots.

Hence, it is convenient to rewrite constraints (3.62) as:

$$\boldsymbol{\tau}^A \in \mathcal{C} \quad \text{with } \mathcal{C} = \left[\tau_{min,1}^A, \tau_{max,1}^A\right] \times ... \times \left[\tau_{min,n_A}^A, \tau_{max,n_A}^A\right] \tag{3.65}$$

from which, pre-multiplying $\mathbf{S}^T$ to both sides, we get:

$$\mathbf{S}^T \boldsymbol{\tau}^A \in \mathbf{S}^T \mathcal{C} \tag{3.66}$$

By defining $\mathbf{a}^*(\lambda) = \mathbf{W}^T \mathbf{a}_O(\lambda)$, $\mathbf{b}^*(\lambda) = \mathbf{W}^T \mathbf{b}_O(\lambda)$, $\mathbf{g}^*(\lambda) = \mathbf{W}^T \mathbf{g}_O(\lambda)$, $\mathcal{C}^* = \mathbf{S}^T \mathcal{C}$, and plugging (3.64) into (3.66), we have:

$$\mathbf{a}^*(\lambda)\ddot{\lambda} + \mathbf{b}^*(\lambda)\dot{\lambda}^2 + \mathbf{g}^*(\lambda) \in \mathcal{C}^* \tag{3.67}$$

Since $\mathcal{C}$ is a parallelotope and $\mathbf{S}^T$ is linear, $\mathcal{C}^*$ is a convex polytope.

As remarked in [74], by using the polytope projection technique, for each λ, determining a value for $\mathbf{a}^*$, $\mathbf{b}^*$ and $\mathbf{g}^*$, $\mathcal{C}^*$ can be projected onto the $\dot{\lambda}^2 - \ddot{\lambda}$ plane to obtain a polygon like the one of Figure 3.1. One possibility could be to derive the analytic constraints from the feasibility polygon, but it is actually more efficient to work on the polygon directly. For example, one could compute $L(\lambda, \dot{\lambda})$ and $U(\lambda, \dot{\lambda})$ at a given λ_0 and $\dot{\lambda}_0$ by simply intersecting the polygon for $\lambda = \lambda_0$ with the vertical line $\dot{\lambda}^2 = \dot{\lambda}_0^2$. This would allow the computation of minimum and maximum pseudo-accelerations, without any knowledge of the analytic form in (3.16) and (3.17), with a complexity of $O(N_v)$, being N_v the total number of vertices of the polygon.

3.3.2 Properties in the phase plane

3.3.2.1 Phase plane trajectory and maximum velocity curve

The solution to (3.54) can be directly represented as a sequence of points in the $\lambda - \dot{\lambda}$ plane, i.e. the *phase* plane, just like any other trajectory parametrized with respect to λ. Such a continuous sequence of points forms a curve, which will be referred to as *phase plane curve* or *phase plane trajectory* (PPT).

Because of the torque constraints, not all the phase plane curves correspond to feasible trajectories. From the considerations made

in Section 3.3.1 about the presence of a maximum pseudo-velocity at each λ, it is possible to conclude that any feasible phase plane curve will be limited above by the so-called *maximum-velocity curve* (MVC) $\dot{\lambda}_{max}^{(\tau)}(\lambda)$. In case additional constraints are enforced, such as those discussed in sections 3.2.3.2, 3.2.3.3 and 3.2.3.4, a stricter limit usually exists for $\dot{\lambda}$, termed $\dot{\lambda}_{max}(\lambda)$ [75]. In the remainder of this section, we assume that no constraint exists other than (3.19), hence $\dot{\lambda}_{max}(\lambda) = \dot{\lambda}_{max}^{(\tau)}(\lambda)$.

The pseudo-acceleration $\ddot{\lambda} = \frac{d\dot{\lambda}}{d\lambda}\dot{\lambda}$, as per (3.19), always lies in between its maximum and minimum value which could be represented, at each λ, as a pair of arrows in the phase plane (illustrated in Figure 3.3), as they indicate the range of directions for the phase plane curve [76]. In other words, the tangent vector of the phase plane curve will always be in between the maximum acceleration and maximum deceleration direction. The direction of the arrows in Figure 3.3 corresponds to the magnitude of the maximum pseudo-acceleration U and maximum pseudo-deceleration L. When divided by $\dot{\lambda}$, they correspond to the tangent vectors of the phase plane trajectories passing by the origin of the arrows. A three-dimensional representation of the same concept is provided in Figure 3.4.

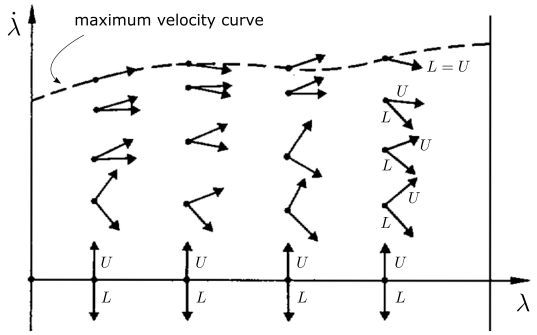


Figure 3.3: Pseudo-accelerations represented in the phase plane with a pair of arrows [76]

At the maximum velocity curve, if no joint satisfying the con-

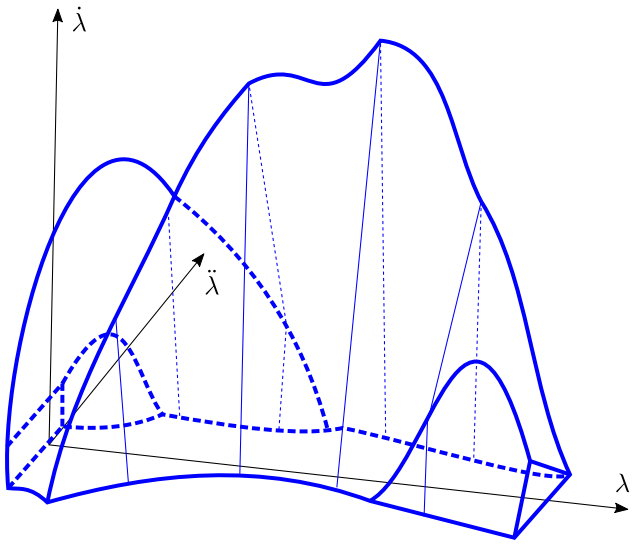


Figure 3.4: Feasibility volume in the space $\lambda - \dot{\lambda} - \ddot{\lambda}$ [60]

dition $a_i(\lambda) = 0$ is bounding the feasible region, only one $\ddot{\lambda} = L(\lambda, \dot{\lambda}_{max}) = U(\lambda, \dot{\lambda}_{max})$ is admissible, leading the jaws of the scissors to close on each other. In this case, only one direction is allowed for the phase plane trajectory when touching the maximum velocity curve.

As far as the minimum time trajectory is concerned, as discussed in Section 3.2, the optimal control input $u(t) = \ddot{\lambda}(t)$ always takes either the maximum or minimum value, which means that the gradient of the phase plane curve is as large as possible, during both acceleration and deceleration phase. This is always true as long as additional constraints limiting $\ddot{\lambda}$, such as those discussed in sections 3.2.3.2, 3.2.3.3 and 3.2.3.4 are not present.

By starting at a certain $\mathbf{s}(0) = \mathbf{s}_0$ with $\dot{\lambda}(0) < \dot{\lambda}_{max}$ (usually $\dot{\lambda}(0) = 0$) and by applying the maximum acceleration $\ddot{\lambda}(t) = U(\lambda, \dot{\lambda})$, the phase plane trajectory may hit the maximum velocity curve at a certain $\lambda = \hat{\lambda}$, before reaching the final state, as Figure 3.5 shows. If the acceleration $\ddot{\lambda} = L(\hat{\lambda}, \dot{\lambda}_{max}) = U(\hat{\lambda}, \dot{\lambda}_{max})$ enters the region above the trajectory curve, it means that the enforcement of such

an input will drive the manipulator off the trajectory. In this case, the control input $\ddot{\lambda} = L = U$ is unfeasible.

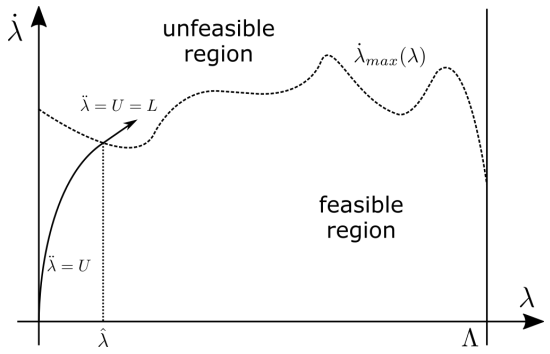


Figure 3.5: Phase plane curve hitting the maximum velocity curve

Thus, in order to keep the phase plane trajectory below the maximum-velocity curve, a switch from maximum acceleration to maximum deceleration has to happen before the curves intersect and, when they do, the maximum control input must point toward the feasible region of the plane. The same behavior could need to be repeated several times before reaching the final state $\mathbf{s}(t_f)$.

Figure 3.6 shows a feasible time-minimum trajectory, where an alternation of maximum acceleration, i.e. $\ddot{\lambda} = U$, and maximum deceleration, i.e. $\ddot{\lambda} = L$, segments leads to the maximization of velocity without crossing the feasible region boundaries. The *switching points* from maximum acceleration to maximum deceleration do not belong to the MVC, while those from maximum deceleration to maximum acceleration do.

While the switching points on the maximum-velocity curve, i.e. λ_1, λ_2 and λ_3 in the example of Figure 3.6, can be found by looking for those points where an entering maximum deceleration curve (*trajectory sink*) and exiting maximum acceleration curve (*trajectory source*) exist at the same time, the others, i.e. λ_a, λ_b, λ_c and λ_d, correspond to the intersection between the trajectory obtained by integrating forward with maximum acceleration and the one obtained by integrating backward with maximum deceleration.

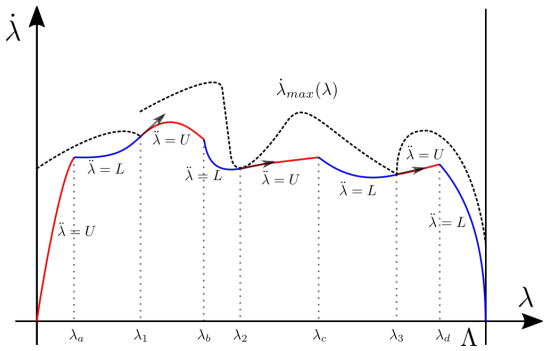


Figure 3.6: Time-minimum trajectory with switching points

As the manipulator cannot track the path any faster than what it can do by following the time-minimum phase trajectory, any other feasible trajectory is limited above by the time-minimum one. In other words, if the manipulator was in a state anywhere in between the time-minimum curve and the maximum-velocity one, it would be forced to hit the maximum-velocity curve and enter the unfeasible region, deviating from the assigned path. This leads to reconsider the feasible region as the portion of the phase plane below the time-minimum curve [57]. All the areas between the time-minimum curve and the maximum-velocity curve are rather classified as *trap regions* [75], as, if the manipulator entered them, it could not exit without violating the acceleration constraints.

An algorithm to compute trap regions is proposed in [62]. The reader may verify that, if trap regions are pre-computed, the time-optimal trajectory can everywhere be chosen as the boundary of such regions, except for the first accelerating segment and the last decelerating segment which link the trap regions boundaries to $\lambda = 0$ and $\lambda = \Lambda$ respectively.

Together with trap regions, *locked regions* may also appear in the phase plane [75]. A region is *locked* if no point $(\lambda, \dot{\lambda})$ exists in the phase plan from which the region can be reached by neither integrating forward along $\ddot{\lambda} = U$, nor integrating backward along $\ddot{\lambda} = L$. For certain resolution techniques, such as dynamic pro-

gramming, these regions can be excluded from the computation, as they are unreachable, speeding up the search of the time-minimum trajectory and saving memory.

Although the curve we have identified by applying the maximum acceleration and deceleration in a bang-bang control fashion is termed "time-minimum trajectory", when dealing with real manipulators, its execution does not necessarily provide the minimum tracking time. In fact, because of unmodelled dynamics and/or actuators saturation, bang-bang trajectories usually result in poor performances in terms of both path tracking error and execution time. In [64, 77], it is demonstrated that it is possible to set torque rate constraints like (3.21) that, although generating lower curves in the phase plane (and, hence, longer planned times), do not saturate the actuators and, at execution, allow for a shorter tracking time than executing theoretical time-minimum trajectories.

3.3.2.2 Calculating the maximum velocity curve

As mentioned above, under the assumption that no joint satisfying the condition $a_i(\lambda) = 0$ is bounding the feasible region in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane, and said $L = L_i$ and $U = U_j$, the maximum velocity curve can be found by imposing

$$L(\lambda, \dot{\lambda}_{max}) = U(\lambda, \dot{\lambda}_{max}) \tag{3.68}$$

which, by using equations (3.16) and (3.17), leads to

$$\dot{\lambda}^2_{max} = \frac{\begin{array}{r} a_i\big(\tau_{j,max}\delta_j + \tau_{j,min}(1-\delta_j) - g_j\big) - \\ -a_j\big(\tau_{i,min}\delta_i + \tau_{i,max}(1-\delta_i) - g_i\big) \end{array}}{a_i b_j - a_j b_i} \tag{3.69}$$

Equivalently, said i and j free indices spanning all the joints, the maximum velocity curve can be computed, as in [78], as

$$\dot{\lambda}^2_{max} = \min_{ij} \left\{ \max_{\tau_i \tau_j} \left\{ \frac{a_j(\tau_i - g_i) - a_i(\tau_j - g_j)}{a_j b_i - a_i b_j} \right\} \right\} \tag{3.70}$$

If $a_i(\lambda) = 0$ and i is the bounding actuator, the maximum velocity curve can be computed from (3.57) as

$$\dot{\lambda}^2_{max} = \frac{\tau_{i,max}\gamma_i + \tau_{i,min}(1-\gamma_i) - g_i}{b_i} \tag{3.71}$$

In order to cover both cases at the same time, more in general, following the analysis in Section 3.3.1, the MVC can be obtained as the solution to the following linear programming problem in the state variables $\ddot{\lambda}$ and $\dot{\lambda}^2$ for each value of λ:

$$\begin{aligned} & \max\{\dot{\lambda}^2\} \\ \text{s.t.} \quad & a_i(\lambda)\ddot{\lambda} + b_i(\lambda)\dot{\lambda}^2 + g_i(\lambda) \leq \tau_{i,max} \\ & a_i(\lambda)\ddot{\lambda} + b_i(\lambda)\dot{\lambda}^2 + g_i(\lambda) \geq \tau_{i,min} \end{aligned} \tag{3.72}$$

The geometric properties in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane, described in Section 3.3.1 are directly exploited in [74] to identify the feasible region without computing the vectors $\mathbf{a}(\lambda)$, $\mathbf{b}(\lambda)$ and $\mathbf{g}(\lambda)$ explicitly. The Authors employ a geometric approach, also used in [79], that iteratively expands the polygon in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane until constraints are satisfied. Eventually, at each λ, the MVC corresponds to the rightmost vertex of the feasibility polygon. This procedure is also advantageous from the computational point of view, as the complexity of finding the vertex of interest is $O(N_v)$, being N_v the total number of vertices.

If the task space trajectory is given analytically, i.e. as a curve equation in the m-dimensional space, the maximum velocity curve can also be calculated analytically. In particular, it can be demonstrated that, in absence of critical arcs (see Section 3.3.2.4), the MVC is made of a sequence of differentiable curves obtained by imposing that two actuators have to saturate at the same time [80]. The method can also be applied in those cases where the actuators models are included in the robot's dynamics and when additional constraints are present (e.g. joints velocities) besides the torque ones. However, it cannot be applied when viscous friction effects (see Section 3.3.3) are considered.

It is clear that, in order for the MVC to exist, the inequality

$$L(\lambda, 0) \leq U(\lambda, 0) \tag{3.73}$$

must hold $\forall\lambda$ [68]. On the contrary, the given path is unfeasible under the assigned torque constraints because the actuator capacities are not even sufficient to balance gravity.

3.3.2.3 Classification of switching points

By using the analysis in the $\lambda - \dot{\lambda}$ plane, the EPMP problem in (3.54) can be reduced to a search of the switching points over the phase plane. Several techniques have been proposed to search for the switching points on the maximum velocity curve in [60], [76] and [81].

Depending on the characteristics of the maximum velocity curve at such points, the following classification can be made [57]:

- *tangent* points are those where the maximum-velocity curve is continuous and differentiable and where the phase trajectory intersects tangentially;
- *critical* points are those where the maximum-velocity curve is continuous, but not differentiable, which can be demonstrated to coincide with the case $a_i(\lambda) = 0$ for the bounding actuator (*zero-inertia* condition)[78];
- *discontinuity* points are those where the maximum-velocity curve is not continuous, which happens when, in the neighborhood of a certain $\lambda = \lambda_1$, said i one of the two bounding actuators, $b_i(\lambda)$ is discontinuous because q_i'' is discontinuous; thus, discontinuity points can be avoided if the geometric path is sufficiently smooth [77].

With reference to Figure 3.6, λ_1, λ_2 and λ_3, are examples of discontinuity, tangent and critical points respectively. Conditions for

which points on the maximum-velocity curve belong to one of the three categories can be determined analytically [57], without the need of computing the maximum-velocity curve, but the belonging to such categories is only necessary for those points to be switching points.

A few additional considerations are now worth to be made about the critical points. As discussed in Section 3.3.1, $a_i(\lambda_3) = 0$ (necessary and sufficient condition to classify λ_3 as a critical point) implies that $\ddot{\lambda}$ is not uniquely determined at $\mathbf{s}_c = [\lambda_3, \dot{\lambda}_{max}]^T$, as $U(\mathbf{s}_c) > L(\mathbf{s}_c)$. In principle, when entering the critical point from the left, $\ddot{\lambda} = L$ should hold while, when exiting the critical point to the right, $\ddot{\lambda} = U$ should hold, as in Figure 3.6. However, sometimes, the direction of U or the opposite of the direction of L may point outside of the feasible region, as depicted in Figure 3.7. This makes it impossible to follow the maximum acceleration to the right and/or the maximum deceleration to the left of the point. When this happens, the critical point is further classified as a *singular* critical point or simply *singular* point [78].

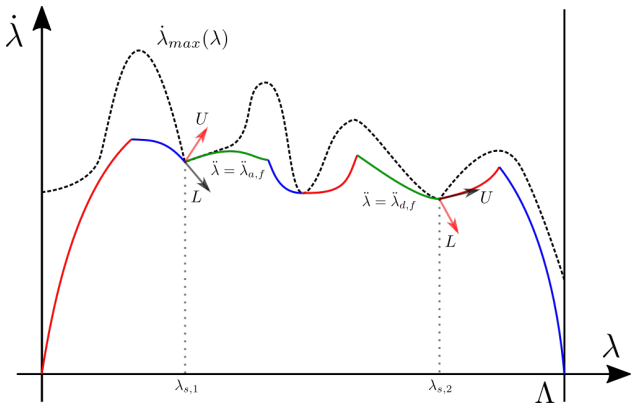


Figure 3.7: Time-minimum trajectory with two singular points

With reference to Figure 3.7, $\lambda_{s,1}$ corresponds to a singular point, as it is a critical point, i.e. $U > L$, and U exits the feasible region. The input $\ddot{\lambda} = U(\lambda_{s,1}, \dot{\lambda}_{max})$ is then unfeasible. Still, $\ddot{\lambda}$ has to be maximized, thus the maximum feasible pseudo-acceleration $\ddot{\lambda}_{a,f}$

shall be chosen and it is immediate to verify that it corresponds to the one along the right-gradient of the maximum velocity curve in $\lambda_{s,1}$ [78], i.e.

$$\ddot{\lambda}_{a,f} = \dot{\lambda}_{max} \frac{d\dot{\lambda}_{max}(\lambda_{s,1}^{+})}{d\lambda} \tag{3.74}$$

Similarly, $\lambda_{s,2}$ also corresponds to a singular point because it is a critical point, i.e. $U > L$, and L enters the feasible region coming from the unfeasible region. The input $\ddot{\lambda} = L(\lambda_{s,2}, \dot{\lambda}_{max})$ is then unfeasible. In order to minimize $\ddot{\lambda}$ while guaranteeing feasibility, it is chosen to be on the direction of the left-gradient of the maximum velocity curve in $\lambda_{s,2}$, i.e.

$$\ddot{\lambda}_{d,f} = \dot{\lambda}_{max} \frac{d\dot{\lambda}_{max}(\lambda_{s,2}^{-})}{d\lambda} \tag{3.75}$$

Both equations (3.74) and (3.75) can be easily obtained from the more general relationship linking $\dot{\lambda}$ with its derivative, by applying the chain rule:

$$\ddot{\lambda} = \frac{d\dot{\lambda}}{dt} = \frac{d\lambda}{dt}\frac{d\dot{\lambda}}{d\lambda} = \dot{\lambda}\frac{d\dot{\lambda}}{d\lambda} \tag{3.76}$$

These conclusions about the singular points demonstrate that, in general, $\ddot{\lambda}$ has to be extremized in agreement with (3.20), but, at singular points, these bounds are not feasible and tighter limits exist.

From the numerical standpoint, it is clear that a critical point cannot exist, as $a_i(\lambda)$ never equals zero exactly. For this reason, said λ_* the critical point, U and L have to be chosen from the left $(-)$ and right $(+)$ neighborhoods of λ_* [78]:

$$\begin{aligned} U(\lambda_*, \dot{\lambda}_{max}) &= \max\{U(\lambda_*^{-}, \dot{\lambda}_{max}), U(\lambda_*^{+}, \dot{\lambda}_{\max})\} \\ &= \max\{L(\lambda_*^{-}, \dot{\lambda}_{max}), L(\lambda_*^{+}, \dot{\lambda}_{\max})\} \\ L(\lambda_*, \dot{\lambda}_{max}) &= \min\{U(\lambda_*^{-}, \dot{\lambda}_{max}), U(\lambda_*^{+}, \dot{\lambda}_{\max})\} \\ &= \min\{L(\lambda_*^{-}, \dot{\lambda}_{max}), L(\lambda_*^{+}, \dot{\lambda}_{\max})\} \end{aligned} \tag{3.77}$$

where the equality in both equations is because $U = L$ at the MVC. Maximum and minimum pseudo-accelerations U and L in the neighborhood of a critical point are represented in Figure 3.8. At the critical point it is clear that $U(\lambda_*, \dot{\lambda}_{max})$ points outside of the feasible region, thus this critical point is also singular.

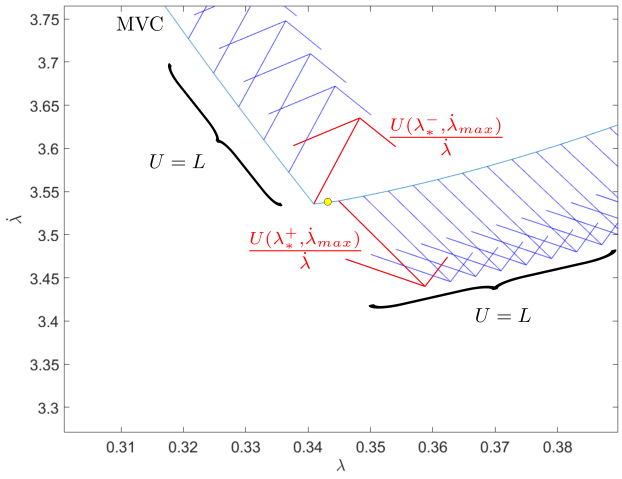


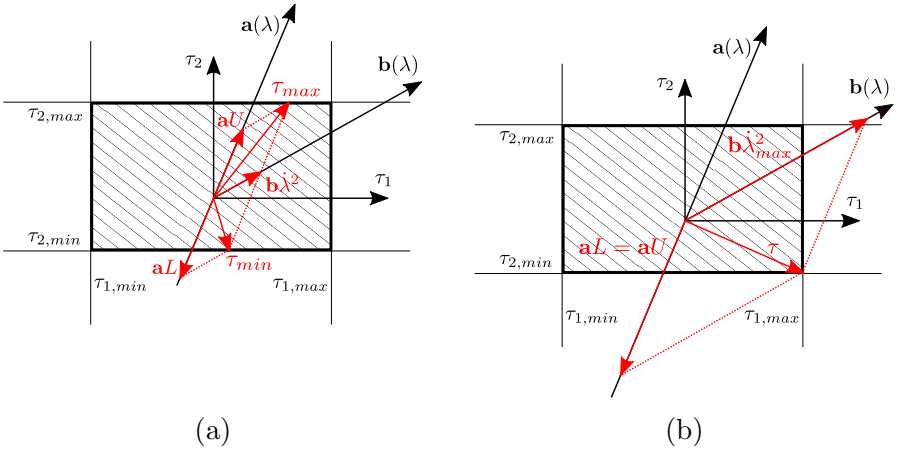
Figure 3.8: Maximum and minimum pseudo-accelerations in the phase plane in the proximity of a critical point, here represented by a yellow circle

The existence of critical points only depends on the mathematical model chosen to represent the torque limits and is not a property of the mechanics of the manipulator-path system. For the sake of clarity, the main findings of [82] are here recalled.

Consider the dynamic model (3.7) and the torque constraints (3.14) and assume, for simplicity, that $\mathbf{g}(\lambda) = 0$:

$$\boldsymbol{\tau}_{min} \leq \mathbf{a}(\lambda)\ddot{\lambda} + \mathbf{b}(\lambda)\dot{\lambda}^2 \leq \boldsymbol{\tau}_{max} \tag{3.78}$$

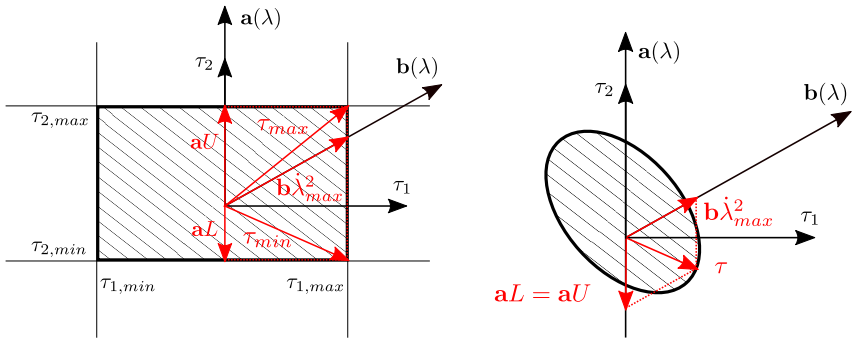
From the geometrical point of view, the torque $\boldsymbol{\tau}$ can be seen as the sum of the vectors $\mathbf{a}(\lambda)\ddot{\lambda}$ and $\mathbf{b}(\lambda)\dot{\lambda}^2$ in the space identified by the admissible torques. If the manipulator is two-jointed, the representation of $\boldsymbol{\tau}$ is as in Figure 3.9.



(a) (b)

Figure 3.9: Geometrical interpretation of control torque off (a) and on (b) the MVC

It is clear that, in case $\mathbf{a}(\lambda)$ becomes parallel to either τ_1, i.e. $a_2(\lambda) = 0$, or τ_2, i.e. $a_1(\lambda) = 0$, when the manipulator is at the MVC, $\boldsymbol{\tau}$ is not uniquely determined, i.e. the point is critical, as shown in Figure 3.10(a).



(a) (b)

Figure 3.10: Geometrical interpretation of a critical point for a polyhedral torque constraint (a) and of a tangent point for a strictly convex torque constraint (b)

Hence, critical points can be generated any time the set of feasible

torques is polyhedral, as in the case of constraints (3.14). On the contrary, if the feasible torques are a strictly convex set, critical points cannot exist, confirming the results of the analysis in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane of Section 3.3.1. A typical case of a strictly convex set, which is also of practical relevance, is a hyper-ellipsoid, i.e.

$$\sum_{i=1}^{n} \left(\frac{\tau_i^2}{\tau_{i,max}^2} \right) \leq 1 \tag{3.79}$$

which is also represented in Figure 3.10(b). Constraints of this kind are used to account for the coupling between the individual actuators when they are driven by a single power supply [82]. If the constraint above is more generally expressed as

$$\sum_{i=1}^{n} \left(\frac{\tau_i^2}{\tau_{i,max}^2} \right)^K \leq 1 \tag{3.80}$$

for $K \to +\infty$, it converges (in the Hausdorff sense) to the constraints (3.14). This means that K can be tuned such that the time-optimal solution is more or less close to the bang-bang one with arbitrary accuracy [83].

3.3.2.4 Critical and singular arcs

Sequences of critical points form *critical arcs*. On critical arcs, the maximum acceleration and deceleration differ for each single point on the arc. If, at such points, the maximum acceleration and/or deceleration are not feasible, the critical arcs are classified as *singular arcs*. Critical arcs occur if the condition $a_i(\lambda) = 0$, with i corresponding to the bounding actuator, is satisfied for a contiguous sequence of values of λ. For a two-link manipulator, the zero-inertia condition is satisfied along lines in the two-dimensional joint space. If the assigned path follows, in the joint space, one of the zero-inertia lines, and the zero-inertia joint is also the bounding joint, then the manipulator will be forced to track a critical arc. Figure 3.11 depicts a critical arc for such a two-link manipulator following a circular path centered in the manipulator's base [78].

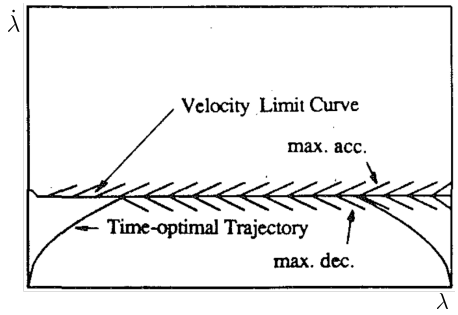


Figure 3.11: Time-optimal trajectory along a singular arc [78]

As shown in Figure 3.11, when a singular arc exists, the time-minimum trajectory simply follows the maximum-velocity curve for the duration of the arc, or until deceleration is necessary to maintain the end-effector on the path. It is worth noticing that singular arcs satisfy the condition

$$\frac{L(\lambda, \dot{\lambda}_{max})}{\dot{\lambda}_{max}} \leq \frac{d\dot{\lambda}_{max}}{d\lambda} \leq \frac{U(\lambda, \dot{\lambda}_{max})}{\dot{\lambda}_{max}} \tag{3.81}$$

that is, the inclination of the maximum velocity curve is always in between the maximum acceleration and the maximum deceleration. If this condition is satisfied $\forall \lambda \in [0, \Lambda]$, then only one singular arc exists and the time-optimal curve $\dot{\lambda}^*(\lambda)$ will be [84]

$$\dot{\lambda}^*(\lambda) = \begin{cases} \dot{\lambda}_U & \lambda \in [0, \lambda_1] \\ \dot{\lambda}_{max} & \lambda \in [\lambda_1, \lambda_2] \\ \dot{\lambda}_L & \lambda \in [\lambda_2, \Lambda] \end{cases} \tag{3.82}$$

where $\dot{\lambda}_U$ and $\dot{\lambda}_L$ are the pseudo-velocities computed by integrating with maximum acceleration and deceleration respectively, and λ_1 and λ_2 are the points at which the time-optimal curve reaches and leaves the maximum velocity curve respectively, as shown in Figure 3.11.

As far as actuators are concerned, it was demonstrated in [56] that *one and only one* actuator saturates on any finite time/path

interval along the time optimal trajectory, except for the following two cases, where *at least one* actuator saturates:

- when an acceleration is chosen at the intersection of the lines in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane, which covers both the cases $\dot{\lambda}^2 < \dot{\lambda}^2_{max}$ (i.e. solution below the maximum-velocity curve) and $\dot{\lambda}^2 = \dot{\lambda}^2_{max}$ (i.e. solution on the maximum-velocity curve);
- when, at singular points or singular arcs, two or more lines bounding $\dot{\lambda}^2$, parallel to the $\ddot{\lambda}$ axis, are coincident with each other.

3.3.3 Viscous friction effects and state-dependent torques

Consider to modify the manipulator's dynamic model to include the actuators viscous friction and to account for state-dependent control torques. Equation (3.7) then becomes:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{B}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \mathbf{v} + \mathbf{K}\dot{\mathbf{q}} \tag{3.83}$$

where $\mathbf{B}$ is the $n \times n$ diagonal matrix of viscous friction coefficients, $\mathbf{v}$ is the n-dimensional vector of actuators input voltages and $\mathbf{K}$ is the $n \times n$ diagonal matrix of coefficients which, depending on the motor, may account for motor winding resistance, voltage source resistance and back E.M.F. generated by the motor [76]. All the other terms are unchanged with respect to the dynamic model of equation (3.7).

By plugging equations (3.1) and (3.2) into (3.83) and recalling that $\mathbf{q}$, $\mathbf{q}'$ and $\mathbf{q}''$ are all functions of λ, the manipulator's dynamic model is parametrized as follows:

$$\mathbf{a}(\lambda)\ddot{\lambda} + \mathbf{b}(\lambda)\dot{\lambda}^2 + \mathbf{c}(\lambda)\dot{\lambda} + \mathbf{g}(\lambda) = \mathbf{v} \tag{3.84}$$

where $\mathbf{a}(\lambda)$ and $\mathbf{b}(\lambda)$ are as in Section 3.2.2 and $\mathbf{c}(\lambda) = (\mathbf{B} - \mathbf{K})\mathbf{q}'(\lambda)$.

Similarly to Section 3.2.3.1, each of the rows of the matrix equation above can be considered separately, such that the right-hand term of the equation corresponds to the voltage applied to the i-th actuator:

$$a_i(\lambda)\ddot{\lambda} + b_i(\lambda)\dot{\lambda}^2 + c_i(\lambda)\dot{\lambda} + g_i(\lambda) = v_i \tag{3.85}$$

If the motor, as in real cases, is characterized by bounded input voltages, one can write:

$$v_{i,min} \leq v_i \leq v_{i,max} \tag{3.86}$$

Assuming that $a_i(\lambda) \neq 0$ and plugging equation (3.85) into (3.86) and solving with respect to $\ddot{\lambda}$, we obtain equation (3.15) again, but the boundaries are re-defined as

$$L_i(\lambda, \dot{\lambda}) = \frac{v_{i,min}\delta_i + v_{i,max}(1-\delta_i) - b_i\dot{\lambda}^2 - c_i\dot{\lambda} - g_i}{a_i} \tag{3.87}$$

$$U_i(\lambda, \dot{\lambda}) = \frac{v_{i,max}\delta_i + v_{i,min}(1-\delta_i) - b_i\dot{\lambda}^2 - c_i\dot{\lambda} - g_i}{a_i} \tag{3.88}$$

All the considerations made in Section 3.3.2 hold here as well, except that bounds (3.87) and (3.88) now represent parabolae, instead of lines, in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane. They may cause the presence of inadmissible regions inside the admissible region of the phase plane, called *inadmissible islands* or just *islands* [76], as Figure 3.12 shows.

When islands exist, forward or backward integration shall be stopped before the trajectory enters the islands. This implies that tangent switching points can be found which belong to the boundaries of the islands. It requires the necessary condition for tangentiality to be modified accordingly, as discussed in [57].

3.3.4 Properties in the three-dimensional phase space

If constraints on the torque rates (see Section 3.2.3.2) are imposed, the problem could still be addressed in the phase plane, as done

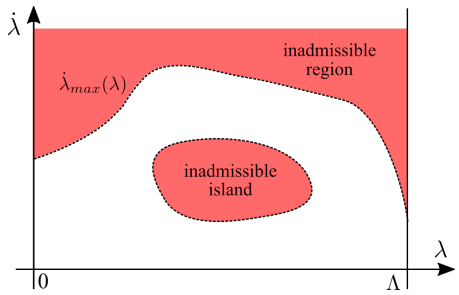


Figure 3.12: Phase plane with highlighted the inadmissible region (above the velocity limit curve) and an inadmissible island

in [64]. However, the methodology developed therein is prone to sub-optimal solutions. Therefore, in [85, 77], the problem is directly addressed in the space $\lambda - \dot{\lambda} - \ddot{\lambda}$, here referred to as *three-dimensional phase space*, where the variable $\dddot{\lambda}$ is termed *pseudo-jerk*.

In [85], the authors demonstrate that the time-optimal profile still has a bang-bang structure in the pseudo-jerk, while in [77], a more robust implementation is proposed to cope with the generation of the optimal solution even in presence of singularities.

With reference to (3.25), the phase space trajectory of Figure 3.13 can be computed by integrating forward along the maximum jerk field, starting at $(0, \dot{\lambda}(0), \ddot{\lambda}(0))$ and backward along the maximum jerk field, starting at $(\Lambda, \dot{\lambda}(\Lambda), \ddot{\lambda}(\Lambda))$. Then a minimum-jerk field has to be found to connect the two profiles computed before: a techniques is proposed in [77]. The control then has a j_{max}-j_{min}-j_{max} structure. It is important to remark that the minimum-jerk field connecting the two maximum-jerk profiles could be not unique, and each of them may have a different cost, thus the bang-bang control is only a necessary condition for global optimality, but not sufficient.

In the two-dimensional case, the torque constraints constitute lower and upper bounds for the pseudo-acceleration and an upper bound for the pseudo-velocity, yielding the MVC. Likewise,

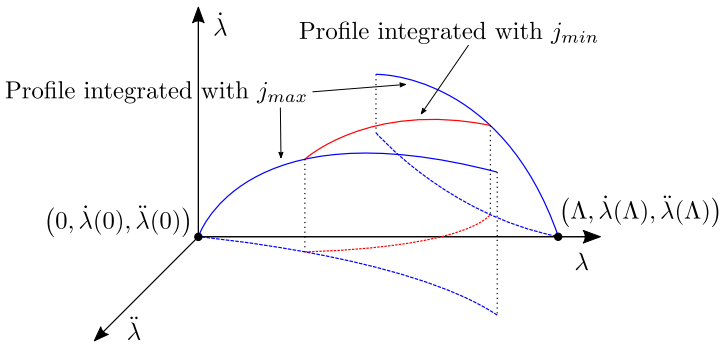


Figure 3.13: Time-optimal phase space trajectory with two maximum jerk segments (blue) and one connecting minimum jerk segment (red); projections onto the $\lambda - \ddot{\lambda}$ plane are represented by dashed lines

in the three-dimensional case, the torque rate constraints provide the lower and upper bounds for the pseudo-jerk and, in addition, both a lower and an upper bound for the pseudo-acceleration, as evident from (3.26). The minimum and maximum pseudo-accelerations generated by torque rate constraints, i.e. $\alpha_{\tau,min}(\lambda, \dot{\lambda})$ and $\alpha_{\tau,max}(\lambda, \dot{\lambda})$ respectively, constitute surfaces in the phase space, termed *Minimum Acceleration Surface* (MiAS) and *Maximum Acceleration Surface* (MaAS) respectively [77].

Like in the phase plane, where we have identified tangent, critical and discontinuity points, in [77], it is argued that the same classification is possible in the three-dimensional phase space, but only critical points (and related features) are analyzed. Similarly to the two-dimensional case, switching points always lie on the MiAS and the MaAS.

The condition triggering critical points in the phase space is the same that triggers critical points in the phase plane, i.e. $a_i(\lambda) = 0$, where i is the bounding actuator. In [77], the authors demonstrate that such points form curves on the MiAS and the MaAS and are

therefore termed *critical curves*[1]. On the MiAS and the MaAS, the maximum and minimum pseudo-jerks are equal everywhere, except on the critical curves.

When a phase space profile approaches a critical curve, it suddenly terminates, and this is similar to the divergence that maximum and minimum acceleration fields expose in the vicinity of critical points in the two-dimensional case (see Section 3.5.1). This requires to "extend" the profiles integrated with maximum jerk through these critical curves. A technique is proposed in [77], where the authors also demonstrate that, in this case, the control is no longer j_{max}-j_{min}-j_{max}, but j_{max}-j_{min}-j_{max}-...-j_{min}-j_{max} and that a critical curve is connected to a forward maximum-jerk profile through a backward minimum-jerk profile, i.e. the j_{min}-to-j_{max} switching point lies exactly on the *maximum critical curve* (critical curve on the MaAS).

3.4 Resolution techniques

3.4.1 Resolution by identification of switching points

In Section 3.3.2.3, we have seen that one method to solve problem (3.54) is by classifying the switching points and integrating forward and backward by using either the maximum acceleration/deceleration or equations (3.74) and (3.75) for singular points.

The algorithms proposed in previous researches, falling in this category, mainly differ in the way the switching points are identified, providing more or less efficiency and robustness with respect to singular points and singular arcs.

In [81], an algorithm is proposed which, starting at $\lambda = 0$, inte-

[1] here we prefer the term *critical* over *singular*, that is used in [77], because of the analogy with critical points in the two-dimensional case, that are not always singular

grates forward with maximum acceleration, until a point on the maximum velocity curve is reached at λ_a. Here, with $\lambda = \lambda_a$ fixed, $\dot{\lambda}$ is progressively decreased and forward integration with maximum deceleration is performed starting from such points until the maximum velocity curve is intersected tangentially, using a try-and-error approach. The point of intersection is recognized as a switching point from which the same procedure can be repeated again. This algorithm requires the maximum velocity curve to be computed explicitly and, in its original version, fails at singular points, as no acceleration other than the minimal and maximal one is allowed.

A similar algorithm is proposed in [76], where, rather than using a try-and-error approach from $\lambda = \lambda_a$, a search is performed on the maximum velocity curve, until the gradient of the time-optimal curve is lower than that of the maximum velocity curve, i.e. a trajectory source is found. This algorithm is slightly more efficient, as less points are calculated to be discarded at a later stage. In [60], the same algorithm is essentially used, with a minor difference in the parametrization of the path. Even in this case, the maximum velocity curve has to be computed explicitly and singular points are not accounted for.

The first algorithm not needing the computation of the maximum velocity curve is proposed in [57], where the classification presented in Section 3.3.2.3 yields the necessary conditions for switching points, which are identified before the time-minimum curve is actually integrated. This method further reduces the amount of points that are computed uselessly and, with a minor modification, can account for singular points.

3.4.1.1 Real-time resolution

A variant of the algorithm presented in [81], is proposed in [66], which is also suitable for an online implementation. The algorithm can manage, similarly to others, joint velocity, acceleration and torque constraints, but joint velocity limits, yielding, in the

continuous time domain, constraints on the pseudo-velocity $\dot{\lambda}$, are transformed into constraints on the pseudo-acceleration $\ddot{\lambda}$, as discussed in Section 3.2.3.3. In this way, pseudo-velocity constraints are guaranteed to be respected on the basis of the satisfaction of pseudo-acceleration constraints. Also, the algorithm does not require the explicit calculation of the MVC, as it can be identified by the condition $L > U$, on a step-by-step basis. Since the implementation is discrete, critical points exist with zero probability.

The algorithm of [66] is similar to [81] in that it is based on the same try-and-error approach: at each iteration along a maximum acceleration profile, an integration with maximum deceleration is performed. If the phase plane trajectory can reach the condition $\dot{\lambda}(\lambda) = 0$ for $\lambda < \Lambda$, without meeting the condition $L > U$, the integration can continue with maximum pseudo-acceleration U, i.e. the phase plane trajectory is still in the feasible region. Otherwise, the next state $\mathbf{s} = [\lambda, \dot{\lambda}]^T$ is in a trap region and, according to the arguments of Section 3.3.2, is already unfeasible. In this case, the integration must proceed along the maximum deceleration field.

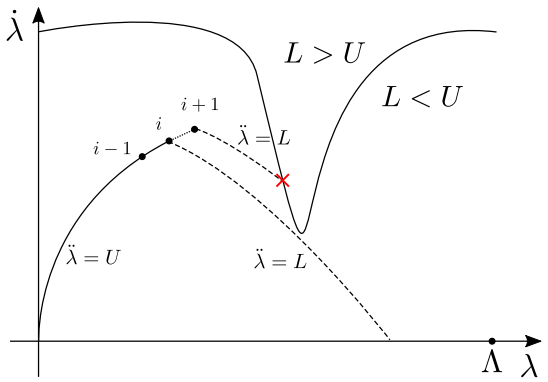


Figure 3.14: On-line resolution through identification of switching points

With reference to Figure 3.14, the phase plane trajectory is being integrated along the maximum acceleration field, i.e. $\ddot{\lambda} = U$. At iteration $i - 1$, the state at the next iteration i is computed with $\ddot{\lambda} = U$ and, from this, an integration along the maximum decel-

eration field is attempted, while the condition $L < U$ is verified at each step. The decelerating phase plane trajectory meets the λ-axis before Λ, meaning that the integration along the maximum acceleration field can continue. At the following iteration, i.e. i, the same procedure is repeated for the next state at iteration $i+1$. This time, the integration along the maximum deceleration field stops because $L > U$, i.e. if integrated from the state in $i+1$, the phase plane trajectory would hit the MVC other than tangentially. The state at iteration $i+1$ is unfeasible and must be discarded. The integration continues along the maximum deceleration field computed at iteration i. A dual procedure is followed when integrating with $\ddot{\lambda} = L$, up until $\lambda = \Lambda$ is reached.

This approach, coupled with a dynamic modulation of the integration interval, allows for an online implementation, and possibly supports re-planning in case of modifications of the geometric path. On the other hand, the coarse discretization that is sometimes necessary to reduce the computation time may yield to non-smooth phase plane trajectories, which, in turn, provoke unacceptable jitters on the actuation torques.

A more relaxed formulation for online time-optimal planning is provided in [58]. Therein, the classical bang-bang integration in the phase plane is coupled with the generation of trapezoidal velocity profiles (TVP) along short segments. Several segments can then be connected to form longer trajectories and allow for replanning, whenever necessary. The usage of TVPs allows to simplify the computation of phase-plane trajectories with non-zero boundary conditions and greatly reduces the computation time, making the system more reactive to unplanned events. On the other hand, solutions are only nearly optimal and performances tend to degrade near singular configurations or for longer path segments.

3.4.2 Resolution by area maximization

A different approach, which does not consider switching points explicitly, but still exploits the properties in the phase plane, is

proposed in [86]. The algorithm has been termed *perturbation trajectory improvement algorithm* (PTIA) as the time-minimum curve is calculated iteratively, by "perturbating" an initial non-optimal feasible solution.

The underlying idea is that bringing the phase plane trajectory as close as possible to the maximum velocity curve is equivalent to maximizing the area underneath. By using a combination of gradient and binary search techniques, the area underneath the trivial solution, e.g. the phase plane trajectory $\dot{\lambda}(\lambda) = 0 \ \forall \lambda \in [0, \Lambda]$, is progressively increased at each iteration.

The domain of λ is discretized and for each sample, at each stage, the pseudo-velocity is increased of the current increment only if the local acceleration constraints are respected, otherwise the increment is halved and the check is repeated. The algorithm terminates when no increment of $\dot{\lambda}$ is possible for any of the discrete values of λ. Like other algorithms examined before, the maximum velocity curve has to be available in order to determine which the maximum velocity increment is at each stage.

Unlike dynamic programming, which will be addressed later, PTIA requires relatively little memory, while the computation time is not negligible and tied to the resolution of the discretized λ-domain. In fact, the time increases as the square of the number of λ-intervals. On the other hand, choosing a small number of intervals would negatively affect the accuracy of the solution. However, given the high degree of *locality*, the algorithm can be easily parallelized, which would be advantageous in case more CPUs are available.

When equating the area maximization and the time minimization, the reader should be aware that the two objectives are not exactly the same and pursuing one or the other may yield different results. It is easy to provide a demonstration in discrete time. In this case, assuming a fixed sampling interval for λ of amplitude $\Delta\lambda$, said k the index identifying such intervals and N_i the total number of

intervals, the overall tracking time can be written as:

$$t = \Delta\lambda \sum_{k=1}^{N_i} \frac{1}{\dot{\lambda}_k} \tag{3.89}$$

On the other hand, the area underneath the phase-plane curve, in the discrete domain, with the same assumptions as above, is:

$$A = \Delta\lambda \sum_{k=1}^{N_i} \dot{\lambda}_k \tag{3.90}$$

By making the sum in (3.89) explicit, the same equation can also be written as:

$$t = \frac{\Delta\lambda \sum_{k=1}^{N_i} \dot{\lambda}_k}{\prod_{k=1}^{N_i} \dot{\lambda}_k} = \frac{A}{\prod_{k=1}^{N_i} \dot{\lambda}_k} \tag{3.91}$$

From the equation above, it is evident that more than one value of t can be obtained for the same value of A, meaning that the area maximization does not correspond, in general, to time minimization. For example, let us assume $N_i = 2$ and $\Delta\lambda = 0.02$. In a first case, let us also assume $\dot{\lambda}_1 = 1$ and $\dot{\lambda}_2 = 2$. The area is

$$A = 0.02(1 + 2) = 0.06 \tag{3.92}$$

Then, let us assume $\dot{\lambda}_1 = 1.5$ and $\dot{\lambda}_2 = 1.5$. Even in this case, the area is

$$A = 0.02(1.5 + 1.5) = 0.06 \tag{3.93}$$

For the first case, the time is

$$t = \frac{0.06}{1 \cdot 2} = 0.03 \tag{3.94}$$

while for the second case, the time is

$$t = \frac{0.06}{1.5 \cdot 1.5} = 0.02\overline{6} \tag{3.95}$$

3.4.3 Resolution by direct integration

An alternative method to find the time-optimal curve is by direct integration. The differential equations to be integrated have been determined in [84], by extending the well-known comparison principle [87] to state-constrained differential inequalities, where the state constraint is precisely given by the maximum velocity curve.

The differential equations are:

$$\frac{dy}{d\lambda} = \begin{cases} y_M(\lambda, \dot{\lambda}) & y < \beta(\lambda) \\ \min\left\{ y_M(\lambda, \dot{\lambda}), \dfrac{d\beta(\lambda)}{d\lambda} \right\} & y \geq \beta(\lambda) \\ w/\ b.c.\ y(0) = 0 \end{cases} \tag{3.96}$$

$$\frac{dy}{d\lambda} = \begin{cases} y_m(\lambda, \dot{\lambda}) & y < \beta(\lambda) \\ \max\left\{ y_m(\lambda, \dot{\lambda}), \dfrac{d\beta(\lambda)}{d\lambda} \right\} & y \geq \beta(\lambda) \\ w/\ b.c.\ y(\Lambda) = 0 \end{cases} \tag{3.97}$$

where $y = |\dot{\lambda}|^2$, $y_m(\lambda, \dot{\lambda}) = 2L(\lambda, \dot{\lambda})$, $y_M(\lambda, \dot{\lambda}) = 2U(\lambda, \dot{\lambda})$ and $\beta(\lambda) = |\dot{\lambda}_{max}|^2$.

Differential equations (3.96) and (3.97) can be integrated by applying directly any well-known numerical integration method (e.g. Euler, Runge-Kutta), yielding solutions $y_F(\lambda)$ and $y_B(\lambda)$ respectively.

Since $\dot{\lambda}_F = \sqrt{y_F}$ and $\dot{\lambda}_B = \sqrt{y_B}$, the time-optimal trajectory is computed as:

$$\dot{\lambda}^*(\lambda) = \min\left\{ \dot{\lambda}_F, \dot{\lambda}_B \right\} \tag{3.98}$$

A discrete implementation of the same concept is provided in [69], where a double-scan search algorithm is designed and demonstrated for several paths and constraints. The λ and $\dot{\lambda}$ axes are discretized. For each value of λ, from $\lambda = 0$ to $\lambda = \Lambda$ (forwards), the maximal value of $\dot{\lambda}$ is found which locally satisfies all the constraints, using the maximum acceleration. This results in a phase

plane trajectory made of feasible and unfeasible (i.e. belonging to a trap region) segments. Then, for each value of λ, from $\lambda = \Lambda$ to $\lambda = 0$ (backwards), the maximal value of $\dot{\lambda}$ is found which locally satisfies all the constraints, using the maximum deceleration. When the backward phase plane trajectory intersects the forward phase plane trajectory, the latter is followed until entering a trap region, where the maximum deceleration is used again. The backward trajectory obtained with this technique corresponds to the time-optimal one and, indeed, is equivalent to considering the minimum between $\dot{\lambda}_F$ and $\dot{\lambda}_B$, as in (3.98). A pictorial view is given in Figure 3.15. The backward integration profile is drawn entirely in order to show the effect of the minimization in (3.98), however, in [69], the forward integration profile is followed backwards at intersections with the backward profile.

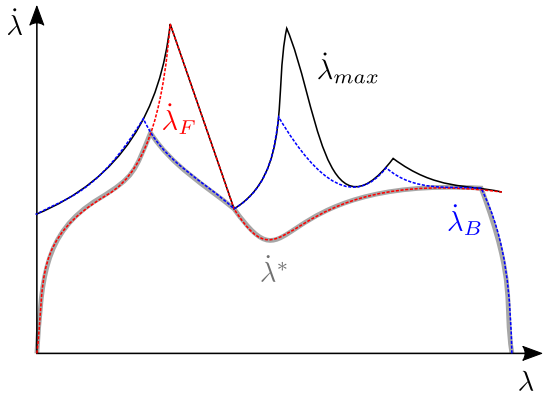


Figure 3.15: Time-optimal planning through direct integration

With respect to the solution obtained by integrating the differential equations (3.96) and (3.97), the algorithm in [69] does not require the computation of the maximum velocity curve, can accommodate several constraints more easily and is able to find time-optimal trajectories even in the presence of inadmissible islands (see Section 3.3.3) with only one more scan per island. On the other hand, the solution accuracy will strictly depend on the discretization step.

3.4.4 Resolution by convex optimization

A completely different approach, which does not make use of any of the properties mentioned in Section 3.3, consists in transforming a possibly more general form of the problem (3.54)-(3.56) into a convex optimization problem [88]. This is done by setting the non-linear change of variables

$$A(\lambda) = \ddot{\lambda} \tag{3.99}$$
$$B(\lambda) = \dot{\lambda}^2 \tag{3.100}$$

and imposing the additional constraints:

$$B'(\lambda) = 2A(\lambda) \tag{3.101}$$
$$B(\lambda) \geq 0 \tag{3.102}$$

Problem (3.54)-(3.56) is then reformulated as:

$$\begin{aligned} \min_{A,B,\boldsymbol{\tau}} \quad & \int_0^{\Lambda} \frac{1}{\sqrt{B(\lambda)}} d\lambda \\ \text{s.t.} \quad & \boldsymbol{\tau} = \mathbf{a}(\lambda)A(\lambda) + \mathbf{b}(\lambda)B(\lambda) + \mathbf{g}(\lambda) \\ & \boldsymbol{\tau}_{min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max} \\ & B'(\lambda) = 2A(\lambda) \\ & B(\lambda) \geq 0 \\ & B(0) = B(\Lambda) = 0 \end{aligned} \tag{3.103}$$

It is important to remark that, in order for the problem to be convex, this approach has some important limitations. First, viscous friction cannot be considered because, as observed in Section 3.3.3, it yields non-linear constraints in the $\dot{\lambda}^2 - \ddot{\lambda}$ plane. Second, constraints on the torque rates (Section 3.2.3.2) cannot be included, as they are not convex.

Despite these limitations, this problem formulation is extremely flexible, as it can easily accommodate additional convex objective functions, together with the time-minimum one, and numerous

convex constraints, which include, but are not limited to, joint-space and task-space velocity and acceleration limits, as those examined in Sections 3.2.3.3 and 3.2.3.4.

Direct transcription can be used to transform problem (3.103) into a large sparse optimization problem, which can be solved using any general-purpose non-linear solver. However, as shown in [88], the objective function and the constraints can be further manipulated to be formulated as a *second-order cone program* (SOCP), which allows to use more efficient dedicated solvers, which are able to return the time-minimum solution in a few seconds of CPU time, even for complex paths.

Although the theoretical formulation and the manipulation of the equations are not straightforward, the time-optimal planning problem can be solved without any knowledge of the maximum velocity curve and of the phase plane properties. Thus, critical and singular points and arcs are automatically supported by the algorithm. Global optimality is guaranteed by the numeric solving procedure.

On the practical side, a few issues may arise by the critical points: results show a little jitter in the motor torques at such points, caused by the numeric optimization procedure. This can be avoided by introducing, in the objective function, a little component penalizing torque jumps, with a negligible impact on the overall execution time.

3.4.5 Resolution by spline interpolation

In Section 3.3.2, we have seen that the phase-plane trajectory is a set of smooth curves separated by critical points, if any. This property paves the way to the possibility of approximating the phase-plane trajectory with a set of splines. In [64], a knot-point, i.e. a point where two splines adjoin, is chosen for each switching point, thereby identified beforehand. Additional knot-points could be chosen to improve the precision of the solution, to the detriment of the time needed for computing the time-optimal tra-

jectory. Usually, cubic splines are sufficient, as they are the lowest-degree polynomial resulting in smooth curves.

With such a problem formulation, the optimization variables are the pseudo-velocities at the knot-points and the slopes at the two end-points of the phase-plane trajectory. All the variables have to be maximized under the constraints that all the limits are respected. In [64], this problem is addressed with the *Flexible Tolerance Method* (FTM), but other numeric techniques, such as evolutionary algorithms, could also be used.

While this method allows to manage torque rate limits very straightforwardly, it does not guarantee the achievement of the minimum time trajectory [77]. On the other hand, because of the intrinsic smoothness of the solution, jitters in the torques in the proximity of critical points are likely to be eliminated.

3.5 Use case definition and resolution by identification of switching points

In this section, we introduce a simple use case that we can later use to present two more techniques that we will analyze in greater detail with respect to those of Section 3.4: genetic algorithms and dynamic programming. For the sake of comparison, we also solve the same use case, as done in [78], by the identification of switching points and management of singular points, so as to always obtain feasible solutions, as explained in Section 3.3.2.3.

Let us consider a 2R planar manipulator with parameters as in Table 3.1, that is requested to track the geometrical path represented in Figure 3.16 in minimum time. The robot's initial configuration is as in Figure 3.16. The use case is essentially the same as [78] and [82], but, since the path parameters are not available therein, the path considered here is generated from the drawing in [82] through a spline interpolation with six control points, i.e., $p_1, ..., p_6$, whose coordinates are reported in Table 3.2, to be as close as possible to

the original path. The path is designed to have 5 switching points in the phase plane, where one is a critical (and singular) point, one is a tangent point and the remaining three are U-to-L switching points.

	Link/Joint 1	**Link/Joint 2**
Link length (m)	1.00	1.00
Link COM position (m)	0.50	0.50
Link mass (kg)	1.00	1.00
Link inertia (kg·m^2)	0.08	0.08
Torque (N·m)	20	10

Table 3.1: Shiller's 2R manipulator parameters

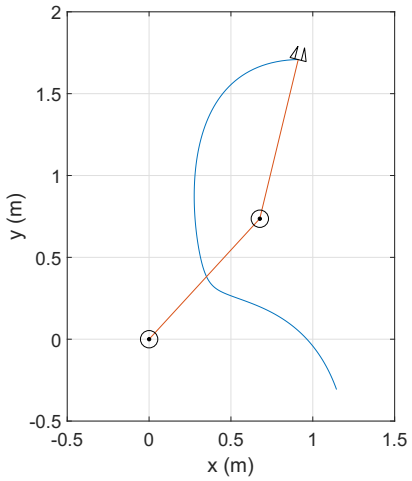


Figure 3.16: Shiller's 2R manipulator (red) and path (blue)

The procedure employed to solve this use case by the identification of switching points is reported in Algorithm 4. Specific checks on the singularity of the critical points are made to generate feasible

	p_1	p_2	p_3	p_4	p_5	p_6
x (m)	0.9112	0.3190	0.3107	0.4697	0.8093	1.1452
y (m)	1.7087	1.2309	0.5155	0.2772	0.1286	-0.3073

Table 3.2: Shiller's path control points

phase plane trajectories according to (3.74) and (3.75), but singular arcs are not taken into account. In this sense, Algorithm 4 is a simplified version of the algorithm in [78], which, on the contrary, considers them.

Algorithm 4 Time-optimal trajectory planning algorithm with identification of switching points and management of singular points

1: *Compute the MVC* $\dot{\lambda}_{max}$
2: *From the initial state* $\mathbf{s}_0 = [0, 0]^T$*, integrate forward with* $\ddot{\lambda} = U$ *until* $\dot{\lambda}_{max}$ *is reached or* $\lambda = \Lambda$ *is reached*
3: **if** $\lambda == \Lambda$ **then** Go to 13
4: **if** $\dot{\lambda} \geq \dot{\lambda}_{max}$ **then**
5: *Search forward on the MVC for the next L-to-U switching point*
6: **if** $\ddot{\lambda} = L$ is singular **then**
7: *Integrate backward with* $\ddot{\lambda} = \ddot{\lambda}_{d,f}$
8: *Integrate backward with* $\ddot{\lambda} = L$ *until the forward profile of the PPT is met*
9: **if** $\ddot{\lambda} = U$ is singular **then**
10: *Integrate forward with* $\ddot{\lambda} = \ddot{\lambda}_{a,f}$
11: *Integrate forward with* $\ddot{\lambda} = U$ *until* $\dot{\lambda}_{max}$ *is reached or* $\lambda = \Lambda$ *is reached*
12: Go to 3
13: *From the final state* $\mathbf{s}_0 = [\Lambda, 0]^T$*, integrate backward with* $\ddot{\lambda} = L$ *until the phase plane trajectory is met*

The time-optimal phase plane trajectory generated by Algorithm 4 on the use case considered here is shown in Figure 3.17, together with the MVC. The trajectory tracking time is $t_{opt} = 0.840$ s, obtained with a resolution of 800 samples for λ. Since the geometrical path is sampled more finely at the higher curvature segments, this number of samples yields a variable integration step between 0.0015 and 0.0066. The results are not immediately comparable with those from [82] since the geometrical path in Figure 3.16 is only an approximate reconstruction of the one in [82]. However,

from the qualitative point of view, the characteristics of the time-optimal phase-plane trajectory are comparable.

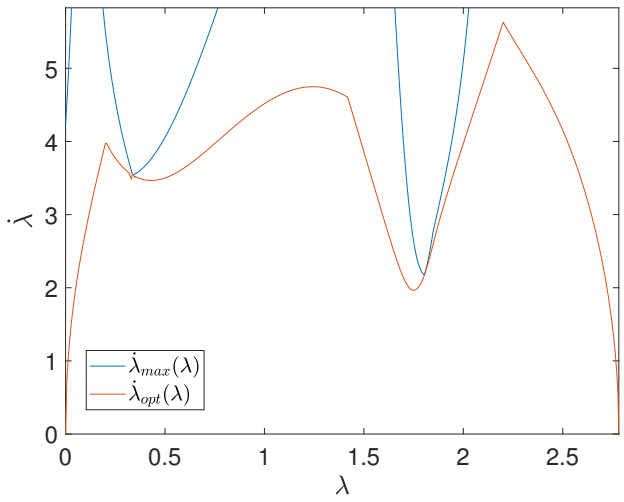


Figure 3.17: Maximum velocity curve and time-optimal phase plane trajectory for the use case of Figure 3.16

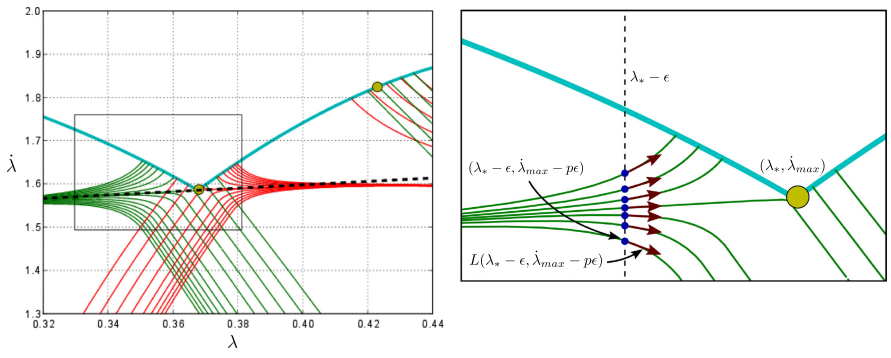
3.5.1 Handling of critical points

The phase-plane trajectory of Figure 3.17 is not everywhere smooth. In particular, some jitters are visible at the singular point. This is a well-known issue when critical points exist.

In Section 3.3.2.3, critical points have been introduced. They are points where the maximum and minimum pseudo-accelerations are not equal, because the bounding inertia term $a_i(\lambda)$ vanishes. In the $\dot{\lambda}^2 - \ddot{\lambda}$ plane, this means that a vertical line bounds the maximum pseudo-velocity. In the $\lambda - \dot{\lambda}$ plane, the incoming pseudo-acceleration in the trajectory sink and the outgoing pseudo-acceleration from the trajectory source are different, marking a discontinuity in the derivative of the phase plane trajectory. In general, it does not represent an issue, as discontinuities of this kind are

likely to be generated for all the switching point from maximum pseudo-acceleration to minimum pseudo-acceleration, however, in the specific case of critical points, such discontinuities could be troublesome.

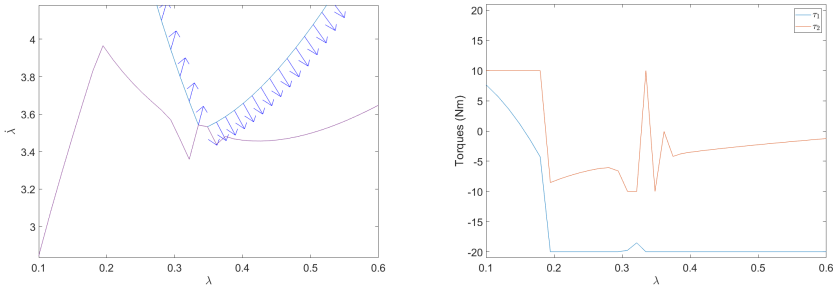
Maximum and minimum pseudo-accelerations form fields in the phase plane and the integration of the phase plane trajectory with one of the methods discussed in Section 3.4.1 follows one of the profiles in the fields for any acceleration or deceleration segment. In the proximity of critical points though, such fields are divergent (see Figure 3.18), meaning that the integration of the phase plane trajectory is unlikely to reach the critical point. In truth, for some of the methods discussed in Section 3.4.1, the integration starts from the critical point, which implies that the phase plane trajectory necessarily passes through it. However, it has been demonstrated that the integration starting from the critical point and following either L (for the backward integration) and U (for the forward integration) or (3.74) and (3.75), in the case of singular points, may cause jitters in the torque profiles [68]. Figure 3.19 shows a detail of the phase plane trajectory in Figure 3.17 about the singular point and highlights this phenomenon.



(a) (b)

Figure 3.18: Fields of L (green) and U (red) in the proximity of a critical point (a); close-up view, zoomed in the black box (b) [68]

Let us consider the case of backward integration. Starting from



(a) (b)

Figure 3.19: Jitter in the phase plane (a) and in the torques (b) caused by divergence of the pseudo-acceleration fields for the same critical point as Figure 3.8

the idea that the phase plane trajectory has to pass by the critical point, for any small $\epsilon > 0$, the gradient g of the phase plane trajectory at $\lambda_* - \epsilon$, with reference to Figure 3.18 (b), must point toward the critical point. It is clear that the only point satisfying this condition in the left neighborhood of the critical point is the one lying on the dashed line of Figure 3.18(a). Thus the gradient can be approximated with

$$g = \frac{\dot{\lambda}_{max} - \dot{\lambda}_{max} + g\epsilon}{\lambda_* - \lambda_* + \epsilon} \tag{3.104}$$

Recalling the chain rule, we also have

$$g = \frac{d\dot{\lambda}}{d\lambda} = \frac{L_i(\lambda_* - \epsilon, \dot{\lambda}_{max} - g\epsilon)}{\dot{\lambda}_{max} - g\epsilon} \tag{3.105}$$

where L_i is the minimum pseudo-acceleration enforced by the i-th joint. In [68], it was demonstrated that, if a critical point is caused by the i-th joint, i.e. $a_i(\lambda) = 0$, then $L = L_i$ to the left of the critical point and $U = U_i$ to the right.

Through some algebraic manipulation and employing a first-order Taylor expansion in ϵ, said $\dot{\lambda}_{max} = \dot{\lambda}_{max}(\lambda_*)$, we can approximate

the gradient as [68]

$$g = -\frac{b_i'(\lambda_*)\dot{\lambda}_{max}^2 + c_i'(\lambda_*)}{\left[2b_i(\lambda_*) + a_i'(\lambda_*)\right]\dot{\lambda}_{max}} \tag{3.106}$$

where

$$c_i(\lambda) = \tau_{i,min}\delta_i(\lambda) + \tau_{i,max}\big(1 - \delta_i(\lambda)\big) - g_i(\lambda) \tag{3.107}$$

Likewise, in order to find the value of the gradient for the forward integration, one should solve

$$g = \frac{-U_i(\lambda_* + \epsilon, \dot{\lambda}_{max} + g\epsilon)}{\dot{\lambda}_{max} + g\epsilon} \tag{3.108}$$

which can be demonstrated to yield equation (3.106) as well.

By integrating from the critical point both backward and forward with gradient (3.106) for a few λ samples, the jitters can be avoided, resulting in much smoother profiles [68].

In the three-dimensional phase space, the same issue arises on the critical curves. The same strategy presented here can be adopted without any major change [77]. It yields a pseudo-jerk different from j_{min} and j_{max} that can be used at the switching point to connect the backward and forward profiles, eliminating jitters on the torques.

3.6 Resolution with a genetic algorithm

The idea behind an Evolutionary Algorithm (EA) is to evolve a population of solutions (also called *genomes* or *individuals*) by means of the mechanisms of selection, recombination and mutation. The evolutionary search starts with a randomly generated population of solutions where each individual has a one-to-one connection with a time-optimal phase plane trajectory, meaning that a genome can directly represent a time-optimal trajectory or

can be used to compute one and only one time-optimal trajectory. Since a torque has to be defined for each joint, it is certainly convenient to work in the λ domain, so that only one variable, i.e., the pseudo-velocity, must be optimized.

Each individual in the population is evaluated through a *fitness* function, which is a measure of its quality with respect to a given performance index. In the case of time-optimal planning, the fitness function must be related to the inverse of time.

Once an initial population is available, the genomes therein can be combined according to the *crossover* operator, in order to generate individuals with a higher fitness. Since the features of the optimal individual might not be present in the initial population, and, as a consequence, cannot be generated by the means of crossover, some sort of *mutation* should also be considered, which is a random variation of the genome itself.

In [89], a class of EAs, called Evolution Strategy (ES), is employed for the same problem of time-optimal planning along prescribed paths that we address here. Therein, the resolution is only performed at kinematic level, while in this section the dynamic properties of the manipulator are considered and the problem is solved by exploiting the results of the phase plane analysis that we introduced in Section 3.3.2. The resolution technique is also different, as we present a formulation based on a Genetic Algorithm (GA), that is a different class of EAs.

3.6.1 Constraints

In a genetic algorithm, the accommodation of constraints is not an easy task. In the literature, it is usual to refer to *direct* constraint handling when the constraints are explicitly enforced on the individuals of the population, and to *indirect* constraint handling when they are included in the objective function. In [90], four techniques are reported for direct constraint handling:

- *elimination*: crossover and mutation are free to generate unfeasible candidates that are later spotted and eliminated by the selection mechanism;

- *repairment*: crossover and mutation are free to generate unfeasible candidates that are later spotted and corrected so as to become feasible, by means of a specific operator;

- *preservation*: crossover and mutation are designed for the specific applications, so that, once an initial population of feasible individuals is given, only feasible individuals can be generated by the operators;

- *decoding*: the search space is transformed so that constraints are automatically satisfied in the domain of the transformed variables.

Recalling the problem formulation in (3.54) and the considerations of Section 3.3.2.1, time-optimal planning, represented in the λ domain, requires the following constraints to be satisfied:

$$0 < \dot{\lambda}(\lambda) < \dot{\lambda}_{max}(\lambda) \qquad \forall \lambda \in \ ]0, \Lambda[\tag{3.109}$$

$$L(\lambda, \dot{\lambda}) \leq \ddot{\lambda}(\lambda, \dot{\lambda}) \leq U(\lambda, \dot{\lambda}) \quad \forall \lambda \in [0, \Lambda] \tag{3.110}$$

$$\dot{\lambda}(0) = 0 \tag{3.111}$$

$$\dot{\lambda}(\Lambda) = 0 \tag{3.112}$$

If viscous friction effects are included in the dynamic model, the constraint (3.109) more generically is

$$\dot{\lambda}(\lambda) \in \mathcal{F} \tag{3.113}$$

where $\mathcal{F}$ represents the feasible region in the phase plane. For each value of λ, because of the presence of islands, $\mathcal{F}(\lambda)$ is not, in general, a connected set.

3.6.2 Definition of genomes and fitness function

The first issue to address in our genetic algorithm is to determine which variables the genomes in the population represent, that is to choose the variables to optimize. As done in [91], let us first assume to discretize the λ domain, i.e.,

$$\lambda(i) = i\Delta \quad \text{with } i = 0, 1, 2, .., N_i - 1 \quad \text{and } N_i = \frac{\Lambda}{\Delta} + 1 \quad (3.114)$$

so that λ_i and i can be used interchangeably. Then, let us associate each discrete value of λ with one and only one *chromosome*, i.e., a variable in the genome.

Then, at least two options are available for the genome semantics:

- each individual in the population represents the vector of pseudo-velocities for each λ, i.e., $\dot{\lambda}(\lambda)$;
- each individual in the population represents the vector of pseudo-accelerations for each λ, i.e., $\ddot{\lambda}(\lambda)$.

As far as the fitness function f is concerned, as discussed in Section 3.4.2, the trajectory time (the cost function to minimize) is inversely proportional to the area underneath the phase-plane trajectory A, that, thereby, needs to be maximized. Hence, it is a natural choice for the fitness function:

$$f = A = \int_0^{\Lambda} \dot{\lambda}(\lambda) d\lambda \quad (3.115)$$

3.6.2.1 Genomes representing pseudo-velocities

If the semantics of the chromosome relates to the pseudo-velocity, one should consider that:

- since a genome is a phase-plane trajectory, the area underneath can be computed directly from it;

- *preservation* can be used to lock the first and last chromosomes to zero in order to satisfy constraints (3.111) and (3.112), meaning that crossover and mutation shall not operate on them;

- *decoding* can be used to adopt a different MVC-dependent domain for each chromosome, so that constraint (3.109) is automatically satisfied;

- constraint (3.110) could be managed indirectly, but the penalty to be considered in the fitness function would be dependent on as many local contributions as the number of samples in the discrete λ-set, posing a problem of sensitivity, especially when the number of samples changes; alternatively, one could think of setting up a repairment mechanism by which, when the pseudo-acceleration is beyond the maximum or minimum threshold, it is saturated to either L or U; however local repairment does not guarantee feasibility because, if the trajectory ends up in a trap region (see Section 3.3.2.1), it cannot exit without violating either the pseudo-acceleration or MVC constraints.

3.6.2.2 Genomes representing pseudo-accelerations

If the semantics of the chromosome relates to the pseudo-acceleration, we have that:

- an integration in the phase plane is necessary to compute $\dot{\lambda}(\lambda)$, from which the fitness function can be computed;

- $\dot{\lambda}(0) = 0$ is the initial condition for the integration, meaning that constraint (3.111) is automatically satisfied; this could be seen as a form of *preservation*, although it only affects the integration and not the crossover and mutation operators;

- ideally, any value between L and U can be used, so that singular points are automatically taken into account;

- constraint (3.109) can be managed indirectly by including it in the fitness function; a measure of the constraint violation is given by the area A_p (*penalty area*) underneath the phase-plane trajectory when it is above the MVC, i.e.,

$$A_p = \int_0^{\Lambda} \dot{\lambda}(\lambda)\big(\dot{\lambda}(\lambda) > \dot{\lambda}_{max}(\lambda)\big)d\lambda \tag{3.116}$$

  which allows to redefine the fitness function f as

$$f = \frac{A}{1 + hA_p} \tag{3.117}$$

  where h is a tunable constant; if inadmissible islands are present in the phase plane, A_p should also include the portion of area inside them;

- *decoding* can be used to satisfy constraint (3.110): the range of values for the pseudo-acceleration is variable with λ and $\dot{\lambda}$, meaning that the domain of the chromosomes is not constant; for this reason, it might be convenient to adopt a domain transformation by which each chromosome represents the gain $k \in [0, 1]$ between the minimum and maximum pseudo-acceleration, i.e.,

$$\ddot{\lambda} = \min\{L, U\} + k\,(\max\{L, U\} - \min\{L, U\}) \tag{3.118}$$

  where the usage of functions min and max is necessary to account for the case when, above the MVC, $L > U$; the reader may verify that constraint (3.110) is automatically satisfied with this domain transformation, while the chromosome's domain is constant across λ;

- *repairment* can be used to enforce constraint (3.112); it is interesting to notice that, because the area maximization is independent of the last deceleration segment bringing the phase-plane trajectory back to zero, it is sufficient to repair just the best solution at the end of the optimization process, without correcting each single individual in the population; in other words, after the last switching point, the

phase-plane trajectory is free to proceed with the maximum acceleration up to $\lambda = \Lambda$, to be then repaired afterwards with a backward integration starting from $\dot{\lambda}(\Lambda) = 0$.

Because of the difficulty in managing constraint (3.110) with chromosomes corresponding to pseudo-velocities, it is convenient to associate genomes to pseudo-accelerations. However, on the other hand, the CPU execution time considerably increases, since a numerical integration must be performed for each single individual in the population. In response, we need to mitigate by limiting its cardinality as much as possible.

3.6.3 Implementation with TurboGA

The implementation of the genetic algorithm performing time-optimal planning is based on *TurboGA* [92], a versatile tool designed to cope with generic genetic problems. It is characterized by the usual sequence of steps, common to several genetic algorithms, reported in Algorithm 5.

Algorithm 5 Overview of a genetic algorithm

1: *Randomly generate the initial population*
2: **for** each generation **do**
3: *Compute fitness of all individuals*
4: *Perform σ-scaling (if configured)*
5: *Select parents*
6: *Perform crossover*
7: *Perform mutation*
8: *Pick best individual*

TurboGA works with binary chromosomes, meaning that the genome is a string of bits. With this representation, the crossover and mutation operations are based on binary operators and are, therefore, very efficient. Each bit in a chromosome, which could be individually modified by the mutation operator, is referred to as *locus*. The number of loci in the chromosome determines the precision of the relating variable. In our implementation, it corresponds to the precision of the pseudo-acceleration gain, and indirectly affects the

precision of the pseudo-acceleration itself. In absence of singular points, since the solution is bang-bang, one locus per chromosome would be sufficient to obtain the time-optimal solution, i.e.,

$$\begin{aligned} k = 0 \;&\Rightarrow\; \ddot{\lambda} = L \\ k = 1 \;&\Rightarrow\; \ddot{\lambda} = U \end{aligned} \tag{3.119}$$

Rather, if singular points exist, the pseudo-acceleration must be chosen in $]L, U[$, meaning that one locus is no longer sufficient to obtain a feasible solution. Since we do not pre-process the MVC to identify and classify the switching points, in our implementation, we assume a constant number of loci for each chromosome.

Once the fitness (3.117) has been computed for all the individuals in the population, as a preparatory stage for selection, we may decide to apply σ-scaling [93], which allows to rescale the fitness to increase or decrease the probability for an individual to be selected as a parent for the new generation. An higher σ-scaling factor flattens the fitness values in the population and causes a slow convergence. On the other hand, it increases the probability that important features do not get lost after crossover, since it gives a chance to low-fitness individuals with high-fitness features to pass them to the next generations. From the optimization standpoint, σ-scaling can be tuned to increase the chances of avoiding local optima.

The parent selection process is based on the σ-scaled fitness values. As many parents as the number of individuals in the population are selected, but the same parent can be selected more than once, and individuals in the population with lower fitness are excluded. Two common selection mechanisms are the *Roulette Wheel* (RW) and the *Stochastic Universal Sampling* (SUS), where the latter guarantees that high-fitness individuals are always selected, while the former does not, especially for small populations [93], as the one we are considering here. It is clear that SUS is the natural choice for time-optimal planning.

Crossover and mutation are implemented through bitmasks, that are randomly generated beforehand and saved in a repository to

use them in the main cycle. Since we are not implementing any form of *preservation*, except for the integration in the phase-plane, these operators do not need to be modified.

One more aspect that deserves a mention is *clamping*, which is the capability of protecting some loci from mutation. In fact, in absence of crossover, the capability of an individual to be present in the next generations decreases with the number of generations. If the probability of mutation is, say, 0.003, the probability for an individual to survive after j generations is $(0.997)^{jl}$, where l is the total number of loci in a genome. The reader may recognize that this probability is very low after a few generations, meaning that, even though an individual is promising, because of mutation it is not likely to survive. This phenomenon is referred to as *mutational drag*. The clamping is a mechanism to "clamp" the loci that in the population are very frequent, which is an indication that crossover preserved them because of their high-fitness, saving from the mutational drag.

Once the best individual is picked from the last generation, it is repaired to respect constraint (3.112) in order to provide a solution where the robot stops at the final endpoint, which is a requirement in our problem. The phase-plane trajectory is integrated backwards from the state $\lambda = \Lambda$ and $\dot{\lambda} = 0$, until it meets the forward profile. This step is common to other time-optimal planning techniques as it is independent of the MVC after the last L-to-U switching point. The assumption here is that, if a phase-plane trajectory is optimal for $\dot{\lambda}(\Lambda) \neq 0$, it is also optimal after the addition of a decelerating segment bringing it to zero.

On the basis of the observations above, the algorithm performing time-optimal planning with genetic search is provided in Algorithm 6. It is assumed that the dynamic parameters of the manipulator are known.

Algorithm 6 Time-optimal trajectory planning with a genetic algorithm

Initialize trajectory $\mathbf{x}(\lambda)$, *or* $\mathbf{q}(\lambda)$
Initialize torque limits τ_{min}, τ_{max}
Initialize the number of chromosomes N_i
Compute $\mathbf{a}$, $\mathbf{b}$, $\mathbf{g}$, $\dot{\lambda}_{max}$ *by using the parameters above*
Generate crossover and mutation bitmasks repos
Randomly generate the initial population
for each generation **do**
    for each genome **do**
        Integrate forwards starting from (3.111) *and transform chromosomes to pseudo-accelerations according to* (3.118) *so as to satisfy* (3.110)
        Compute fitness as in (3.117) *so as to tend to* (3.109)
    Perform σ*-scaling*
    Select parents with SUS
    Perform locus-wise crossover
    Perform locus-wise mutation
$\dot{\lambda}(\lambda) \leftarrow$ best individual in the population
Repair $\dot{\lambda}(\lambda)$ *so as to satisfy* (3.112)
Compute trajectory execution time t *from* $\dot{\lambda}(\lambda)$

3.6.4 Results on a 2R planar manipulator

Let us consider again the use case of Section 3.5. The list of parameters used in TurboGA simulations is reported in Table 3.3. A few comments are worthwhile:

- the size of the population, because of the need to perform an integration for each genome, must be kept small;
- the number of generations has been tuned experimentally, but in the large majority of cases, convergence happens around generation 2000.
- the σ-scaling coefficient is relatively small compared to typical values and this is to guarantee a faster convergence; experimental results show that the time-optimal planning GA is unlikely to end up in local maxima that are "too far" from the global one;
- the clamping parameters have been tuned experimentally to compensate the mutation rate, as discussed in Section 3.6.3.

Name	Value	Notes
`nchrom`	150	number of chromosomes (N_i)
`nloci`	32	number of loci in a chromosome
`lower`	0	lower bound of chromosome domain
`upper`	1	upper bound of chromosome domain
`popSize`	100	number of genomes in the population
`maxGens`	3000	number of generations
`probCrossover`	1.000	probability of crossover
`probMutation`	0.003	probability of locus-wise mutation
`sigmaScalingCoeff`	0.04	σ-scaling coefficient (multiplier of variance)
`crossoverType`	2	uniform crossover
`flagFreq`	0.1	percentage of occurrence of a 0 or 1 for given locus across the population above which the locus is flagged for clamping
`unflagFreq`	0.1	percentage of occurrence of a 0 or 1 for a given locus across the population below which the locus is unflagged for clamping
`flagPeriod`	150	number of generations after which a flagged locus is clamped
`h`	0.15	see equation (3.117)

Table 3.3: Parameters used in the TurboGA runs

By using the switching points technique described in Algorithm 4, the optimal execution time, with a λ-discretization of 150 samples, is

$$t_{opt} = 0.870 \text{ s} \tag{3.120}$$

corresponding to an optimal fitness function of

$$f_{opt} = A_{opt} = 10.557 \tag{3.121}$$

Algorithm 6 is executed with the parameters of Table 3.3. The average fitness and standard deviation over 20 runs are

$$\begin{aligned} \mu_f &= 10.557 \\ \sigma_f &= 0.005 \end{aligned} \tag{3.122}$$

corresponding to an average time and standard deviation of

$$\begin{aligned} \mu_t &= 0.869 \text{ s} \\ \sigma_t &= 0.5 \text{ ms} \end{aligned} \tag{3.123}$$

The genetic algorithm always obtains a solution in the neighborhood of the optimal one and, sometimes, provides a shorter time than (3.120). This is another consequence of the discretization of λ. In fact, assume an MVC that has no singular points and that the switching points method is used to compute the time-optimal phase-plane trajectory. Theoretically, the solution must be bang-bang. However, if λ is discretized, the solution is everywhere bang-bang, except at the U-to-L switching points, where the leftmost point of the backward integration trajectory must be connected to the rightmost point of the forward integration trajectory. If the number of loci in a chromosome is greater than one, other pseudo-accelerations than L and U can be chosen. Hence, the solution adapts better at the U-to-L switching points and can produce a slightly faster solution. The coarser the discretization is, the higher is the margin for the genetic algorithm to perform better than the switching points technique. A typical solution of Algorithm 6, indicated with $\dot{\lambda}_{ga}(\lambda)$, which also encompasses this aspect, is shown in Figure 3.20. The phase-plane trajectory $\dot{\lambda}_{opt}(\lambda)$, computed with the switching points technique, is also shown for comparison. The solution has 5 switching points, coherently to [82]. The reader may recognize that almost everywhere $\dot{\lambda}_{ga} \simeq \dot{\lambda}_{opt}$. In the range between the third and fourth switching point, where $\lambda \sim 1.5$, the 32-bits sensitivity provided by the loci in the chromosomes allows to "adapt" better to the discretization and gain a portion of area above the $\dot{\lambda}_{opt}$. The best individual after 3000 generations, from one of the runs, is shown in Figure 3.21. The reader may recognize the quasi-bang-bang shape of the solution with three accelerating segments and two decelerating segment. The last maximum deceleration segment, bringing to satisfaction of constraint (3.112), as discussed before, is added through repairment so that to generate the $\dot{\lambda}_{ga}$ curve of Figure 3.20. Also, the

pseudo-acceleration gains in Figure 3.21 clearly show the higher sensitivity in the vicinity of the switching points.

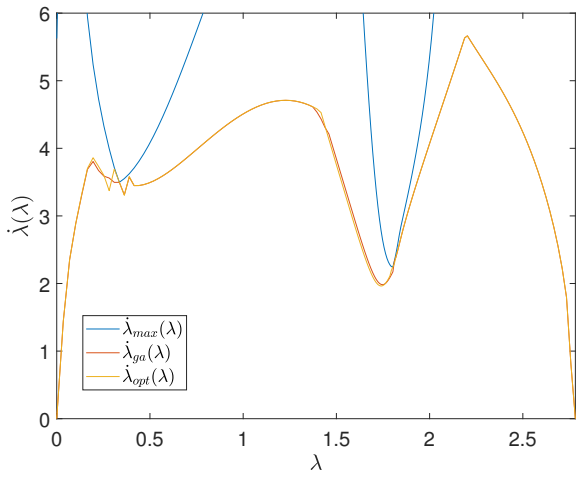


Figure 3.20: MVC (blue) and phase-plane trajectories computed with switching points (yellow) and genetic algorithm (red) for manipulator and path in Figure 3.16

As far as the singular point is concerned (corresponding to the second switching point), the time-optimal phase-plane trajectory presents some discontinuities in its derivatives. This is an expected behavior, that causes jitters in the torque profile, as theorized in [68]. With the genetic algorithm solution, a smoother profile is obtained to the left of the singular point. Once again, this is due to the higher sensitivity in the pseudo-accelerations. Although this is not common to all runs, it is quite a frequent behavior in genetic solutions. Also, at the singular point, the genetic algorithm "automatically" chooses pseudo-accelerations that do not violate the MVC constraint.

The execution time of Algorithm 6, implemented in MATLAB®, with the parameters in Table 3.3, is $\sim$ 2.4 minutes, with a 64-bit Windows 10 OS running on an Intel® Xeon(R) E-2146G CPU @ 3.50GHz. The time required to compute the parametrized dynam-

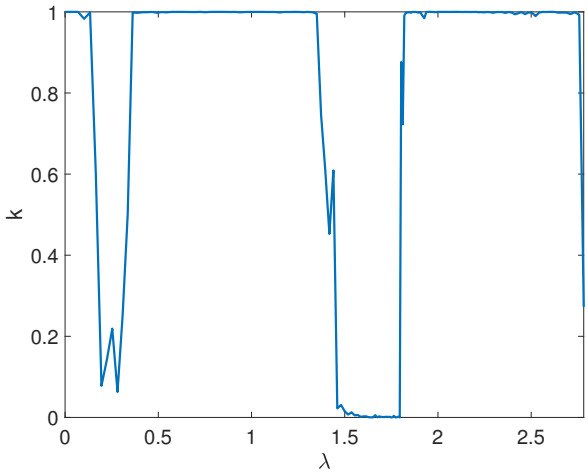


Figure 3.21: Best individual after 3000 generations from one of the runs

ics and maximum velocity curve in step 4 is about 10 seconds.

3.6.5 Parameter sensitivity analysis

All the parameters reported in Table 3.3 are standard parameters for a GA, except for the parameter h that we defined in (3.117). It is clear that, if h is too small, unfeasible solutions will be penalized less, and are likely to survive to the selection process. On the other hand, if h is too large, solutions close to the optimal one will be penalized too much and are likely to be excluded by the selection process. A sensitivity analysis for the parameter h, showing this phenomenon, is reported in Figure 3.22. The graph to the left represents the fitness function mean over 20 runs and related variance for $h \in [0.10, 1000]$. When $h = 0.10$, the fitness function is much higher than for other values, but all the 20 solutions are unfeasible: the phase-plane trajectory is above the MVC. As h tends to infinity, the fitness function decreases, moving away from the optimal value. Rather, the graph to the right represents

the fitness function mean over 20 runs and related variance for $h \in [0.10, 0.20]$, i.e., in the vicinity of 0.15, which is the value we selected for our GA. While for $h > 0.14$, the fitness function is at the optimal value indicated in (3.122); for $h < 0.14$, unfeasible solutions start to be accepted. For instance, for $h = 0.13$, only 11 solutions are feasible out of 20.

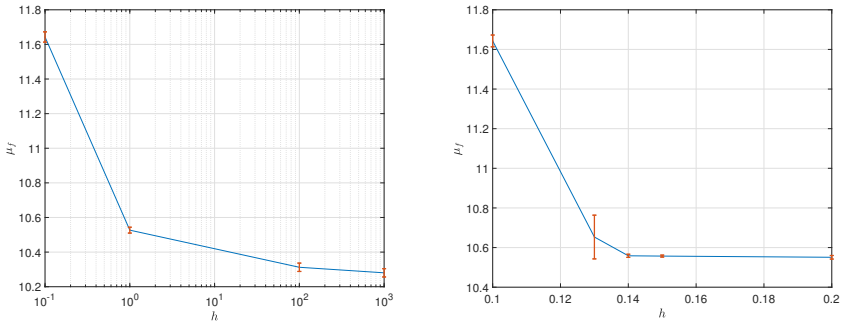


Figure 3.22: Mean of fitness function and related variance (represented with vertical bars), on 20 runs, with respect to parameter h on a logarithmic scale (left) and in the vicinity of the selected value (right)

3.6.6 Scalability analysis

Together with the parameter sensitivity above, it is interesting to estimate the performance of the GA with respect to the variation of N_i, the number of chromosomes, or number of waypoints in the path. We set six different values for N_i, and execute 20 runs for each of them. In order to estimate the CPU execution time, runs are stopped as soon as they reach the optimal area, that is precomputed from the switching points method. Results are shown in Table 3.4 and Figure 3.23, showing that the CPU execution time is approximately linear with the number of waypoints/chromosomes.

Table 3.4: Optimal area, average μ_{cpu} and standard deviation σ_{cpu} of the CPU execution time over 20 runs for different numbers of waypoints/chromosomes

$\mathbf{N_i}$	**50**	**150**	**250**	**350**	**450**	**550**
$\mathbf{A_{opt}}$	10.477	10.557	10.567	10.590	10.609	10.618
$\boldsymbol{\mu}_{\mathbf{cpu}}$ **(s)**	26.3	143.3	220.8	345.9	375.4	457.1
$\boldsymbol{\sigma}_{\mathbf{cpu}}$ **(s)**	6.1	5.0	6.7	6.4	6.0	7.1

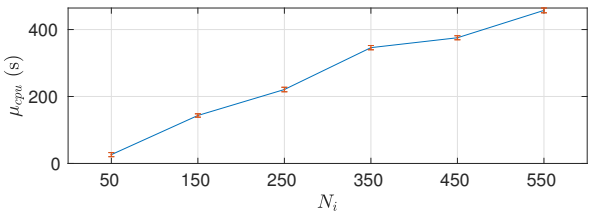


Figure 3.23: Average μ_{cpu} and standard deviation σ_{cpu} of the CPU execution time over 20 runs for different numbers of waypoints/chromosomes

3.6.7 Parameters robustness to path characteristics

One of the most important drawbacks of genetic algorithms is their high number of parameters, most of which have to be tuned experimentally. In general, changes in the problem formulation, such as robot characteristics (e.g., kinematic structure, dynamic parameters) and path characteristics (e.g., regularity, maximal curvature, distance from workspace boundaries) might require the re-tuning of the GA parameters. In order to estimate the robustness of our GA-based time-optimal planning algorithm with respect to path characteristics, it is worth testing a second path that has different characteristics from the one of Figure 3.16. The latter is a quite irregular path, with varying curvature (but never straight) that

starts very close to the workspace outer boundary and presents two critical points, one of which is singular. Hence, a good choice for our second use case could be the path of Figure 3.24: it is a straight line between points $(0.3, 1)$ and $(0.3, -1)$, that passes very close to the inner workspace boundary, i.e., the manipulator's base, and only presents one singular point that, in the optimal solution, lies very close to the preceding U-to-L switching point.

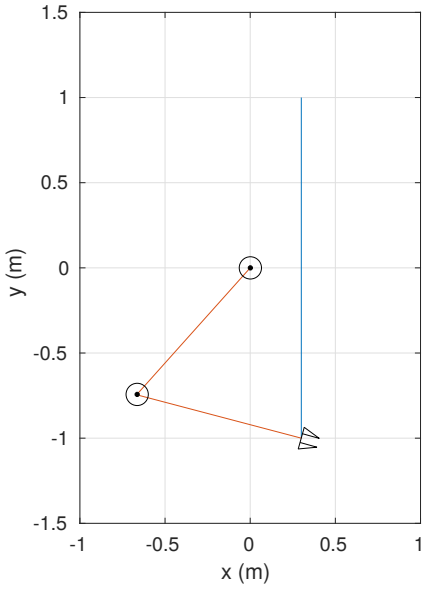


Figure 3.24: Shiller's 2R manipulator at its starting configuration (red) and straight path (blue)

The problem is solved again with the switching points shooting method, providing the ground truth for this second use case, having the following optimal execution time:

$$t_{opt} = 0.781 \tag{3.124}$$

corresponding to an optimal fitness function of

$$f_{opt} = A_{opt} = 5.930 \tag{3.125}$$

Since, in Section 3.6.4, we did not make any path-specific consideration in the tuning of the parameters, Algorithm 6 is executed again with the same values as before, reported in Table 3.3. The average fitness and standard deviation over 20 runs are

$$\begin{aligned} \mu_f &= 5.963 \\ \sigma_f &= 0.002 \end{aligned} \tag{3.126}$$

corresponding to an average time and standard deviation of

$$\begin{aligned} \mu_t &= 0.777 \text{ s} \\ \sigma_t &= 0.1 \text{ ms} \end{aligned} \tag{3.127}$$

Although we assigned a fixed number of generations for all the runs, for this use case, the GA converges to the globally-optimal solution at around generation 500, corresponding to a CPU execution time of $\sim 46\,s$. A comparison between a typical solution of Algorithm 6 and that of the switching points shooting method is provided in Figure 3.25.

In this particular case, the GA-based algorithm is able to completely eliminate the torque jitter at the singular point. Since it cannot be observed from the phase plane trajectories in Figure 3.25 directly, torques profile are reported in Figure 3.26.

3.7 Resolution with dynamic programming

3.7.1 Problem formulation

The first demonstration of solving the trajectory planning problem over a specified path with dynamic programming (DP) was given in [91]. Therein, the problem formulation is generic enough to accommodate a family of integral performance indices, including the trajectory tracking time. The authors propose to discretize

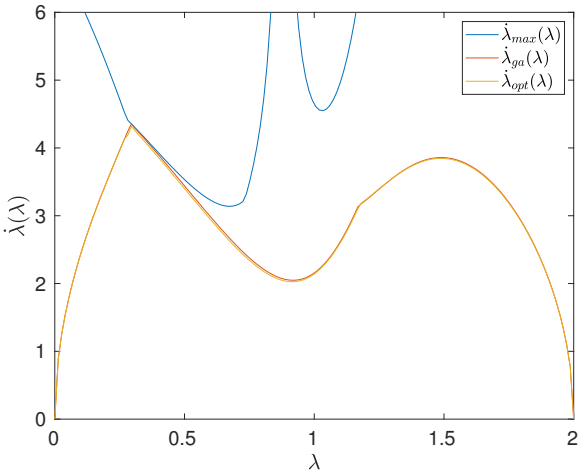


Figure 3.25: MVC (blue) and phase-plane trajectories computed with switching points (yellow) and genetic algorithm (red) for manipulator and path in Figure 3.24

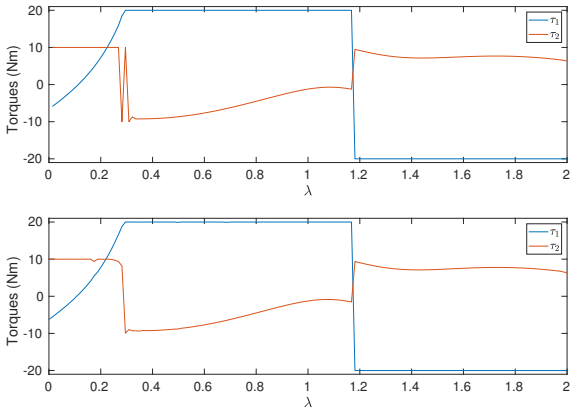


Figure 3.26: Comparison of torque profiles between the switching points shooting method (top) and Algorithm 6 (bottom)

the phase plane so as to create a grid in λ and $\dot{\lambda}$, i.e.

$$\begin{aligned} \lambda(i) &= i\Delta_\lambda \quad \text{with } i = 0, 1, 2, .., N_i \quad \text{and } N_i = \frac{\Lambda}{\Delta_\lambda} \\ \dot{\lambda}_j &= j\Delta_{\dot{\lambda}} \quad \text{with } j = 0, 1, 2, .., N_j \quad \text{and } N_j = \frac{\dot{\lambda}_M}{\Delta_{\dot{\lambda}}} \end{aligned} \tag{3.128}$$

where Δ_λ and $\Delta_{\dot\lambda}$ are the sampling intervals for the λ-domain and $\dot\lambda$-domain respectively, while $\dot\lambda_M$ is a constant representing the theoretical upper limit for any feasible phase plane trajectory, so that, said $\dot\lambda_{opt}(\lambda)$ the minimum-time trajectory, $\dot\lambda_{opt}(\lambda) < \dot\lambda_M \ \forall\lambda$, e.g.

$$\dot\lambda_M = \max_\lambda\{\dot\lambda_{max}(\lambda)\} \tag{3.129}$$

It is anyhow worth remarking that, for MVCs with high peaks, where trap regions are particularly large, (3.129) may cause a loss of resolution in the feasibility region, which is where the time-optimal phase-plane trajectory must be searched. Hence, $\dot\lambda_M$ might be set to a conservative value, that is later increased only if the time-optimal trajectory hits the upper bound of the grid. This may require more than one execution of the algorithm, but repeated searches with increasing resolution are one of the techniques that are often used in dynamic programming to cope with the computation time (see, e.g. [71, 72]).

Each cell in such a grid is a state $\mathbf{s}_j(i) = [\lambda(i), \dot\lambda_j(i)]^T$ that can be evaluated by the dynamic programming algorithm. For each of them, the maximum and minimum pseudo-accelerations $L_j(i) = L(\lambda(i), \dot\lambda_j(i))$ and $U_j(i) = U(\lambda(i), \dot\lambda_j(i))$ can be pre-computed, as well as the parametrized dynamics, represented by the vectors $\mathbf{a}(i) = \mathbf{a}(\lambda(i))$, $\mathbf{b}(i) = \mathbf{b}(\lambda(i))$ and $\mathbf{g}(i) = \mathbf{g}(\lambda(i))$, that are shared by all the states $\mathbf{s}_j(i)$ for fixed i and variable j.

The dynamic programming algorithm can be implemented in the traditional way, i.e. *backward*, from the final waypoint to the initial one (as in [91]), or, considering that all the states possibly visited by the DP algorithm are already available, from $i = 1$ to $i = N_i$, i.e. *forward*. Thus, according to the latter, for a given phase-plane trajectory, a generic objective function can be written as:

$$I(N_i) = \psi\big(\mathbf{s}_0\big) + \sum_{k=1}^{N_i} \phi\big(\mathbf{s}(k-1), \mathbf{s}(k)\big) \tag{3.130}$$

where $\psi\big(\mathbf{s}_0\big)$ is a cost associated to the initial state and ϕ is the cost computed locally between two adjacent states. More specifically,

since the objective function to minimize is the time, there is no cost associated to the initial state, i.e. $\psi\big(\mathbf{s}_0\big) = 0$ and $I = t$. Therefore, using, for the discrete time, the same approximation as [71], we have:

$$t(N_i) = \sum_{k=1}^{N_i} 2\frac{\lambda(k) - \lambda(k-1)}{\dot{\lambda}(k) + \dot{\lambda}(k-1)} \tag{3.131}$$

or, in a recursive form,

$$\begin{aligned} t(0) &= 0 \\ t(i) &= t(i-1) + 2\frac{\lambda(i) - \lambda(i-1)}{\dot{\lambda}(i) + \dot{\lambda}(i-1)} \end{aligned} \tag{3.132}$$

It has to be noted that dynamic programming, with respect to other techniques, is flexible with respect to the inclusion of other performance indices within the same objective function to optimize. For instance, in [14], the time-optimality is coupled with energy-optimality. Weights can be used to privilege one or the other index, as in [14], or Pareto optimality can be achieved.

In order to be complete, the problem definition must include the usual constraints on the pseudo-velocity and maximum and minimum pseudo-accelerations. In particular, we may define the stage-dependent set of admissible pseudo-velocities as $\mathcal{A}_i$ and the stage-dependent set of admissible pseudo-accelerations as $\mathcal{B}_i$ such that the constraints are generally formalized as

$$\begin{aligned} \dot{\lambda}(i) &\in \mathcal{A}_i \\ \ddot{\lambda}(i) &\in \mathcal{B}_i\big(\dot{\lambda}(i)\big) \end{aligned} \tag{3.133}$$

It is worth noticing that such a formulation is generic enough to accommodate the majority of the constraints presented in Section 3.2.3. In case constraints exist on the pseudo-jerk, such as (3.25), an additional set can be defined to include them in the problem definition. Also, in case viscous friction is modeled (see Section 3.3.3), inadmissible islands are automatically taken into account in $\mathcal{A}_i$. The dependence of $\mathcal{B}_i$ on $\dot{\lambda}$ is necessary because the pseudo-acceleration depends on both λ and $\dot{\lambda}$. The sets in (3.133) can be

combined so as to define the set of reachable pseudo-velocities for each stage i:

$$\mathcal{C}_i = \mathcal{A}_i \cap \left\{ \dot{\lambda}(i) : \frac{\dot{\lambda}(i) - \dot{\lambda}(i-1)}{\Delta_\lambda} \dot{\lambda}(i-1) \in \mathcal{B}_{i-1}\big(\dot{\lambda}(i-1)\big), \right. \\ \left. \text{with } \dot{\lambda}(i-1) \in \mathcal{A}_{i-1} \right\} \tag{3.134}$$

By minimizing over the admissible values of $\dot{\lambda}$, the equations in (3.132) become:

$$\begin{aligned} t(0) &= 0 \\ t_{opt}(i) &= \min_{\dot{\lambda}(i) \in \mathcal{C}_i} \left[2\frac{\lambda(i) - \lambda(i-1)}{\dot{\lambda}(i) + \dot{\lambda}(i-1)} + t(i-1) \right] \end{aligned} \tag{3.135}$$

While $t_{opt}(N_i)$ represents the optimized function, the same function $t_{opt}(i)$ at a generic stage i is also called *optimal return function* and corresponds to the minimum value of the objective function if the process stopped at stage i.

It is worth remarking that this problem formulation is valid under the hypothesis that the torque constraints can be expressed as in (3.14) or in a similar manner which allows for the definition of $L(\lambda, \dot{\lambda})$ and $U(\lambda, \dot{\lambda})$. If the actuators were driven by the same power supply, as mentioned in Section 3.3.2.3, this might not be the case. As instance, if the torque constraint was expressed, as in [91], as

$$\boldsymbol{\tau} = [\tau_1, \tau_2, ..., \tau_n]^T \in E(\mathbf{q}, \dot{\mathbf{q}}) \tag{3.136}$$

the maximum and minimum pseudo-accelerations would not be defined, making the employment of the sets in (3.133) impossible. In this case, the vector of joint torques should be computed explicitly in order to verify constraint (3.136). Rather, if the coupling of the actuator torques was modeled as in (3.79), it would still be possible to define L and U, as shown in [82], and use the sets in (3.133).

A similar approach to trajectory optimization is proposed in [71], where the λ axis is replaced by the list of waypoints and the $\dot{\lambda}$ axis is replaced by the velocity of a non-stationary joint. Conceptually, there is no difference with the approach presented here, since there exists a one-to-one correspondence between a waypoint in the workspace (or in the joint space) and λ. By recalling (3.1), the same is true for $\dot{q}_i(\lambda)$ and $\dot{\lambda}$. With respect to the formulation discussed in this section, using the joint variables directly implies to verify the constraints on each kinematic and dynamic variable in play, e.g. velocity and torque of each joint.

In [72], the authors adopt a slightly different formulation based on the Hamilton-Jacobi-Bellman equation, to which they give an approximate solution using finite difference methods. This approach allows computing the optimal control sequence $\ddot{\lambda}$ and the execution time without calculating the phase-plane trajectory explicitly, that is found only at a later stage by integrating the optimal controls.

3.7.2 Algorithmic implementation

In light of the problem formulation above, we can define the time-optimal trajectory planning (TOTP) algorithm based on dynamic programming as in Algorithm 7.

As briefly mentioned in Section 3.7.1, at step 1, the vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{g}$ can be pre-computed on the basis of the parametrized trajectory (given either in the joint space or in the workspace), as discussed in Section 3.2.1. Since all the values of λ and $\dot{\lambda}$ are given beforehand, the pseudo-acceleration limits can also be pre-computed (step 3), as well as the constraints limiting the pseudo-velocity (step 2). If the problem is defined in a way that the manipulator starts from a rest condition, i.e. $\dot{\lambda}(0) = 0$, this can be easily imposed by configuring the set $\mathcal{A}_0$ accordingly. The reader may notice that here it is not required the explicit computation of the MVC, which is rather necessary with different algorithms. In fact, if $L_j(i)$ and $U_j(i)$ are available, the MVC condition can be

Algorithm 7 Time-optimal trajectory planning with dynamic programming

1: *Compute* $\mathbf{a}$, $\mathbf{b}$, $\mathbf{g}$ *from* $\mathbf{x}(\lambda)$, *or* $\mathbf{q}(\lambda)$
2: *Initialize* $\mathcal{A}_i, \forall i = 0..(N_i - 1)$
3: *Initialize* $\mathcal{B}_i(\dot{\lambda}_j), \forall i = 1..N_i, \forall j = 0..(N_j - 1)$
4: *Initialize* $\mathcal{C}_i = \varnothing, \forall i = 0..N_i$
5: *Initialize cost map* $t_{i,j} = +\infty, \forall i = 0..(N_i - 1), \forall j = 0..(N_j - 1)$
6: $t_{0,j} \leftarrow 0, \forall j = 0..(N_j - 1)$
7: $\mathcal{C}_0 \leftarrow \mathcal{A}_0$
8: **for** $i \leftarrow 0$ to $N_i - 1$ **do**
9: **for each** $\dot{\lambda}_j \in \mathcal{C}_i$ **do**
10: **for each** $\dot{\lambda}_k \in \mathcal{A}_{i+1}$ **do**
11: $\ddot{\lambda} \leftarrow \frac{\dot{\lambda}_k - \dot{\lambda}_j}{\Delta_\lambda} \dot{\lambda}_j$
12: **if** $\ddot{\lambda} \in \mathcal{B}_{i+1}$ **then**
13: $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_{i+1} + \{\dot{\lambda}_k\}$
14: *Compute instantaneous cost function* ϕ
15: **if** $t_{i,j} + \phi < t_{i+1,k}$ **then**
16: $t_{i+1,k} = t_{i,j} + \phi$
17: *Let* $\dot{\lambda}_j$ *at stage* i *be the predecessor of* $\dot{\lambda}_k$ *at stage* $i+1$
18: $t_{opt}(N_i) = t_{N_i,0}$
19: *Build function* $\dot{\lambda}(i)$ *of optimal pseudo-velocities by screening the predecessors map backward*

verified through the test $L_j(i) > U_j(i)$.

The pseudo-acceleration is computed in-line (step 11) for each pair of states $\mathbf{s}_j(i)$ and $\mathbf{s}_k(i+1)$. We remark that, under the rest condition $\dot{\lambda}(0) = 0$, the pseudo-acceleration needs to be computed with the same approximation as [81], that is

$$\ddot{\lambda} \sim \frac{d\dot{\lambda}}{d\lambda} = \frac{\dot{\lambda}_k - \dot{\lambda}_j}{\Delta_\lambda} \tag{3.137}$$

If $L \leq \ddot{\lambda} \leq U$ (step 12), the state $\mathbf{s}_k(i+1)$ is reachable and can be added to the set of nodes that will be visited at the next waypoint, i.e. step 13. Among all the states $\mathbf{s}_j(i)$, with variable j, that can reach $\mathbf{s}_k(i+1)$, a reference to the minimum-time one is saved (step 17), so that each reachable state $\mathbf{s}_k(i+1)$, with variable k, will have one and only one predecessor. Predecessors can be stored in a separate map of references or as pointers embedded in the nodes, depending on the selected implementation paradigm.

Since the dynamic programming algorithm computes the minimum-

time trajectory for each possible final state $\mathbf{s}_j(N_i)$, if the problem definition requires the manipulator to end its motion in a rest condition, i.e. $\dot{\lambda}(N_i) = 0$, the solution is immediately available by picking the related state from the vector of final states, as done at step 18.

In this formulation we assume that the grid nodes are connected through straight segments, i.e. first-order interpolation, but more complex functions can be used as proposed in [91, 14]. In particular, if the grid resolution is coarse, it might be convenient to perform an interpolation of an higher order. In such cases, the violation of the torque constraint (corresponding to a pseudo-acceleration constraint) might happen anywhere between two consecutive stages $\lambda(k)$ and $\lambda(k+1)$, implying that the test at step 12 is no longer sufficient to guarantee the feasibility, but the whole curve connecting the nodes must be tested.

Eventually, if constraints on the torque rates exist, Algorithm 7 should be modified to work on a three-dimensional grid (including the $\ddot{\lambda}$ axis) and to test the limits on the pseudo-jerk in (3.25), although the underlying logic remains unchanged [91]. The introduction of an additional dimension is also the solution proposed in [77], but the underlying method is the switching points technique in the phase plane. It should be noted that the employment of jerk and/or torque rate constraints becomes particularly important when the trajectories have to be executed on the real hardware, however the introduction of an additional dimension might make the problem intractable from the practical standpoint. This is the issue addressed in [14], where jerk and torque rate constraints are enforced in a planning scenario based on dynamic programming. Therein, continuous and differentiable profiles are generated in the phase plane through interpolation between two consecutive stages. The constraints of interest, including jerk and torque rate, are verified at selected check points along such profiles. This way, the search space does not need to be augmented and the complexity can be controlled without exiting the phase plane.

On the same path identified by [14], as an additional alternative to enlarging the state space, the pseudo-jerk should be computed according to a discrete approximation, that is

$$\dddot{\lambda} = \dot{\lambda}''\dot{\lambda}^2 + \left(\dot{\lambda}'\right)^2 \dot{\lambda} \tag{3.138}$$

With a backward Euler approximation, it can be computed as

$$\begin{aligned}\dddot{\lambda}(k) &= \frac{\dot{\lambda}'(k) - \dot{\lambda}'(k-1)}{\Delta_\lambda}\dot{\lambda}^2(k) + \left[\dot{\lambda}'(k)\right]^2 \dot{\lambda}(k) = \\ &= \frac{\dot{\lambda}(k) - 2\dot{\lambda}(k-1) + \dot{\lambda}(k-2)}{\Delta_\lambda^2}\dot{\lambda}^2(k) + \\ &+ \left[\frac{\dot{\lambda}(k) - \dot{\lambda}(k-1)}{\Delta_\lambda}\right]^2 \dot{\lambda}(k)\end{aligned} \tag{3.139}$$

where, at each step, $\dot{\lambda}(k)$, $\dot{\lambda}(k-1)$ e $\dot{\lambda}(k-2)$ are available through the predecessors map. With respect to [14], this is a simpler solution, but it is expected to provide less accurate results due to the discrete approximation. In both cases, when jerk and torque rate constraints are checked without augmenting the search space, the global optimality is compromised [14].

In Section 4.2, when we will deal with redundant robots, we will see that both augmenting the search space and generating differentiable profiles between waypoints is practically unfeasible and using simple Euler approximations is the only way to impose such kind of constraints. Nevertheless, in that case, because of the impossibility of calculating the pseudo-jerk explicitly, the discrete joint torque rates will be directly computed. Despite the coarse approximation, we will see that the planned trajectory can be executed on the real robot with satisfactory results.

3.7.3 Reference generation

Once the vector $\dot{\lambda}(i)$ is available for each i from the execution of Algorithm 7, the references for the robot controller can be gener-

ated from equations (3.1) and (3.2). The objective here is to derive the discrete approximations of such equations, assuming that the forward Euler method is adopted.

For a generic i, the pseudo-velocity $\dot{\lambda}(i)$ and parametric joint velocities $\dot{\mathbf{q}}(i)$ can be respectively written as:

$$\dot{\lambda}(i) = \frac{\lambda(i+1) - \lambda(i)}{t(i+1) - t(i)} \tag{3.140}$$

$$\dot{\mathbf{q}}(i) = \frac{\mathbf{q}(i+1) - \mathbf{q}(i)}{t(i+1) - t(i)} \tag{3.141}$$

From the first equation, we have that

$$t(i+1) - t(i) = \frac{\lambda(i+1) - \lambda(i)}{\dot{\lambda}(i)} \tag{3.142}$$

that, folded into (3.141), yields

$$\dot{\mathbf{q}}(i) = \frac{\mathbf{q}(i+1) - \mathbf{q}(i)}{\lambda(i+1) - \lambda(i)}\dot{\lambda}(i) = \mathbf{q}'(i)\dot{\lambda}(i) \tag{3.143}$$

that corresponds to the discrete form of equation (3.1).

As far as the parametric joint accelerations are concerned, we have that

$$\ddot{\mathbf{q}}(i) = \frac{\dot{\mathbf{q}}(i+1) - \dot{\mathbf{q}}(i)}{t(i+1) - t(i)} \tag{3.144}$$

By substituting (3.143) into the equation above, we get

$$\begin{aligned}
\ddot{\mathbf{q}}(i) &= \frac{\mathbf{q}'(i+1)\dot{\lambda}(i+1) - \mathbf{q}'(i)\dot{\lambda}(i)}{t(i+1) - t(i)} = \\
&= \frac{\mathbf{q}'(i+1)\dot{\lambda}(i+1) + \mathbf{q}'(i+1)\dot{\lambda}(i) - \mathbf{q}'(i+1)\dot{\lambda}(i) - \mathbf{q}'(i)\dot{\lambda}(i)}{t(i+1) - t(i)} = \\
&= \frac{\mathbf{q}'(i+1) - \mathbf{q}'(i)}{t(i+1) - t(i)}\dot{\lambda}(i) + \frac{\dot{\lambda}(i+1) - \dot{\lambda}(i)}{t(i+1) - t(i)}\mathbf{q}'(i+1)
\end{aligned} \tag{3.145}$$

Using again equation (3.142) for the first term and noticing that the second term includes the pseudo-acceleration, we can write

$$\ddot{\mathbf{q}}(i) = \mathbf{q}''(i)\dot{\lambda}^2(i) + \ddot{\lambda}(i)\mathbf{q}'(i+1) \tag{3.146}$$

that corresponds to the discrete form of (3.2) [94].

Once joint velocities and accelerations are available, the actuation torques can be algebraically computed from (3.7) through equations (3.143) and (3.146). Alternatively, they can be obtained directly from (3.8) by using the pre-computed vectors $\mathbf{a}(i)$, $\mathbf{b}(i)$ and $\mathbf{g}(i)$, the pseudo-velocity $\dot{\lambda}(i)$ and pseudo-acceleration $\ddot{\lambda}(i)$. In the same way, in case of viscous friction and state-dependent torques, the input voltages can be computed from (3.83).

Because of the discrete forms (3.143) and (3.146), the parametrized dynamics terms in (3.8) are

$$\mathbf{a}(i) = \mathbf{H}\big(\mathbf{q}(i)\big)\mathbf{q}'(i+1) \tag{3.147}$$

$$\mathbf{b}(i) = \mathbf{H}\big(\mathbf{q}(i)\big)\mathbf{q}''(i) + \big(\mathbf{q}'(i)\big)^T\mathbf{C}\big(\mathbf{q}(i)\big)\mathbf{q}'(i) \tag{3.148}$$

that are also used in the pre-processing step 1 of Algorithm 7.

Lastly, in this formulation, it should be noted that the forward Euler equation in (3.140) is not coherent with (3.132), the latter corresponding to a trapezoidal approximation. With a large discretization step, this misalignment yields non-negligible errors on the final control inputs [94]. However, if a forward Euler approximation was used even for the evaluation of the cost function, specific approximations should be foreseen at the boundaries, when the pseudo-velocity equals zero. On the other hand, if a trapezoidal approximation was consistently used for all computations, numerical oscillations would be likely to arise.

3.7.4 Application to a 2R planar robot

The TOTP dynamic programming algorithm is applied to the same use case as Section 3.5. The sampling intervals are set in

such a way that $N_i = 150$ and $N_j = 10^4$. As remarked in [91, 71], when discretizing the axes λ and $\dot{\lambda}$, it is important to choose N_j and N_i such that the sets $\mathcal{B}_i(\dot{\lambda}_j)$ are not empty, with fixed i, for all the possible values of j. Usually, this requires $N_j >> N_i$. On the other hand, if N_j is too large, the execution time of the dynamic programming algorithm considerably grows. As far as the pseudo-velocity upper bound is concerned, in this example, it is set to $\dot{\lambda}_M = 10$ because we have a prior knowledge of the phase plane trajectory (computed with the switching points technique). Otherwise, the considerations of Section 3.7.1 hold. It is also worth mentioning that, with these values of the discretization steps, the grid nodes can be connected by straight segments and it is not necessary to use more complex curves, thus the formulation of sections 3.7.1 and 3.7.2 is valid. Refer to [14] for an implementation that considers more complex interpolations, which allow to drastically reduce the number of nodes.

The time-minimum phase plane trajectory computed with Algorithm 7 is reported in Figure 3.27, together with the MVC and, for the sake of comparison, the time-minimum trajectory computed with the switching points technique of Algorithm 4. The trajectory tracking time corresponding to the optimal solution is $t_{opt} = 0.875$ s. The algorithm, implemented in MATLAB®, executed in ~ 2 minutes on a 64-bit Windows 10 OS running on an Intel® Core™ i7-7600U CPU @ 2.80GHz. No parallel execution model was used in the tests.

It is interesting to notice that the optimal solution is extremely close to the switching point analogue and that the algorithm behaves better to the left of the critical point. The reason must be found, as for the genetic algorithm, in the fact that values of the pseudo-acceleration other than U and L can be selected by the algorithm, although the resolution of pseudo-accelerations is much lower in this case and is dependent on N_j. Like the genetic algorithm, the DP algorithm is able to gain some area over the switching points solution because of the discrete approximation in the neighborhood of the U-to-L switches, which is anyhow negli-

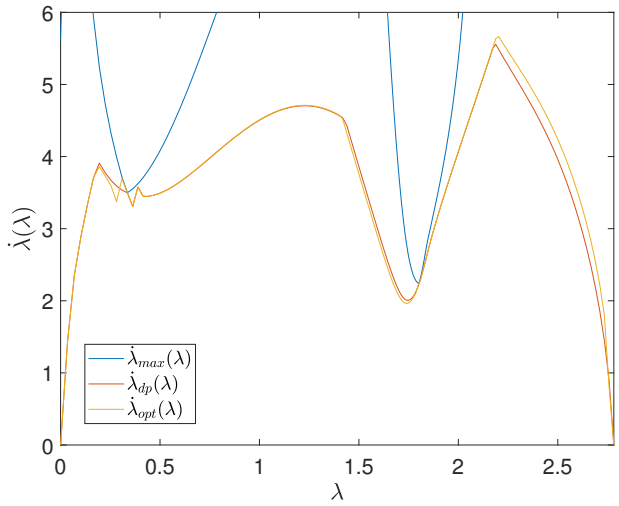


Figure 3.27: MVC (blue) and phase plane trajectories computed with switching points (yellow) and dynamic programming (red)

gible. As evident from the shape of the phase-plane trajectory in the proximity of the critical point, jitters in the torques are not completely eliminated. For a more extensive explanation of this phenomenon, refer to Section 3.5.1.

3.8 Trajectory tracking

All the resolution techniques analyzed in this chapter allow the trajectory to be planned at the limits of the robot capacity. However, when it comes to executing these time-optimal trajectories, additional issues may arise. Let us recall that at least one actuator has to saturate for every single point of the path. Controlling the motors close to their limits, so that they saturate, reserves no margin for the controller to compensate for the unmodeled dynamics (like elasticity in the joints or the controller dynamics themselves) and uncertainties in the modeled dynamics. Some very simple, as well as low-performance techniques to account for this issue are

[95]:

- to adopt conservative torque bounds;
- to estimate the uncertainty in the dynamic parameters and plan in the worst-case scenario;
- to respect the execution time while allowing for small trajectory deviations.

The first two solutions are flat, meaning that they apply uniformly along the path, causing an overall degradation of performances. The latter is only valid for some applications that allow for small path deviations.

More elaborate solutions exist and can be classified in two categories: trajectory pre-shaping and online trajectory scaling, that we discuss in this section.

3.8.1 Trajectory pre-shaping

Often, in commercial robots, the primary controller that receives the references from the user and generates commands for the actuators is delivered with the robot itself and cannot be modified. Usually, the dynamic parameters of the robot are available or can be estimated with some uncertainty. In these situations, the references generated by a planning system can take into account the robot dynamics, but not the controller's. Nominally, this is not an issue, as the controller has the specific function to allow for trajectory tracking and compensate for uncertainties and unmodeled dynamics. However, when the robot is driven close to its torque limits, such function is lost, resulting in poor tracking. As demonstrated in [96], better tracking performances can be obtained with a simple feed-forward scheme that is designed to "cancel" the controller dynamics.

The proposed solution is to manipulate (or pre-shape) the planned trajectory to include a correction term that accounts for the un-

modeled dynamics of the controller. This is still an off-line process, that does not require the primary controller to be modified. When applied to robotic manipulators, the assumption is made that an independent joint control is implemented according to a SISO model. The high-level scheme is reported in Figure 3.28, where $P(s)$ and $C(s)$ are the transfer functions of the plant and controller respectively. The Laplace transform of the desired output $Y_d(s)$ is multiplied by the chain $P^{-1}(s)C^{-1}(s)$ to obtain the additive correction term $Z(s)$ so as to generate the actual reference $R(s)$ for the controller.

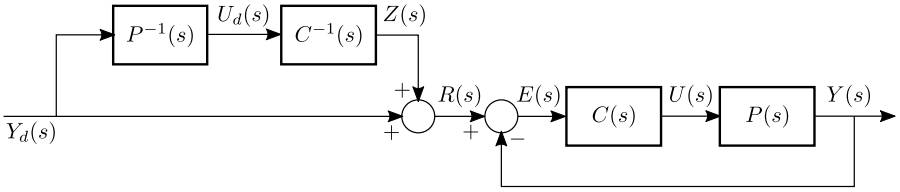


Figure 3.28: Trajectory pre-shaping control scheme [96]

The reader may recognize that the closed-loop transfer function of the system in Figure 3.28 is the identity, meaning that, in theory, the controller dynamics can be removed. Clearly, in practice, this scheme can only be implemented if both the plant and the controller dynamics are known. If the latter is unknown, but at least its structure is known, its parameters can be estimated with a learning approach based on an iterative optimization. However, as remarked in [96], its performances are heavily dependent on an initial guess. If this is wrong, the tracking accuracy is even worse than a simple feed-forward scheme without pre-shaping. Also, the controller has to respect specific properties for this method to be implemented. In particular, it must not have unstable zeros and its inverse must still be a causal system.

Assuming that all the hypotheses are respected, a better tracking can be achieved. Residual errors have to be anyhow expected, which depend on the estimation of the controller's parameters and uncertainty associated with $P(s)$. In [97], the work [96] is extended to underline that the residual error, in the experiments considered

therein, is mainly due to the unmodeled dynamics of the brushless motors' friction. The issue is addressed by extending $P(s)$ with a linear model of the viscous friction, so that it is invertible and can be used in pre-shaping. The Authors demonstrate that the residual error can be further reduced.

For complex dynamics and longer paths, the uncertainties accumulate and open-loop control cannot guarantee a bounded error [98]. In these cases, a feedback scheme has to be adopted.

3.8.2 Trajectory scaling

For many industrial tasks, tracking accuracy is vital and, therefore, most of the effort in research has been put in designing controllers that sacrifice time to achieve better accuracy. This is the principle of online trajectory scaling. To this end, several techniques exist.

3.8.2.1 Scaling of the phase plane trajectory

In [95] the idea is presented to design a secondary controller on top of the primary one. Here, the primary controller is the one that generates the motor torques to be sent to the joint actuators. The assumption is made that it is accessible and can be modified. The scheme is presented for two different primary controllers: feed-forward torque controller and computed torque controller.

Let us assume that a feed-forward torque controller is used. The commanded torques are computed from position and velocity feedback as

$$\boldsymbol{\tau} = \mathbf{H}(\mathbf{q}_d)\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_d + \mathbf{d}(\dot{\mathbf{q}}_d) + \mathbf{g}(\mathbf{q}_d) + \mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}} \quad (3.149)$$

where $\mathbf{q}_d, \dot{\mathbf{q}}_d$ and $\ddot{\mathbf{q}}_d$ are the vectors of desired joint positions, velocities and accelerations respectively, $\mathbf{d}(\dot{\mathbf{q}}_d)$ is the $n \times 1$ vector of joint friction torques, $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$, $\dot{\mathbf{e}} = \dot{\mathbf{q}}_d - \dot{\mathbf{q}}$, $\mathbf{K}_p$ and $\mathbf{K}_d$ are the

matrices of proportional and derivative gains respectively. $\mathbf{q}$ and $\dot{\mathbf{q}}$ are intended here as measured quantities.

The model in (3.149) can be parametrized in the same way as the robot's dynamic model, yielding

$$\boldsymbol{\tau} = \mathbf{b}_1(\lambda, \mathbf{q})\ddot{\lambda} + \mathbf{b}_2(\lambda, \dot{\lambda}, \mathbf{q}, \dot{\mathbf{q}}) \tag{3.150}$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are $n \times 1$ vectors that depend on the reference, but also on the tracking error. Thus, an updated limit for the pseudo-acceleration can be computed from

$$\boldsymbol{\tau}_{min} \leq \mathbf{b}_1\ddot{\lambda} + \mathbf{b}_2 \leq \boldsymbol{\tau}_{max} \tag{3.151}$$

The larger the tracking error is, the stricter will be the new limits on the pseudo-acceleration. According to [95], they can be used in the secondary controller as in the scheme of Figure 3.29. The actual pseudo-acceleration that is passed to the primary controller is updated online to limit the slope of the phase plane trajectory. Nominally, $\gamma = 1$ and the error $\dot{\lambda}_d^2 - \dot{\lambda}^2$ is zero, so that the desired pre-planned pseudo-acceleration $\ddot{\lambda}_d$ can be delivered to the primary controller, as long as the limits L and U, that are computed on the basis of measured data, do not clip the signal. If the actuators saturate, a little scaling can be imposed by tuning γ adaptively.

The proposed technique works in general, not only for time-optimal trajectories, and solves two issues at the same time:

- when the pseudo-acceleration reference is not compliant with the error in the pseudo-velocity, the former is saturated up to reaching the desired pseudo-velocity; the controller limits the slope of the phase plane trajectory so that it becomes feasible;
- when a time-optimal trajectory is being executed, the pseudo-velocity is modified (through scaling of pseudo-acceleration) to be trackable.

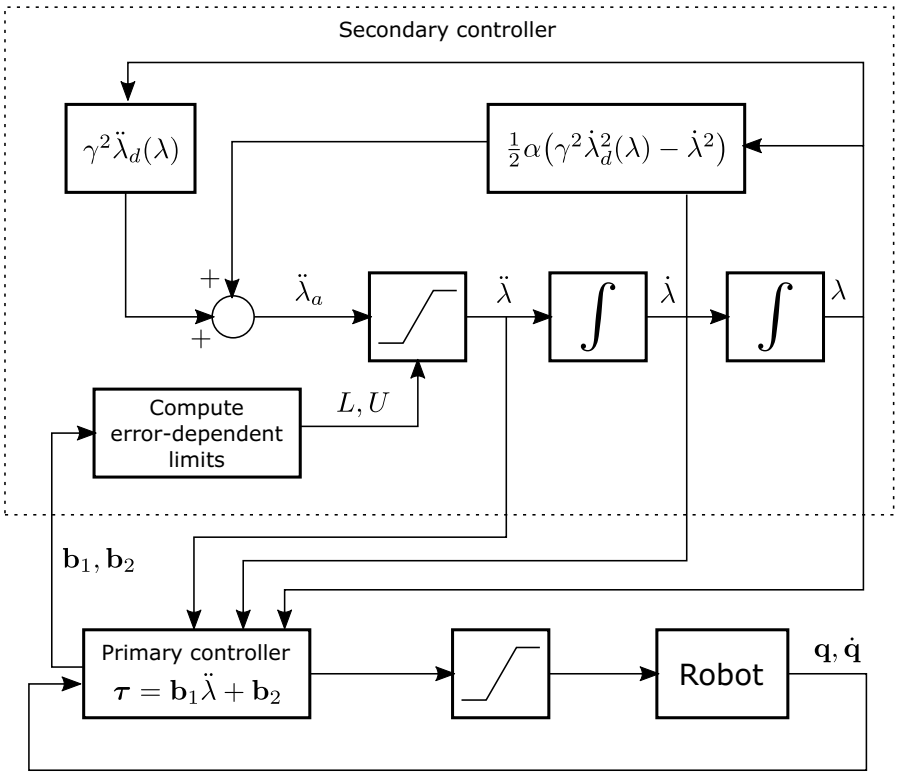


Figure 3.29: Trajectory scaling with primary and secondary controllers and adaptive pseudo-acceleration [95]

The advantage of this technique is that, although the controller has to be parametrized, its gains $\mathbf{K}_p$ and $\mathbf{K}_v$ do not need re-tuning. Thus the nominal properties of robustness are preserved. The functioning of this method on a real industrial robot is illustrated in [99].

A control scheme very similar to the one of Figure 3.29, is proposed in [5], where a different parametrization of the path is proposed. Rather than using the simple scalar information of the curvilinear coordinate λ, the Authors propose to parametrize the path through its tangential and normal components. The former is still the curvilinear coordinate, while the latter represents the vector of all directions along which the manipulator can deviate from the

reference path.

The two components (tangential and normal) are controlled independently. The normal component has priority, so that, when the controller has to keep the manipulator on the path, the tangential velocity can be limited. On the other hand, when the tracking error is small, the controller can increase the tangential component so as to gain speed. The normal component control is performed through a simple PID, while that of the tangential component is performed equivalently to [95]. In this case though, the pseudo-acceleration limits are regulated on the basis of the normal components error.

As for [95], the technique proposed in [5] is not based on any feature of time-optimal solutions, like the bang-bang control law. For this reason, it can also be applied in different contexts.

The principle of Figure 3.29 is also at the basis of the solution proposed in [100]. Assuming a computed torque controller, the given time law $\lambda(t)$, possibly planned with a time-optimal algorithm, is multiplied to a correction term $c(t)$, so that the actual joint-space reference trajectory is given by

$$\mathbf{q}_d(t) = \mathbf{q}_d\big(c(t)\lambda(t)\big) \tag{3.152}$$

The primary controller is then parametrized to take into account this factor, yielding:

$$\boldsymbol{\tau} = \mathbf{b}_1(c, \lambda, \mathbf{q})\ddot{c} + \mathbf{b}_2(c, \dot{c}, \lambda, \dot{\lambda}, \mathbf{q}, \dot{\mathbf{q}}) \tag{3.153}$$

The control law is defined on c, equivalently to [95] and even the limits c_{max} and c_{min} are computed from the equation above by using the torque limits and the measured data.

It is worth remarking that all three schemes recalled here can only be implemented if the robot primary controller is accessible and can be modified.

3.8.2.2 Disturbance models

Another category of online time-scaling controllers is based on modeling the unknown dynamics of the controller-robot system as disturbances. In [98], they are modeled as additive disturbances $\ddot{\boldsymbol{\delta}}$ on the accelerations $\ddot{\mathbf{q}}$, so that

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{H}(\mathbf{q})\ddot{\boldsymbol{\delta}} = \boldsymbol{\tau} \tag{3.154}$$

should represent an "exact model" of the robot, where the vector $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$ includes all the centrifugal, Coriolis, friction and gravity terms. In this case, the problem becomes to estimate the disturbances $\ddot{\boldsymbol{\delta}}$ due to the model uncertainties.

One possibility is to estimate them as

$$\ddot{\boldsymbol{\delta}} = \mathbf{H}^{-1}(\mathbf{q})\big(\boldsymbol{\tau} - \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})\big) \tag{3.155}$$

where the quantities $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ and $\boldsymbol{\tau}$ are measured from a trail execution. Assuming that $\ddot{\boldsymbol{\delta}}$ will be similar for a similar trajectory, and assuming a computed torque controller, the error can be predicted through the linear relationship:

$$\ddot{\mathbf{e}}(t) + \mathbf{K}_d\dot{\mathbf{e}}(t) + \mathbf{K}_p\mathbf{e}(t) = \ddot{\boldsymbol{\delta}} \tag{3.156}$$

and, as a consequence, the desired torques $\boldsymbol{\tau}_d$ can be computed as

$$\boldsymbol{\tau}_d = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_\delta \tag{3.157}$$

where

$$\boldsymbol{\tau}_\delta = \mathbf{H}\big(\mathbf{q}\big)(\mathbf{K}_d\dot{\mathbf{e}} + \mathbf{K}_p\mathbf{e}\big) \tag{3.158}$$

are the *corrective torques* necessary to cancel the disturbances.

The time-optimal planning session can be executed directly with the model in (3.157) that considers the disturbance torques $\boldsymbol{\tau}_\delta$. This is equivalent to saving some margin during planning that allows to control even in presence of uncertainties and actuator saturation. This feed-forward term is coupled, in [98], with a time-scaling controller similar to the one in Figure 3.29, where

the pseudo-acceleration limits L and U are computed through a controllability analysis in the phase plane, while $\ddot{\lambda}_a$ only contains the pre-planned term.

Another example of managing uncertainties as disturbances is provided in [101]. Starting from a standard dynamic model like

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{d}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{3.159}$$

uncertainties are modeled as disturbances on torques, i.e.

$$\hat{\mathbf{H}}\ddot{\mathbf{q}} + \boldsymbol{\tau}_\delta(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \boldsymbol{\tau} \tag{3.160}$$

$\hat{\mathbf{H}}$ is a diagonal inertia matrix with constant terms, estimated by evaluating the frequency response of each single axis independently, while $\boldsymbol{\tau}_\delta$, in a similar manner as before, are the disturbance torques including the remaining dynamics, the coupling effects and the payload uncertainty. The elements of $\boldsymbol{\tau}_\delta$ are given by

$$\tau_{\delta,i} = \sum_{j=1, j\neq i}^{n} H_{ij}(\mathbf{q})\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} C_{ijk}\dot{q}_j\dot{q}_k + g_i + d_i + \left(H_{ii}(\mathbf{q}) - \hat{H}_{ii}\right)\ddot{q}_i \tag{3.161}$$

With the model in (3.160), it is possible to design a *disturbance observer* for each single joint, so that $\boldsymbol{\tau}_\delta$ can be estimated and summed to the commanded torques to cancel it. Under this hypothesis, the system to control is simply given by

$$\hat{\mathbf{H}}\ddot{\mathbf{q}} = \boldsymbol{\tau} \tag{3.162}$$

Being the system linear and decoupled, a simple control scheme can be used, e.g. PD control, as shown in Figure 3.30.

The disturbance observer can make the system unstable, while, in [101], the stability is guaranteed through the introduction of an additional saturation element.

For time-optimal trajectories, the disturbance observer might not work properly if the disturbance is bigger than the physical torque

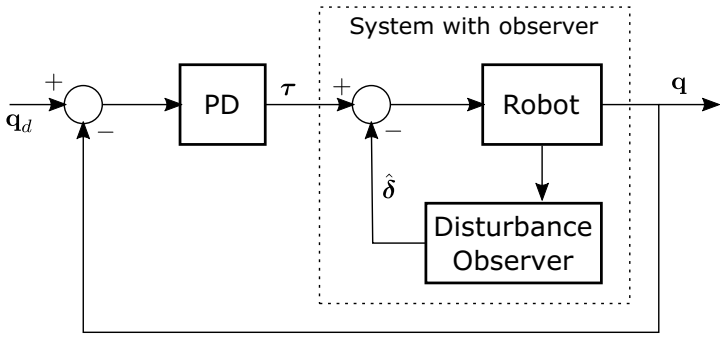


Figure 3.30: Control loop with disturbance observer [101]

limit. For this reason, the scheme in Figure 3.30 should be extended with a module performing online scaling. As for the other scaling techniques, it is based on the definition of new acceleration limits (in this case, in the Cartesian space rather than in the phase space) and of an additional Cartesian acceleration correction term $\ddot{\mathbf{x}}_c$. Such module is only activated when actuator saturation is detected. The overall scheme is shown in Figure 3.31, where a saturation is also shown between the PD controller and the plant indicating that the maximum output the controller can deliver is dependent on the disturbance, i.e.

$$u_{max} = \boldsymbol{\tau}_{max} - \hat{\boldsymbol{\delta}} \tag{3.163}$$

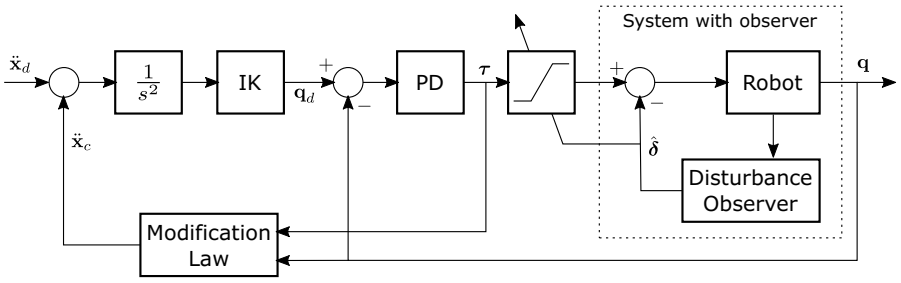


Figure 3.31: Control loop with disturbance observer and online trajectory scaling [101]

In [102], a control scheme is proposed for generic dynamic systems controlled with torques close to saturation. It foresees the addition

of a feedback controller with anti-windup and of an online trajectory planner. The former is necessary to ensure stability, reject disturbances and measurement noise and reduce the degradation of performance in presence of saturation. The latter allows to keep the tracking accuracy within a defined tolerance. The tracking error is associated to the contouring error, that is the difference between the actual position at a given time and the closest point on the path, regardless of time. The error is controlled through the introduction of additional waypoints that are inserted every time a torque is predicted to saturate. An online interpolation technique is employed. More waypoints have the effect of scaling the velocity along the path. The overall scheme is shown in Figure 3.32.

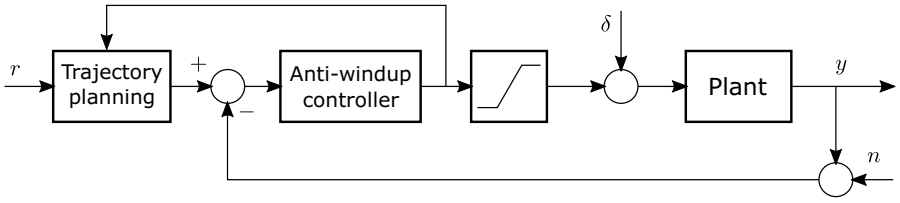


Figure 3.32: Control loop with anti-windup controller and contouring-error-based online trajectory planner [102]

Even though the scheme is sufficiently generic, in [102], the trajectory planner is based on a decoupled dynamic model and results show some chattering in the controller output due to instantaneous changes in velocity when waypoints are added. The Authors only demonstrate their methodology on a 2-axes Cartesian position system.

3.8.2.3 Other techniques

When a time-optimal trajectory is being executed for a non-redundant manipulator, one actuator is in saturation while the others adjust to keep the end-effector on the path. Since control with saturation is prone to poor tracking performances, the non-saturating

actuators can be used to reduce the tracking error in the Cartesian space. This is the underlying idea in [103]. The control is based on the minimization of a cost function defined on the path deviation and normal acceleration, suitably weighted. Perturbations are defined that are associated to the switching times, final times, and output of the controllers corresponding to the non-saturating joints. The advantage of this technique is that the gains associated with such perturbations can be pre-computed with a linear-quadratic problem, so as to reduce the work that the controller has to perform online. Compared to a classic feed-forward scheme, the proposed method allows respecting the path constraint and achieving tracking times comparable to those planned.

A much more simple controller, with respect to the ones analyzed in this section, is the one proposed in [89]. It is a *sliding mode controller* (SMC) whose task is to provide corrections to the nominal torques to take into account the dynamic model uncertainties, that are supposed to be large. The scheme is shown in Figure 3.33.

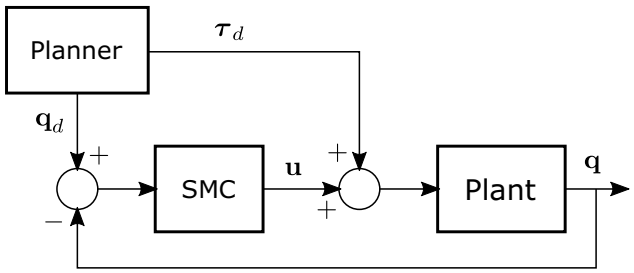


Figure 3.33: Feed-forward planner and sliding mode controller (SMC) [89]

If the dynamic model is very inaccurate, bang-bang control is prone to fail. Thus, the idea is to generate the desired torques $\boldsymbol{\tau}_d$ with such a model, but only imposing kinematic constraints, like velocity, acceleration and jerk limits. Motivated by the same assumption, the SMC is not model-based, but its parameters are found with an evolutionary technique, called Evolution Strategy (ES). The advantage of ES is that it is flexible with respect to

notable variations of dynamics, as when the manipulator has to carry a payload of non-negligible mass and size.

Chapter 4

Time-optimal planning of redundant robots

4.1 Existing problem formulations

Let us recall that kinematic redundancy occurs when the assigned task is characterized by m equality constraints, while n, the number of degrees of freedom in the robotic system, is strictly greater than m. The difference $r = n - m$ is referred to as *degree of redundancy* and corresponds to the dimension of the subspace of the configuration space into which it is possible to find the infinite joint-space solutions to the inverse kinematic problem when the task is assigned.

For kinematically redundant systems, a question naturally arises on how the extra degrees of redundancy can be exploited to the purposes of time-optimal planning. Intuitively, since the pseudo-velocity $\dot{\lambda}$ constitutes an optimization variable (or, in other words, a dimension along which the optimization is possible), the presence of r additional degrees of freedom could be used to the same purposes and they undoubtedly increase the dimensionality of the space where the optimal solution is found. Nevertheless, kinematically redundant systems are nowadays extremely common in

a multitude of applications, thus investigating their capability of performing time-optimal motion is of utmost interest.

The techniques performing time-optimal planning for redundant robots along prescribed paths differ in

- the solution optimality, i.e. locally-optimal or globally-optimal;
- the way the kinematic path constraint is handled within the problem formulation: some techniques directly use the position kinematics and embed it in the parametrization of dynamics, as done in Section 3.2.2 for non-redundant robots, others use first- to fourth-order derivatives of position kinematics, which explicitly appear as constraints of the problem;
- the adopted analytic or numeric resolution technique;
- the possibility of producing solutions that are smooth enough to be directly used as references for real robot controllers.

4.1.1 Problem formulation with calculus of variations

If the task is assigned directly in the joint space, all the theory of non-redundant manipulators holds without modifications. The path parametrization respects the usual form of (3.1) and (3.2) and the problem is that in (3.54). On the contrary, if the task is more commonly defined in the task space, the Jacobian $\mathbf{J}$ is not invertible, implying that parametrizations (3.5) and (3.6) are no longer valid.

Since it is not possible to determine $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ as functions of the path parameter λ because of the non-square Jacobian, the state equation in (3.54) no longer represents the problem at hand. In order to fully describe the system, the state $\mathbf{s}$ must be extended

to include $\mathbf{q}$ and $\dot{\mathbf{q}}$ [83], i.e.:

$$\mathbf{s}(t) = \begin{bmatrix} \mathbf{q}^T & \lambda & \dot{\mathbf{q}}^T & \dot{\lambda} \end{bmatrix}^T \tag{4.1}$$

Since the dynamic equations cannot be parametrized if the path is given in the task space, they must be treated separately. For convenience, if the dynamic model is as in (3.7), define $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})$. If viscous friction is modeled, as in (3.83), define $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{q}}^T\mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{B}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q})$. Rewrite the dynamic equations as

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} \tag{4.2}$$

In addition, consider the new definition of the state in (4.1), and rewrite the kinematic constraints as:

$$\boldsymbol{\phi}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{q}, \lambda) = \mathbf{x}(\lambda) - \mathbf{k}(\mathbf{q}) = \mathbf{0} \tag{4.3}$$

where $\mathbf{x}(\lambda)$ is the assigned task-space path, belonging to the class of twice continuously-differentiable functions, and $\mathbf{k} : \mathbb{R}^n \to \mathbb{R}^m$ is the kinematic mapping from the joint space to the task space. The first-order and second-order kinematic constraints can then be respectively written as:

$$\begin{aligned} \boldsymbol{\chi}(\mathbf{s}) &= \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{s}}\dot{\mathbf{s}} \\ \boldsymbol{\psi}(\mathbf{s}) &= \frac{\partial \boldsymbol{\chi}}{\partial \mathbf{s}}\dot{\mathbf{s}} \end{aligned} \tag{4.4}$$

The constraints above are indeed equivalent to (4.3), provided that $\boldsymbol{\phi}(\mathbf{s}_0) = \mathbf{0}$ and $\boldsymbol{\chi}(\mathbf{s}_0) = \mathbf{0}$ are satisfied, with $\mathbf{s}_0 = \mathbf{s}(t = 0)$.

Let $\phi_i(\mathbf{s})$ be one of the equations in (4.3) and $\boldsymbol{\phi}_r(\mathbf{s}) \in \mathbb{R}^{m-1}$ the vector of the remaining ones, representing a *reduced* task, with $\boldsymbol{\chi}_r(\mathbf{s})$ and $\boldsymbol{\psi}_r(\mathbf{s})$ its first and second time derivatives. Let us explicitly derive ϕ_i twice with respect to time. We obtain:

$$-\mathbf{J}_i\ddot{\mathbf{q}} + x_i'\ddot{\lambda} - \dot{\mathbf{J}}_i\dot{\mathbf{q}} + x_i''\dot{\lambda}^2 = 0 \tag{4.5}$$

where $\mathbf{J}_i$ is the i-th row of the manipulator Jacobian and x_i is the i-th component of the assigned path $\mathbf{x}(\lambda)$.

The dynamic model (4.2) and the constraint (4.5) can be bundled together to yield:

$$\begin{bmatrix} \mathbf{H} & 0 \\ -\mathbf{J}_i & x_i' \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \ddot{\lambda} \end{bmatrix} + \begin{bmatrix} \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) \\ -\dot{\mathbf{J}}_i \dot{\mathbf{q}} + x_i'' \dot{\lambda}^2 \end{bmatrix} = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0}^T \end{bmatrix} \boldsymbol{\tau} \tag{4.6}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix.

By inverting the equation above and using the state vector $\mathbf{s}$, the following dynamic system is derived, having input $\mathbf{u} = \boldsymbol{\tau}$:

$$\dot{\mathbf{s}} = \mathbf{a}_e(\mathbf{s}) + \mathbf{b}_e(\mathbf{s})\mathbf{u} \tag{4.7}$$

where

$$\begin{aligned} \mathbf{a}_e(\mathbf{s}) &= \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\lambda} \\ -\begin{bmatrix} \mathbf{H} & 0 \\ -\mathbf{J}_i & x_i' \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) \\ -\dot{\mathbf{J}}_i \dot{\mathbf{q}} + x_i'' \dot{\lambda}^2 \end{bmatrix} \end{bmatrix} \\ \mathbf{b}_e(\mathbf{s}) &= \begin{bmatrix} \mathbf{0} \\ \begin{bmatrix} \mathbf{H} & 0 \\ -\mathbf{J}_i & x_i' \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0} \end{bmatrix} \end{bmatrix} \end{aligned} \tag{4.8}$$

The subscripts in the vectors above indicate that this is an *extended* dynamic system, as it groups the manipulator's dynamic model with a kinematic equation that depends on the assigned task.

By inserting (4.7) in (4.4), the dependence of the differential con-

straints on the input vector $\mathbf{u}$ becomes explicit:

$$\boldsymbol{\chi}_r(\mathbf{s},\mathbf{u}) = \frac{\partial \boldsymbol{\phi}_r}{\partial \mathbf{s}}\left(\mathbf{a}_e(\mathbf{s}) + \mathbf{b}_e(\mathbf{s})\mathbf{u}\right) \tag{4.9}$$

$$\boldsymbol{\psi}_r(\mathbf{s},\mathbf{u}) = \frac{\partial \boldsymbol{\chi}_r}{\partial \mathbf{s}}\left(\mathbf{a}_e(\mathbf{s}) + \mathbf{b}_e(\mathbf{s})\mathbf{u}\right) \tag{4.10}$$

where only the reduced task has been considered.

The reader may recognize that the system described by (4.2) and (4.3) is indeed equivalent to that represented by (4.7) and (4.10). The problem of finding the minimum time trajectory can thus be formalized as follows:

$$\begin{aligned} &\min_{\mathbf{u}} \int_0^{t_f} dt \\ \text{s.t.}\quad &\dot{\mathbf{s}} = \mathbf{a}_e(\mathbf{s}) + \mathbf{b}_e(\mathbf{s})\mathbf{u} \\ &\boldsymbol{\tau}_{min} \leq \mathbf{u} \leq \boldsymbol{\tau}_{max} \\ &\boldsymbol{\psi}_r(\mathbf{s},\mathbf{u}) = \mathbf{0} \\ \text{w/ b.c.}\quad &\mathbf{s}(0) = \mathbf{s}_0,\ \mathbf{s}(t_f) = \mathbf{s}_f \\ &\boldsymbol{\phi}(\mathbf{s}_0) = \mathbf{0},\ \boldsymbol{\phi}(\mathbf{s}_f) = \mathbf{0} \end{aligned} \tag{4.11}$$

where $\mathbf{s}_0 = \mathbf{s}(t=0) = \left[\mathbf{q}^T(0), 0, \mathbf{0}^T, 0\right]^T$ and $\mathbf{s}_f = \mathbf{s}(t=t_f) = \left[\mathbf{q}^T(t_f), \Lambda, \mathbf{0}^T, 0\right]^T$.

A numerical solution to the problem above, slightly modified in the input (torque) constraints, is given in [83] by making use of the Extended Pontryagin's Maximum Principle (EPMP). Assuming that $\mathbf{u}^*$ are the optimal control inputs and $\mathbf{x}(t)$ is a regular time-optimal trajectory, it is possible to demonstrate that at most $n-m+1$ actuators are at their bounds. If the control problem (4.11) is non-singular, exactly $n-m+1$ actuators saturate. Moreover, these results can be extended to point-to-point time-optimal control problems with constraints on the state and/or the input variables [104].

On the other hand, in [105], the authors demonstrate that also a minimum number of saturating torques exists. They consider an

extension of the model in (4.2), namely

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{T}(\mathbf{q})\boldsymbol{\tau} + \boldsymbol{\chi}^T(\mathbf{q})\mathbf{w} \tag{4.12}$$

where $\mathbf{q} \in \mathbb{R}^n$ is the vector of configuration coordinates of the system, including both active and passive joints, $\boldsymbol{\tau} \in \mathbb{R}^p$ is the vector of input torques, with $p \leq n$, $\mathbf{T} \in \mathbb{R}^{n \times p}$ is the matrix mapping the torques onto the configuration coordinates, $\boldsymbol{\chi} \in \mathbb{R}^{n \times v}$ is the set of differential constraints (holonomic or non-holonomic), such that $\boldsymbol{\chi}(\mathbf{q})\dot{\mathbf{q}} = 0$ and $\mathbf{w} \in \mathbb{R}^v$ is the vector of internal forces necessary to respect the constraints in $\boldsymbol{\chi}$. The model is rather generic and can describe a large class of systems: redundant and non-redundant, as well as closed chains where the control of internal forces is of utmost importance.

By using such a dynamic description, it is demonstrated in [105] that the minimum number of torques and/or internal forces brought to saturation in a time-optimal control problem is $v+p+1-n$. It is interesting to notice that this result is task-independent and thus, independent of the degrees of redundancy. If $p = n$ and $v = 0$, i.e. all joints are actuated and there are no differential constraints, there would be only one actuator in saturation, finding again the single-torque bang-bang control of Section 3.2.4. In conclusion, for a redundant robotic system without internal forces to control and all joints actuated, there is at least one and no more than $n - m + 1$ actuators at their bounds.

4.1.2 Locally-optimal redundancy resolution in time-optimal planning

The EPMP formulation of Section 4.1.1 is characterized by the same drawbacks of the non-redundant case. Among those, it is especially awkward to deal with generic constraints, as those discussed in Section 3.2.3. For this reason, a simpler, but also sub-optimal technique is preferred in several real-world applications, which is based on the projection of joint-space velocities in the null-space of the Jacobian. Let us recall once again that, if

the manipulator is redundant, (3.3) is not valid because $\mathbf{J}^{-1}$ does not exist. However, a general solution to the first-order inverse kinematics can be written by making use of the Moore-Penrose pseudo-inverse:

$$\mathbf{q}' = \mathbf{J}^{\dagger}\mathbf{x}' + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\boldsymbol{\gamma}' \tag{4.13}$$

where $\mathbf{J}^{\dagger} = \mathbf{J}^T \left(\mathbf{J}\mathbf{J}^T\right)^{-1}$ is the unweighted Moore-Penrose pseudo-inverse, $\boldsymbol{\gamma}' \in \mathbb{R}^n$ is a vector of additional joint-space pseudo-velocities and $\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}$ is an $n \times n$ matrix projecting $\boldsymbol{\gamma}'$ in the null-space of the Jacobian, producing internal motion. Usually $\boldsymbol{\gamma}' = -k\frac{\partial G}{\partial \mathbf{q}}$, where k is a gain factor and G is a performance index to optimize.

Alternatively, the joint-space solution can be modified by the effect of a weighted pseudo-inverse, i.e.

$$\mathbf{q}' = \mathbf{J}_W^{\dagger}\mathbf{x}' \tag{4.14}$$

where $\mathbf{J}_W^{\dagger} = \mathbf{W}^{-1}\mathbf{J}^T \left(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T\right)^{-1}$ and $\mathbf{W}$ is a symmetric, positive definite, possibly configuration-dependent weight matrix.

Both forms (4.13) and (4.14) can be employed to parametrize the joint velocities and accelerations with respect to the path parameter λ, when $\mathbf{x}(\lambda)$ is given. In particular (4.13), or (4.14), is differentiated once with respect to the path parameter to yield $\mathbf{q}''$. Both $\mathbf{q}'$ and $\mathbf{q}''$ can then be inserted in (3.1) and (3.2) to obtain $\dot{\mathbf{q}}(\lambda, \dot{\lambda})$ and $\ddot{\mathbf{q}}(\lambda, \dot{\lambda}, \ddot{\lambda})$. Being the parametrization complete, the problem formulation is the same as discussed for non-redundant manipulators and a time-optimal solution can be obtained with one of the techniques of Section 3.4, Section 3.6 or Section 3.7.

If one wants to exploit the redundancy for time-optimal planning, the problem is simply reduced to finding a suitable form for the performance index G or the weight matrix $\mathbf{W}$. The former method is investigated in [6], while the latter in [7]. In both cases, the free parameters of the problem are heuristically selected so as to maximize the acceleration/deceleration capacity of the manipulator, with evident benefits on the overall execution time. In particular, in [7], the authors come up with a straightforward formulation demonstrating that maximizing the acceleration/deceleration

capacity yields a penalization of the joints with a higher inertia-torque ratio. They observe that the overall trajectory tracking time is notably reduced, concluding that weighted pseudo-inverse control is favorable with respect to a simpler unweighted pseudo-inverse scheme.

The same idea is essentially exploited in [8], but with a second-order locally-optimal redundancy resolution scheme like

$$\mathbf{q}'' = \mathbf{J}_W^{\dagger}(\mathbf{x}'' - \mathbf{J}'\mathbf{q}') + (\mathbf{I} - \mathbf{J}_W^{\dagger}\mathbf{J})\boldsymbol{\gamma}'' \tag{4.15}$$

where $\boldsymbol{\gamma}'' = -k\frac{\partial G}{\partial \mathbf{q}} - \mathbf{D}\mathbf{q}'$ and $\mathbf{D}$ is a $n \times n$ positive semi-definite damping matrix, used to guarantee bounded displacements in the joint space. In this formulation, null-space projection and weighted pseudo-inverse are used together to achieve better performances. Like in [7] and [6], the matrix $\mathbf{W}$ and the performance index G are selected so as to maximize the acceleration/deceleration capabilities of the manipulator along the path. On the other hand, additional parameters are introduced, whose tuning is crucial to the improvement of the trajectory tracking time. In [8], the authors propose to select ranges for each parameter and search the best values on a discretized grid after a large number of simulations.

The formulation in (4.15) can be further enriched by the addition of a stabilizing PD term in order to avoid numerical drifts [8]:

$$\mathbf{q}'' = \mathbf{J}_W^{\dagger}(\mathbf{x}'' - \mathbf{J}'\mathbf{q}' + \mathbf{K}_d\mathbf{e}' + \mathbf{K}_p\mathbf{e}) + (\mathbf{I} - \mathbf{J}_W^{\dagger}\mathbf{J})\boldsymbol{\gamma}'' \tag{4.16}$$

where $\mathbf{K}_d > 0$ and $\mathbf{K}_p > 0$ are diagonal gain matrices, $\mathbf{e} = \mathbf{x} - \mathbf{k}(\mathbf{q})$ and $\mathbf{e}' = \mathbf{x}' - \mathbf{J}(\mathbf{q})\mathbf{q}'$.

Common to all the techniques of this section is the problem of determining the initial condition for the integration of the first-order or second-order differential equations defined on the curvilinear coordinate domain. Usually, these are decided by the current robot state before planning. Nevertheless, for repeated tasks, it might be of interest to start the motion from the best initial configuration, i.e. the one allowing for the shortest trajectory tracking time.

In [8], a time control scheme equivalent to (4.16) is proposed to choose $\mathbf{q}(0)$ and $\mathbf{q}'(0)$ so as to maximize the acceleration capabilities at rest, coherently with the underlying idea of the control scheme. Thus, starting from an arbitrary configuration, typically still satisfying the kinematic constraint $\mathbf{x} = \mathbf{k}(\mathbf{q})$, the joints move by mainly exploiting internal motions until reaching the best initial configuration.

4.1.3 Problem formulation in the extended phase space

It is still possible for redundant manipulators to conduct an analysis in the phase plane, as shown in Section 3.3.2 for non-redundant manipulators. However, intuitively, since r more parameters are free, one should expect that the phase "plane" actually is a more generic *phase space*, of a greater number of dimensions. In order to formulate the problem this way, it is convenient to adopt a redundancy parametrization based on joint selection or joint combination [106]. The former is also known as *Joint Space Decomposition* (JSD) [11]. Here, let us analyze the problem by assuming to parametrize redundancy with a subset of the joint position vector, termed $\mathbf{q}_j \in \mathbb{R}^r$, while the vector of the remaining joint positions is termed $\mathbf{q}_r \in \mathbb{R}^m$. We recall that $m+r = n$. The first and second λ-derivatives of the workspace path can thus be written as:

$$\begin{aligned} \mathbf{x}' &= \mathbf{J}_j(\mathbf{q})\mathbf{q}_j' + \mathbf{J}_r(\mathbf{q})\mathbf{q}_r' \\ \mathbf{x}'' &= \mathbf{J}_j(\mathbf{q})\mathbf{q}_j'' + \mathbf{J}_r(\mathbf{q})\mathbf{q}_r'' + \mathbf{J}_j'(\mathbf{q},\mathbf{q}')\mathbf{q}_j' + \mathbf{J}_r'(\mathbf{q},\mathbf{q}')\mathbf{q}_r' \end{aligned} \tag{4.17}$$

The Jacobian $\mathbf{J}_r \in \mathbb{R}^{m\times m}$, made up of the columns multiplying the non-redundant joints $\mathbf{q}_r$, is square and, far from singularities of the kinematic subchain, can be inverted. On the other hand, $\mathbf{J}_j \in \mathbb{R}^{m\times r}$ is made up of the columns multiplying the redundant joints $\mathbf{q}_j$. One can solve the equations above for $\mathbf{q}'$ and $\mathbf{q}''$:

$$\mathbf{q}_r' = \mathbf{J}_r^{-1}(\mathbf{q})\big(\mathbf{x}' - \mathbf{J}_j(\mathbf{q})\mathbf{q}_j'\big) \tag{4.18}$$

$$\mathbf{q}_r'' = \mathbf{J}_r^{-1}(\mathbf{q})\big(\mathbf{x}'' - \mathbf{J}_j(\mathbf{q})\mathbf{q}_j'' - \mathbf{J}_j'(\mathbf{q},\mathbf{q}')\mathbf{q}_j' - \mathbf{J}_r'(\mathbf{q},\mathbf{q}')\mathbf{q}_r'\big) \tag{4.19}$$

Inserting (4.18) and (4.19) in (3.1) and (3.2) and substituting (4.18) in (4.19) yields:

$$\begin{aligned}\dot{\mathbf{q}}_r =& \mathbf{J}_r^{-1}\big(\mathbf{x}' - \mathbf{J}_j\mathbf{q}_j'\big)\dot{\lambda} \\ \ddot{\mathbf{q}}_r =& \mathbf{J}_r^{-1}\big[\mathbf{x}'\ddot{\lambda} + \big(\mathbf{x}'' - \mathbf{J}_r'\mathbf{J}_r^{-1}\mathbf{x}'\big)\dot{\lambda}^2 - \mathbf{J}_j\big(\mathbf{q}_j'\ddot{\lambda} + \mathbf{q}_j''\dot{\lambda}^2\big) - \\ & - \big(\mathbf{J}_j' - \mathbf{J}_r'\mathbf{J}_r^{-1}\mathbf{J}_j\big)\mathbf{q}_j'\dot{\lambda}^2\big]\end{aligned} \tag{4.20}$$

where the dependency of the Jacobians and their derivatives on $\mathbf{q}$ and $\mathbf{q}'$ has been omitted for the sake of clarity.

If one replaced $\mathbf{q}_j'\dot{\lambda} = \dot{\mathbf{q}}_j$ and $\mathbf{q}_j'\ddot{\lambda} + \mathbf{q}_j''\dot{\lambda}^2 = \ddot{\mathbf{q}}_j$, the equations above would become:

$$\dot{\mathbf{q}}_r = \mathbf{J}_r^{-1}\big(\mathbf{x}'\dot{\lambda} - \mathbf{J}_j\dot{\mathbf{q}}_j\big) \tag{4.21}$$

$$\ddot{\mathbf{q}}_r = \mathbf{J}_r^{-1}\big[\mathbf{x}'\ddot{\lambda} + \big(\mathbf{x}'' - \mathbf{J}_r'\mathbf{J}_r^{-1}\mathbf{x}'\big)\dot{\lambda}^2 - \mathbf{J}_j\ddot{\mathbf{q}}_j - \big(\mathbf{J}_j' - \mathbf{J}_r'\mathbf{J}_r^{-1}\mathbf{J}_j\big)\dot{\mathbf{q}}_j\dot{\lambda}\big] \tag{4.22}$$

From the expression of $\dot{\mathbf{q}}_r$ and $\ddot{\mathbf{q}}_r$ above, it results that velocity and acceleration of the non-redundant joints are functions of the path parameter λ and its derivatives and of the redundant joint positions $\mathbf{q}_j$ and their derivatives, i.e.

$$\begin{aligned}\dot{\mathbf{q}}_r &= \dot{\mathbf{q}}_r(\lambda, \dot{\lambda}, \mathbf{q}_j, \dot{\mathbf{q}}_j) = \dot{\mathbf{q}}_r(\lambda, \dot{\lambda}, \mathbf{q}_j, \mathbf{q}_j') \\ \ddot{\mathbf{q}}_r &= \ddot{\mathbf{q}}_r(\lambda, \dot{\lambda}, \ddot{\lambda}, \mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) = \ddot{\mathbf{q}}_r(\lambda, \dot{\lambda}, \ddot{\lambda}, \mathbf{q}_j, \mathbf{q}_j', \mathbf{q}_j'')\end{aligned} \tag{4.23}$$

The same decomposition between redundancy parameters and remaining joints at acceleration level can be adopted for the dynamic model in (3.7), yielding:

$$\mathbf{H}_r(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{H}_j(\mathbf{q})\ddot{\mathbf{q}}_j + \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} \tag{4.24}$$

where $\mathbf{H}_j \in \mathbb{R}^{n\times r}$ is made up of the columns of $\mathbf{H}$ multiplying the redundant joints $\mathbf{q}_j$ and $\mathbf{H}_r \in \mathbb{R}^{n\times m}$ is made up of the remaining columns of $\mathbf{H}$. From this form of the dynamic model, using the expression of $\ddot{\mathbf{q}}_r$ in (4.22), we get [107]:

$$\mathbf{a}(\lambda, \mathbf{q}_j)\ddot{\lambda} + \mathbf{A}(\lambda, \mathbf{q}_j)\ddot{\mathbf{q}}_j + \boldsymbol{\eta}(\lambda, \dot{\lambda}, \mathbf{q}_j, \dot{\mathbf{q}}_j) = \boldsymbol{\tau} \tag{4.25}$$

where

$$\begin{aligned} \mathbf{a}(\lambda, \mathbf{q}_j) = &\ \mathbf{H}_r \mathbf{J}_r^{-1} \mathbf{x}' \\ \mathbf{A}(\lambda, \mathbf{q}_j) = &\ \mathbf{H}_j - \mathbf{H}_r \mathbf{J}_r^{-1} \mathbf{J}_j \\ \boldsymbol{\eta}(\lambda, \dot{\lambda}, \mathbf{q}_j, \dot{\mathbf{q}}_j) = &\ \mathbf{f} + \mathbf{H}_r \mathbf{J}_r^{-1} \big[\big(\mathbf{x}'' - \mathbf{J}_r' \mathbf{J}_r^{-1} \mathbf{x}' \big) \dot{\lambda}^2 - \\ &\ - \big(\mathbf{J}_j' - \mathbf{J}_r' \mathbf{J}_r^{-1} \mathbf{J}_j \big) \dot{\mathbf{q}}_j \dot{\lambda} \big] \end{aligned} \tag{4.26}$$

We may define the control input $\mathbf{u} = \left[\ddot{\lambda}, \ddot{\mathbf{q}}_j^T \right]^T \in \mathbb{R}^{1+r}$ and $\mathbf{M} = [\mathbf{a}, \mathbf{A}] \in \mathbb{R}^{n \times (1+r)}$ and rewrite (4.25) as:

$$\mathbf{M}(\lambda, \mathbf{q}_j)\mathbf{u} + \boldsymbol{\eta}(\lambda, \dot{\lambda}, \mathbf{q}_j, \dot{\mathbf{q}}_j) = \boldsymbol{\tau} \tag{4.27}$$

With this parametrization, the problem (4.11) can be reformulated in a simpler form as:

$$\begin{aligned} \min_{\mathbf{u}} & \int_0^{t_f} dt \\ \text{s.t.} \quad & \boldsymbol{\tau}_{min} \leq \mathbf{M}\mathbf{u} + \boldsymbol{\eta} \leq \boldsymbol{\tau}_{max} \\ \text{w/ b.c.} \quad & \dot{\lambda}(0) = 0, \ \dot{\lambda}(t_f) = 0 \\ & \dot{\mathbf{q}}_j(0) = \mathbf{0}, \ \dot{\mathbf{q}}_j(t_f) = \mathbf{0} \end{aligned} \tag{4.28}$$

The reader may notice the similarity to the non-redundant problem definition in (3.54).

Solving problem (4.28) is not an easy task, especially because of the impossibility to compute the input limits explicitly and dealing with an optimization in the multi-dimensional space. An attempt to solve an approximation of the problem above is given in [107]. Being the input constraint inequalities linear in $\mathbf{u}$, the authors propose to use a linear programming (LP) approach. For each possible value of the state $\mathbf{s} = \left[\lambda, \mathbf{q}_j^T, \dot{\lambda}, \dot{\mathbf{q}}_j^T \right]^T$, by which both $\mathbf{M}$ and $\boldsymbol{\eta}$ are determined, they solve an LP problem with objective function $I = \ddot{\lambda}$, that has to be minimized or maximized, depending on whether the phase-space trajectory is following an acceleration or deceleration profile.

Of course, in order to provide an initial state for $\lambda = 0$, they fix the initial joint positions for the redundant parameters, i.e. $\mathbf{q}_j(0) = \mathbf{q}_{j,0}$. Also, since the MVC-equivalent surface cannot be computed explicitly, many of the algorithms analyzed for non-redundant manipulators cannot be applied. The integration in the phase space thus resembles the try-and-error approach first proposed in [81] and recalled in Section 3.4.1. The proposed technique does not use any backward integration, whose starting condition would be impossible to define without a prior identification of the switching points and with the presence of the additional degrees of freedom $\mathbf{q}_j$. In addition, in order to make the algorithm more efficient, the number of switching points is possibly reduced with respect to the real time-optimal trajectory. Lastly, the proposed algorithm cannot enforce the boundary condition $\dot{\mathbf{q}}_j(t_f) = \mathbf{0}$. For all these reasons, even though (4.28) is formulated as a globally-optimal control problem, the technique of [107] can only generate sub-optimal trajectories.

4.1.4 Problem formulation for multiple shooting

All the techniques mentioned so far adopt some sort of parametrization of robot dynamics resembling that of Section 3.2.2, discussed for non-redundant manipulators. In other words, the dynamic equations establishing the connection between the applied torques and the kinematic quantities are manipulated to extract the path parameter and its derivatives, that are subject to optimization.

An alternative formulation consists in addressing the optimization problem directly, without any prior manipulation of the quantities in play. The time axis is divided in a fixed number of intervals N_i of the same variable length, whose sum is t_f, that is the result of the optimization process. When t_f is given, the duration of each time interval in the time domain is also determined. The problem is to find, for each of the intervals, the value of the path parameter

λ, the joint position vector $\mathbf{q}$ and their derivatives $\dot{\lambda}$ and $\dot{\mathbf{q}}$. More formally, said i the interval index, the objective is to find

$$\mathbf{s}_i = \begin{bmatrix} \mathbf{q}_i^T & \lambda_i & \dot{\mathbf{q}}_i^T & \dot{\lambda}_i \end{bmatrix}^T \tag{4.29}$$

so that t_f is minimum [9]. The reader may notice the similarity with (4.1), but here the state vector is discretized.

In order to impose the torque constraints, it suffices to discretize the dynamic model in (4.2), that is:

$$\boldsymbol{\tau}_{min} \leq \mathbf{H}(\mathbf{q}_i)\ddot{\mathbf{q}}_i + \mathbf{f}(\mathbf{q}_i, \dot{\mathbf{q}}_i) \leq \boldsymbol{\tau}_{max} \tag{4.30}$$

The imposition of the path constraint is rather non-trivial, as infinite joint position vectors exist that satisfy it, and they are not equivalent with respect to the optimization goal. In [9] and [11], two techniques are proposed to this respect, both considering at least the second-order derivative of the kinematic constraint: joint-space decomposition and null-space projection. In the former case, the constraint, suitably discretized, can be written as

$$\ddot{\mathbf{q}}_{r,i} = \mathbf{J}_r^{-1}\left(\ddot{\mathbf{x}}_i - \dot{\mathbf{J}}\dot{\mathbf{q}}_i - \mathbf{J}_j\ddot{\mathbf{q}}_{j,i}\right) \tag{4.31}$$

where the meaning of symbols is the same as Section 4.1.3. The input control vector is, in this case, $\mathbf{u}_i = \left[\ddot{\mathbf{q}}_{j,i}^T, \ddot{\lambda}_i\right]^T$. In the latter case, the constraint, again suitably discretized, can be written as

$$\ddot{\mathbf{q}}_i = \mathbf{J}_i^{\dagger}\left(\dot{\mathbf{x}}_i - \mathbf{J}\mathbf{q}_i\right) + \mathbf{N}\boldsymbol{\gamma}_i \tag{4.32}$$

where $\mathbf{N}$ is a null-space basis of the Jacobian which allows to span the infinite solutions to the IK problem, and $\boldsymbol{\gamma}_i$ is the vector of scaling factors at the i-th interval and can be used to impose specific boundary conditions, e.g. zero velocity along the self-motion manifold at the beginning and at the end of the trajectory. The input control vector is, in this case, $\mathbf{u}_i = \left[\boldsymbol{\gamma}_i^T, \ddot{\lambda}_i\right]^T$. For both techniques, since the path constraint is enforced through the integration of (4.31) or (4.32), a closed-loop inverse kinematics scheme [108] needs to be used, as in [10].

Finally, in order to guarantee the coherency between $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$, the state equation needs to be imposed, that is

$$\mathbf{s}_{i+1} = \mathbf{h}(\mathbf{s}_i, \mathbf{u}_i) \tag{4.33}$$

Since the problem is addressed directly, the optimization vector $\mathbf{w}$ includes all the free variables, that is:

$$\mathbf{w} = \begin{bmatrix} \mathbf{s}_0^T & \mathbf{u}_0^T & \dots & \mathbf{s}_{N_i}^T & \mathbf{u}_{N_i}^T & t_f \end{bmatrix}^T \tag{4.34}$$

Summarizing all the above, the problem can be formulated as follows

$$\begin{aligned}
\min_{\mathbf{w}} \quad & t_f \\
\text{s.t.} \quad & \boldsymbol{\tau}_{min} \leq \mathbf{H}(\mathbf{q}_i)\ddot{\mathbf{q}}_i + \mathbf{f}(\mathbf{q}_i, \dot{\mathbf{q}}_i) \leq \boldsymbol{\tau}_{max} \\
& \mathbf{s}_{i+1} = \mathbf{h}(\mathbf{s}_i, \mathbf{u}_i) \\
& 0 \leq \lambda_i \leq \Lambda \\
& \dot{\lambda}_i \geq 0 \\
& \ddot{\mathbf{q}}_{r,i} = \mathbf{J}_r^{-1}\left(\ddot{\mathbf{x}}_i - \dot{\mathbf{J}}\dot{\mathbf{q}}_i - \mathbf{J}_j\ddot{\mathbf{q}}_{j,i}\right) \\
\text{or} \quad & \ddot{\mathbf{q}}_i = \mathbf{J}_i^{\dagger}\left(\dot{\mathbf{x}}_i - \mathbf{J}\mathbf{q}_i\right) + \mathbf{N}\boldsymbol{\gamma}_i \\
\text{w/ b.c.} \quad & \lambda_0 = 0,\ \lambda_{N_i} = \Lambda \\
& \dot{\lambda}_0 = 0,\ \dot{\lambda}_{N_i} = 0 \\
& \mathbf{q}_0 = \bar{\mathbf{q}}_0,\ \mathbf{q}_{N_i} = \bar{\mathbf{q}}_{N_i} \\
& \dot{\mathbf{q}}_0 = \mathbf{0},\ \dot{\mathbf{q}}_{N_i} = \mathbf{0}
\end{aligned} \tag{4.35}$$

where $\bar{\mathbf{q}}_0$ and $\bar{\mathbf{q}}_{N_i}$ are assigned joint positions at the beginning and at the end of the trajectory.

Together with the above, the formulation is flexible enough to include constraints on the joint positions and their derivatives, as well as other application-specific constraints. For instance, in [11], constraints are considered on joint positions, velocities, accelerations, jerks and jounces (i.e. the fourth-order derivative of the position), through which the authors demonstrate that it is possible to obtain continuous and quasi-differentiable motor torques.

It is clear that, in this case, the kinematic path constraints need to be derived further to obtain a closed-form equation for the control inputs. This result is of utmost importance since controlling the torques at their bounds, as discussed above, rules out close-loop motor control because of insufficient margins. On the other hand, the system is subject to unacceptable vibrations. In conclusion, as commented in [11], the insufficient continuity of globally-optimal minimum time trajectories is the main shortcoming when it comes to the control of real robots.

The problem (4.35) can be solved with direct multiple shooting using the interior-point method, after an initial guess on the variables t_f, λ_i, $\mathbf{q}_i$ (or $\boldsymbol{\gamma}_i$) has been made. Since (4.35) is non-convex, globally-optimal solutions cannot be guaranteed and, in general, the quality of the resulting solution strictly depends on the initial guess [10].

4.2 Time-optimal control of redundant robots with dynamic programming

Both in Section 2.7 and Section 3.7, we have seen that dynamic programming can be beneficial to overcome the limits of calculus of variations, first of all because of its flexibility that allows to tackle complex real-world scenarios. Since it is a successful methodology for both inverse kinematics of redundant robots and time-optimal planning of non-redundant ones, its application to time-optimal control of redundant robots comes naturally.

At the same time, we also learned that dynamic programming is very sensitive to system complexity and a problem that can be solved in a few seconds or minutes might become unsolvable in any reasonable amount of time, just with a little increase of complexity, even for off-line applications. For this reason, all the considerations and analyses of the previous sections will need to be thoroughly put in practice to achieve time-optimal planning

and control of redundant robots.

4.2.1 Problem formulation with dynamic programming

For both problems addressed in the previous sections, we arranged the dynamic programming formulation as a graph search problem, where the graph is conveniently represented as a grid or multi-grid. Intuitively, the same can be done for time-optimal planning of a redundant robot by suitably combining the state information coming from one domain and the other. Thus, we expect to be dealing with a problem of an increased dimension, and cope with complexity by using all the information and constraints that come from a real-world scenario.

Let us consider an m-dimensional task $\mathbf{x}(\lambda)$ such that a given manipulator with n degrees of freedom is redundant, i.e. $n > m$. Let us also suppose that the given task only constrains the end-effector path, but not the time law along it. In a discrete-time formulation, both the sequence of joint-space configurations and the sequence of timestamps along the path are unknown and they fully represent the state of the system along the path:

$$\mathbf{s} = \begin{bmatrix} \mathbf{q} & \dot{\lambda} \end{bmatrix}^T \tag{4.36}$$

The system evolves in agreement with a certain dynamic model, that can be generally written, in discrete time, as

$$\mathbf{s}(i+1) = f\big(\mathbf{s}(i), \mathbf{u}(i)\big) \tag{4.37}$$

The control input $\mathbf{u}$ in the equation above usually equates to the actuator torques, that, in discrete time, can be expressed as:

$$\boldsymbol{\tau}(i) = \mathbf{H}\big(\mathbf{q}(i)\big)\ddot{\mathbf{q}}(i) + \mathbf{f}\big(\mathbf{q}(i), \dot{\mathbf{q}}(i)\big) \tag{4.38}$$

where, by omitting the dependence on i for simplicity,

$$\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{q}}^T \mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{d}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \tag{4.39}$$

and $\mathbf{d}(\dot{\mathbf{q}})$ is the $n \times 1$ vector of Coulomb and viscous friction terms.

Let $S_{(\cdot)}(i)$ be the discrete-time function (or sequence of samples) of the continuous-time function $(\cdot)(\lambda)$ up until the i-th sample, obtained through a discretization of λ such that

$$\lambda(i) = i\Delta_\lambda \quad \text{with } i = 0, 1, 2, .., N_i \quad \text{and } N_i = \frac{\Lambda}{\Delta_\lambda} \tag{4.40}$$

For instance, $S_{\dot{\lambda}}(i) = \{\dot{\lambda}(0), ..., \dot{\lambda}(i)\}$ is the discrete-time phase-space trajectory up until stage i.

The joint velocities and accelerations at a given stage i, $\dot{\mathbf{q}}(i)$ and $\ddot{\mathbf{q}}(i)$ respectively, in a discrete-time formulation, are functions, in general, of the sequences $S_{\mathbf{q}}(i)$, $S_{\dot{\mathbf{q}}}(i)$ and $S_t(i)$, i.e.

$$\dot{\mathbf{q}}(i) = f_{\dot{\mathbf{q}}}\big(S_{\mathbf{q}}(i), S_t(i)\big) \tag{4.41}$$

$$\ddot{\mathbf{q}}(i) = f_{\ddot{\mathbf{q}}}\big(S_{\dot{\mathbf{q}}}(i), S_t(i)\big) \tag{4.42}$$

The specific form of the functions above depends on the discrete approximation used, ranging from simple Euler approximations that only involve stages i and $i-1$, to more complex approximations that, at the limit, can involve all the stages up until the initial one.

Now, recalling Section 2.7 and adopting a joint selection (or joint space decomposition) scheme [106, 13], each of the joint positions in $S_{\mathbf{q}}(i)$ can be computed from inverse kinematics, like

$$\mathbf{q}(i) = \mathbf{k}^{-1}\big(\mathbf{x}(i), \mathbf{v}(i), g(i)\big) \tag{4.43}$$

where $\mathbf{v} \in \mathbb{R}^r$ is the vector of redundant joints (or redundancy parameters) and g is an indicator of the specific extended aspect in which the manipulator lies at stage i. This way defined, $\mathbf{q}(i)$ is unique. Since $\mathbf{x}(\lambda)$ is a bijective function, we may rewrite the equation above as

$$\mathbf{q}(i) = \mathbf{k}^{-1}\big(\lambda(i), \mathbf{v}(i), g(i)\big) \tag{4.44}$$

In the same way, as done in Section 3.7, each timestamp in $S_t(i)$ can be derived from the phase-space trajectory up until stage i,

through some discrete approximation, e.g. (3.131), that is

$$t(i) = f_t\big(S_\lambda(i), S_{\dot\lambda}(i)\big) \tag{4.45}$$

Let us remark, once again, that $S_{\dot\lambda}(i)$ is unknown, and, as a consequence, $t(i)$ is unknown too.

Now, we may fold equations (4.44) and (4.45), for each i, into (4.41) and (4.42), yielding

$$\dot{\mathbf{q}}(i) = f_{\dot{\mathbf{q}}}\big(S_\lambda(i), S_{\dot\lambda}(i), S_{\mathbf{v}}(i), S_g(i)\big) \tag{4.46}$$
$$\ddot{\mathbf{q}}(i) = f_{\ddot{\mathbf{q}}}\big(S_\lambda(i), S_{\dot\lambda}(i), S_{\mathbf{v}}(i), S_g(i)\big) \tag{4.47}$$

By substituting both equations above into (4.38), we find that, in general, the input is a function of the four identified sequences, i.e.

$$\boldsymbol{\tau}(i) = f_{\boldsymbol{\tau}}\big(S_\lambda(i), S_{\dot\lambda}(i), S_{\mathbf{v}}(i), S_g(i)\big) \tag{4.48}$$

This means that, given the current state $\mathbf{s}(i)$, selecting the input $\boldsymbol{\tau}(i)$ that drives the discrete-time system in (4.37) to $\mathbf{s}(i+1)$ is equivalent to select the parameters $\dot\lambda$, $\mathbf{v}$ and g at the next stage, i.e. the input is a subset of the state variables. This is coherent with the logic of discretizing the state space instead of the input space that we adopted in Section 2.7 and Section 3.7, that is also a way to control the curse of dimensionality. It is also worth noting that the parameters $\dot\lambda$, $\mathbf{v}$ and g also constitute a minimum representation of the state for each stage, as opposite to (4.1) and (4.29), that, rather, are redundant representations. In a formulation based on calculus of variations, this result is somehow related to the minimum number of differential equations required to fully represent the system at hand, discussed in Section 2.5.2 and in [106].

Thus, let us proceed discretizing the variables $\dot\lambda$ and $\mathbf{v}$. The former is

$$\dot\lambda_l = l\Delta_{\dot\lambda} \quad \text{with } l = 0, 1, 2, .., N_l \quad \text{and } N_l = \frac{\dot\lambda_M}{\Delta_{\dot\lambda}} \tag{4.49}$$

where $\dot{\lambda}_M$ is the maximum pseudo-velocity value that the phase-space trajectory can reach. The latter is

$$\mathbf{v} = \mathbf{j} \circ \mathbf{\Delta}_v + \mathbf{v}_{min} \tag{4.50}$$

where '$\circ$' denotes the Hadamard product, $\mathbf{\Delta}_v = [\Delta_{v,1}, ..., \Delta_{v,r}]$ is the vector of the sampling intervals and $\mathbf{j} \in \mathbb{N}^r$ the vector of indices for each of the redundancy parameters, and $\mathbf{v}_{min}$ is the vector of lower bounds of the redundancy parameters domains. The elements of $\mathbf{j}$ take the maximum values $N_{j,1}, ..., N_{j,r}$ so that $[N_{j,1}, ..., N_{j,r}] \circ \mathbf{\Delta}_v + \mathbf{v}_{min}$ equals the upper bound of the redundant joint domains.

Let us recall equations (3.11) and (3.12), where the dynamic parameters $\mathbf{a}(\lambda)$ and $\mathbf{b}(\lambda)$ are expressed as functions of the geometric path. In the case considered here, the manipulator's Jacobian $\mathbf{J}$ is no longer invertible and both the dynamic parameters may take infinite values, depending on where, in the null-space of the Jacobian, the joints move. As a consequence, unlike time-optimal planning of non-redundant robots, the maximum and minimum pseudo-accelerations, as defined in (3.20), cannot be precomputed. As done for other problem formulations, we can still create a grid on the basis of the discrete values of the state, but the content of each single node should be reconsidered in light of this observation. Each node in the grid will then contain a state given by

$$\mathbf{s}_{l\mathbf{j}g}(i) = [\dot{\lambda}_l(i), \mathbf{q}_{\mathbf{j}g}(i)] \tag{4.51}$$

where $\mathbf{q}_{\mathbf{j}g}$ is the vector of joint positions obtained from equation (4.44) using the redundancy parameters $\mathbf{v}_{\mathbf{j}}$ and selecting the extended aspect g.

Since all the states are available in the grid, the dynamic programming problem is again a graph search problem, where the objective function can be generally defined as:

$$I(N_i) = \psi\big(\mathbf{s}_0\big) + \sum_{k=1}^{N_i} \phi\big(\mathbf{s}(k-1), \mathbf{s}(k)\big) \tag{4.52}$$

with the same meaning of terms as (3.130). More specifically, since the objective function to minimize is the time, there is no cost associated to the initial state, i.e. $\psi(\mathbf{s}_0) = 0$ and $I = t$. Assuming a backward Euler approximation,

$$t(N_i) = \sum_{k=1}^{N_i} \frac{\lambda(k) - \lambda(k-1)}{\dot{\lambda}(k)} \tag{4.53}$$

or, in a recursive form,

$$\begin{aligned} t(0) &= 0 \\ t(i) &= t(i-1) + \frac{\lambda(i) - \lambda(i-1)}{\dot{\lambda}(i)} \end{aligned} \tag{4.54}$$

For each $\dot{\lambda}(i) = 0$, a different approximation should be used, like the one proposed in [71].

As far as constraints are concerned, here we tackle the case of real robots, so that all the possible constraints that may arise from the control of a physical system are considered to reduce the search space and eventually lead to a solution in a relatively short time. In particular, let us include the following limits:

- minimum/maximum joint positions
- maximum joint velocities
- maximum joint accelerations
- maximum joint jerks
- maximum joint torques
- maximum joint torque rates

In addition, optional constraints can be taken into account, like:

- obstacles in the workspace

- maximum Cartesian velocity
- maximum Cartesian acceleration
- initial/final joint configurations
- cyclicity (for closed workspace paths)
- power limit
- maximum/minimum forces exchanged with the environment in an interaction scenario

We discussed in the previous sections about the flexibility of dynamic programming algorithms with respect to the inclusion of arbitrary constraints and indeed the list above is certainly non-exhaustive. Many other constraints can be designed depending on the specific application at hand.

As opposite to time-optimal planning of non-redundant robots, in this case, it is not convenient to perform any parametrization of the dynamics and hence, of the constraints. In facts, since the parametric vectors $\mathbf{a}(\lambda)$ and $\mathbf{b}(\lambda)$ cannot be pre-computed, they would need to be computed in-line, which does not provide any advantage with respect to verifying the constraints directly on the joint variables. Therefore, let us define the stage-dependent set $\mathcal{A}_i$, containing all nodes returning joint positions that respect the joint domains, the path constraint and, possibly, imposed initial/final configurations. Such constraints are formalized as

$$\mathbf{s}_{ljg}(i) \in \mathcal{A}_i \tag{4.55}$$

Joint velocities, accelerations and jerk limits can be directly encoded in equivalent stage-dependent, as well as state-dependent sets $\mathcal{B}_i^1$ $\mathcal{B}_i^2$ and $\mathcal{B}_i^3$, so that:

$$\dot{\mathbf{q}}(i) \in \mathcal{B}_i^1\big(\mathbf{s}(i)\big) \tag{4.56}$$

$$\ddot{\mathbf{q}}(i) \in \mathcal{B}_i^2\big(\mathbf{s}(i)\big) \tag{4.57}$$

$$\dddot{\mathbf{q}}(i) \in \mathcal{B}_i^3\big(\mathbf{s}(i)\big) \tag{4.58}$$

Equivalently, joint torque and torque rate limits can be encoded in similar sets $\mathcal{C}_i^1$ and $\mathcal{C}_i^2$:

$$\boldsymbol{\tau}(i) \in \mathcal{C}_i^1\big(\mathbf{s}(i)\big) \tag{4.59}$$

$$\dot{\boldsymbol{\tau}}(i) \in \mathcal{C}_i^2\big(\mathbf{s}(i)\big) \tag{4.60}$$

More commonly, all the quantities above are given within fixed domains that do not change along the trajectory, are not configuration dependent and not velocity-dependent, but the generality of the framework allows for the accommodation of more complex constraints. For example, in Section 3.7.1, we argued that it is not possible to handle interacting torque constraints in the form of (3.136) with a complete parametrization of the problem. To this respect, the formulation that we introduce here for redundant robots is more generic and flexible and also encompasses (3.136).

All the sets above can be combined together to define the set $\mathcal{D}_i$ of reachable states for a generic stage i, that is

$$\begin{aligned}\mathcal{D}_i = \mathcal{A}_i \cap \Bigg\{ \mathbf{s}(i) : \dot{\mathbf{q}}(i) \in \mathcal{B}_i^1, \ddot{\mathbf{q}}(i) \in \mathcal{B}_i^2, \dddot{\mathbf{q}}(i) \in \mathcal{B}_i^3, \\ \boldsymbol{\tau}(i) \in \mathcal{C}_i^1, \dot{\boldsymbol{\tau}}(i) \in \mathcal{C}_i^2 \\ \text{with } \mathbf{s}(i-1) \in \mathcal{A}_{i-1}, ..., \mathbf{s}(0) \in \mathcal{A}_0 \Bigg\}\end{aligned} \tag{4.61}$$

The set $\mathcal{D}_i$ can be used to complete our dynamic programming formulation, by minimizing over the admissible states, such that equations (4.54) become:

$$\begin{aligned} t_{opt}(0) &= 0 \\ t_{opt}(i) &= \min_{\mathbf{s}(i) \in \mathcal{D}_i} \left[t_{opt}(i-1) + \frac{\lambda(i) - \lambda(i-1)}{\dot{\lambda}(i)} \right] \end{aligned} \tag{4.62}$$

where $t_{opt}(i)$ at a generic stage i is the *optimal return function* and $t_{opt}(N_i)$ represents the optimal cost.

4.2.2 Algorithmic implementation

Based on the problem formulation above, we can define the time-optimal trajectory planning algorithm for redundant robots (TOTP-R) with dynamic programming as in Algorithm 8. A pictorial view is provided in Figure 4.1.

Algorithm 8 Time-optimal trajectory planning for redundant robots with dynamic programming

1: *Initialize state space grid through inverse kinematics and discretization of* $\dot{\lambda}$*, according to equations* (4.44) *and* (4.51)
2: *Initialize* $\mathcal{A}_i, \forall i = 0..(N_i - 1)$
3: *Initialize* $\mathcal{B}_i^1, \mathcal{B}_i^2, \mathcal{B}_i^3, \mathcal{C}_i^1, \mathcal{C}_i^2 \ \forall i = 0..(N_i - 1)$ *with state-independent information*
4: *Initialize* $\mathcal{D}_i = \varnothing, \forall i = 0..N_i$
5: *Initialize cost map* $t_{i,l,\mathbf{j},g} = +\infty \ \forall i, l, \mathbf{j}, g$
6: $t_{0,l,\mathbf{j},g} \leftarrow 0 \ \forall l, \mathbf{j}, g$
7: $\mathcal{D}_0 \leftarrow \mathcal{A}_0$
8: **for** $i \leftarrow 0$ to $N_i - 1$ **do**
9: **for each** $\mathbf{s}_{l\mathbf{j}g} \in \mathcal{D}_i$ **do**
10: **for each** $\mathbf{s}_{m\mathbf{k}h} \in \mathcal{A}_{i+1}$ **do**
11: *Compute* $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, $\dddot{\mathbf{q}}$, $\boldsymbol{\tau}$, $\dot{\boldsymbol{\tau}}$
12: **if** Constraints (4.56) - (4.60) are satisfied **then**
13: $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_{i+1} \cup \{\mathbf{s}_{m\mathbf{k}h}\}$
14: *Compute instantaneous cost function* ϕ
15: **if** $t_{i,l,\mathbf{j},g} + \phi < t_{i+1,m,\mathbf{k},h}$ **then**
16: $t_{i+1,m,\mathbf{k},h} = t_{i,l,\mathbf{j},g} + \phi$
17: *Let* $\mathbf{s}_{l\mathbf{j}g}$ *at stage* i *be the predecessor of* $\mathbf{s}_{m\mathbf{k}h}$ *at stage* $i+1$
18: $t_{opt}(N_i) = \min_{\mathbf{j},g} \left[t_{N_i,0,\mathbf{j},g}\right]$
19: *Build functions* $\dot{\lambda}(i)$ *and* $\mathbf{q}(i)$ *of optimal pseudo-velocities and joint positions by screening the predecessors map backward*

The algorithm assumes that a workspace path is given, as a discrete set of points and that they are associated to discrete values of the curvilinear coordinate, from $\lambda = 0$ up to the length of the path $\lambda = \Lambda$. At step 1, the redundancy parameters vector is discretized according to (4.50), which allows to compute inverse kinematic (IK) solutions as in (4.44). The pseudo-velocity is also discretized according to (4.49), so that the grid nodes can be defined as in (4.51).

Usually, with reference to (4.44), analytic IK solvers are such that all the IK solutions for all extended aspects are obtained at once with one call to the solver, but, if this is not the case, or an analytic solver is not available, the parameter g should be specified.

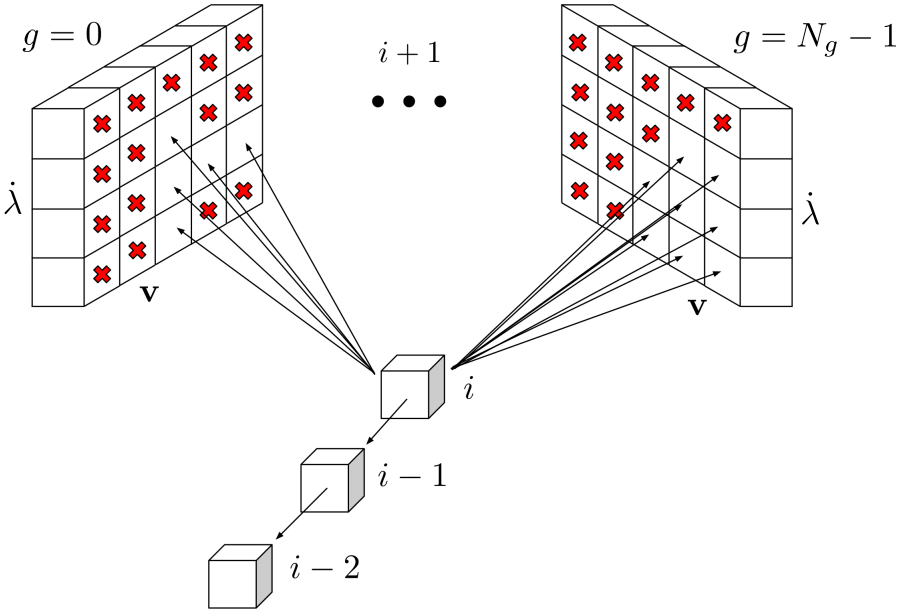


Figure 4.1: Pictorial view of the dynamic programming algorithm showing the possible transitions between a generic node at stage i and all the admissible nodes at stage $i + 1$; red crosses represent nodes that are not enabled at the next stage, meaning that they cannot be reached by any node at the current stage

In practice, this means solving a constrained inverse kinematic problem since g should represent a condition by which one and only one IK solution is available for a given combination of redundancy parameters and workspace pose. As already discussed in Chapter 2 and in [13], finding such a condition is not trivial. If it cannot be found, g no longer represents the extended aspect, but should be interpreted as a simple *grid index* enumerating the IK solutions. The state space grids obtained in this case are termed *non-homogeneous* [13].

At step 2, the nodes in the grid are enabled/disabled according to geometrical constraints, including for example, configurations that bring the robot to collide with the surrounding environment. Also, if IK solutions are homogeneous, entire grids can be excluded to

make the DP algorithm find a trajectory in specific extended aspects. The sets $\mathcal{A}_i$ can also be used to impose that the robot must start and finish its motion at rest, or even stop along the path if this is required by the specific task. In time-optimal planning of non-redundant robots, the same sets $\mathcal{A}_i$ were used to impose the MVC constraint. In this case, since we do not perform any parametrization of dynamics and constraints, this is not possible, and the MVC constraint is implicitly checked at the time of verifying (4.56)-(4.60) at step 12.

In most practical cases, constraints on joint velocities, accelerations, jerks, torques and torque rates are given in terms of state-independent connected sets, e.g. $\dot{\mathbf{q}} \in [\dot{\mathbf{q}}_{min}, \dot{\mathbf{q}}_{max}]$. In this case, the sets in (4.56)-(4.60) are completely defined beforehand, as done at step 2. Conversely, if they were state-dependent, they could be re-computed at the time the constraints are checked, where the current velocity and acceleration of the system are known.

At a given stage i, for each pair of nodes (steps 9 and 10) the discrete-time functions (or sequences) of parameters $S_\lambda(i)$, $S_{\dot\lambda}(i)$, $S_{\mathbf{v}}(i)$, $S_g(i)$ are available through back-pointers to compute joint velocities, accelerations and torques as in (4.46)-(4.48) respectively, as well as joint jerks and torque rates with equivalent discrete approximations. If constraints are satisfied, we say that the node $\mathbf{s}_{m\mathbf{k}h}(i+1)$ can be reached by the node $\mathbf{s}_{l\mathbf{j}g}(i)$. The former is then added to the set of nodes $\mathcal{D}_{i+1}$ that can be visited at the next stage (step 13). Among all the nodes at stage i that can reach $\mathbf{s}_{m\mathbf{k}h}(i+1)$, at step 17, we save the pointer to the one that provides the lowest cumulative cost (or optimal return function), according to (4.62).

After the process above has been repeated for all the workspace points, the one with minimum cumulative traversing time is picked (step 18) and, from it, the information about the time law, i.e. $\dot\lambda(i)$, and exploitation of the null-space, i.e. $\dot{\mathbf{q}}(i)$, are retrieved by following the map of predecessors backwards (step 19). The timestamps can be computed through (4.45) and applied to the joint space path to obtain the globally-optimal joint space trajectory. Like-

wise, globally-optimal joint velocities, accelerations and torques can be computed from (4.46)-(4.48).

4.3 Application to a 7-DOF robotic arm

The dynamic programming algorithm presented in Section 4.2 is validated on a spatial 7-DOF manipulator, that has been already used in Chapter 2 for redundancy resolution along a pre-scribed workspace trajectory. Here the objective is to address the specific issues that arise at control level and that are connected with the employment of a dynamic programming algorithm and time-optimal planning.

In particular, from Chapter 2 and from several other works, e.g. [14], we know that the execution of a DP-planned trajectory on a real system yields additional challenges, like that of the control references smoothness, mainly in terms of continuity of accelerations and torques. On the other hand, from Section 3.8, we know that several other issues arise when tracking time-optimal trajectories and several solutions exist to cope with them. The control scheme that guarantees the best performances must be necessarily evaluated on a case-by-case basis, as robots have different characteristics and the knowledge of their dynamic model can be more or less accurate. The parameters that allow us to assess the best solution mainly are:

- the possibility of replacing or modifying the robot's primary controller, provided by the manufacturer;
- the availability of an accurate dynamic model;
- several other factors including the robustness of the controller-robot system, the availability of pre-tuned controllers, the flexibility of the control scheme with respect to changes in the architecture, etc.

After addressing the issues above for the specific architecture at hand, the objective is to demonstrate that the developed dynamic programming algorithm can be effectively employed for time-optimal planning and control of redundant robots along prescribed paths. It is clear that, at control level, the resulting tracking accuracy cannot be properly judged if the task is not contextualized within a specific application. Nevertheless, the control architecture can be modified later on to cope with specific requirements, while the optimization of the controller itself for a specific task goes beyond the scope of this dissertation.

4.3.1 Franka Emika's Panda

The Panda robot by Franka Emika [52] is a 7-DOF manipulator conceived for both research and industrial applications with or without the presence of human operators. Some of its kinematic characteristics and Denavit-Hartenberg parameters have been already introduced in Section 2.8. In this section, we want to provide some more information about the robot, especially concerning its dynamic parameters, limits and hardware/software architecture.

4.3.1.1 Joint limits and other constraints

The Panda robot is characterized by the joint limits reported in Table 4.1. In Section 2.8.2, we already discussed about the effect of such limits on the null space observing that it gets significantly reduced for the assigned trajectory and the selected redundancy parameter $\mathbf{v} = q_4$.

Franka Emika also provide limits in the Cartesian space in terms of translation and rotational velocity, acceleration and jerk of the end-effector, but they are not considered here since the experiments always showed that they are implicitly respected when joint limits are respected. Also, our tasks do not enforce specific requirements for the end-effector motion, except for the assigned geometrical path. However, the dynamic programming algorithm

Joint	q_{min} [rad]	q_{max} [rad]	$\dot{q}_{max}$ [rad/s]	$\ddot{q}_{max}$ [rad/s^2]	$\dddot{q}_{max}$ [rad/s^3]	τ_{max} [Nm]	$\dot{\tau}_{max}$ [Nm/s]
1	−2.8973	2.8973	2.1750	15	7500	87	1000
2	−1.7628	1.7628	2.1750	7.5	3750	87	1000
3	−2.8973	2.8973	2.1750	10	5000	87	1000
4	−3.0718	−0.0698	2.1750	12.5	6250	87	1000
5	−2.8973	2.8973	2.6100	15	7500	12	1000
6	−0.0175	3.7525	2.6100	20	10000	12	1000
7	−2.8973	2.8973	2.6100	20	10000	12	1000

Table 4.1: Panda joint limits [54]

that we discussed in Section 4.2 does not introduce any restriction to this respect and Cartesian space constraints can be easily accommodated in the framework.

4.3.1.2 Dynamic model

The dynamic parameters that make up the model of the Panda are not available from the official Franka Emika channels. The robot is provided with a library, called *libfranka*, that allows to retrieve dynamic parameters on demand, by providing specific joint positions and velocities. *libfranka* also supports off-line requests, where dynamic parameters can be obtained for joint positions and velocities other than the current ones.

By default, the Panda robot performs automatic implicit compensation of gravity and joint friction, meaning that, when the robot is commanded in torques, only the net efforts τ_{net} have to be provided to the controller, i.e.

$$\boldsymbol{\tau}_{net} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \tag{4.63}$$

The dynamic parameters that can be retrieved from the *libfranka* interface are

- inertia matrix $\mathbf{H}(\mathbf{q})$;

- Coriolis vector $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$;
- gravity vector $\mathbf{g}(\mathbf{q})$.

Unfortunately, the usage of the *libfranka* library requires the connection with the *Franka Control Interface* be established, which is not always desirable in an off-line planning scenario. Also, the experiments showed that calls to *libfranka* are particularly slow, which discourages a systematic usage of the interface and practically rules out its employment in our dynamic programming algorithm where the dynamic model has to be computed billions of times. As far as friction torques are concerned, as highlighted in [109], they are not negligible, but, at the same time, they are not available through *libfranka*. For all these reasons, it is certainly convenient to work with the identified model of [109] and related libraries: they provide a much faster implementation and allow to retrieve friction torques directly. It is worth remarking that the friction model used in [109] is not a simple viscous friction model, but it also includes Coulomb terms and adopts a sigmoidal formulation to avoid discontinuities for low joint velocities.

4.3.1.3 Hardware and software architecture

The high-level hardware and software architecture of the Panda robot is summarized in Figure 4.2. The *Franka Control Interface* (FCI) allows to command the robot through a Linux workstation. It enables a UDP packet-based communication over Ethernet, preferably on a point-to-point connection to avoid network delays. On the workstation side, real-time capabilities are required to respect the controller frequency of 1 kHz. For this reason, the operating system is required to be patched with `PREEMPT_RT` [110].

The FCI provides two real-time interfaces, at different levels. If the user wants to control the robot with high level commands such as joint or Cartesian positions and velocities, the *motion generators* interface is used. Rather, if the user wants to control with

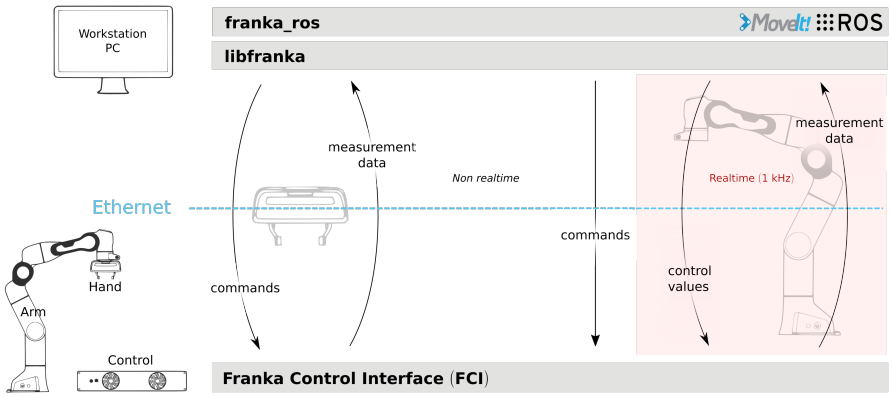


Figure 4.2: High-level hardware and software architecture of Franka Emika's Panda robot [54]

torques, commands are sent through the *external controller* interface. In the first case, the commands provided through the interface are completed (by computing forward or inverse kinematics) and provided to some internal controller to generate torques. In the second case, the provided net torques are summed to gravity and friction components and forwarded to the actuators. Internal controllers can generate torques to follow joint references (*joint impedence controller*) or Cartesian references (*Cartesian impedence controller*). A summarizing scheme is provided in Figure 4.3.

On the workstation side, on top of the operating system, the Panda communication protocol is implemented by *libfranka*, that is the proprietary library by Franka Emika. It allows user programs to access the kinematic and dynamic model, send commands and receive measurement data. Through *libfranka*, both the motion generators interface and the external controller interface can be used. It also implements some signal processing functions, that are a *low pass filter* and a *rate limiter*. The former is needed to smooth the user-commanded signal to allow for a more stable motion, while the latter ensures that the interface limits are not exceeded in terms of rate of change of the signals sent by the user (acceleration and jerks when the motion generators interface is

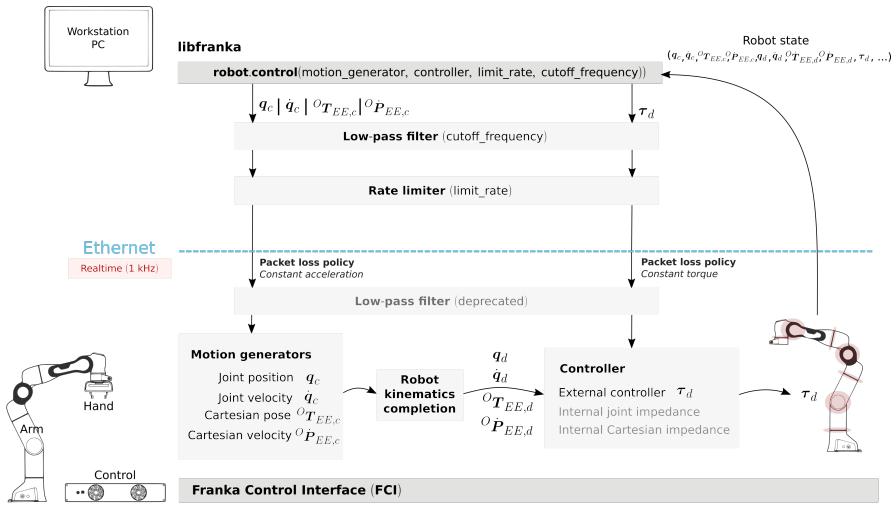


Figure 4.3: Panda's data flow in real-time communications [54]

used, torque rates when the external controller is used).

With reference to Figure 4.2, the Panda is also provided with a ROS package called `franka_ros`, that implements an interface for `ros_control` [111]. This way, all the ROS and MoveIt! technologies can be used to control the Panda robot through the capabilities offered by *libfranka*.

4.3.2 Task definition

Two paths are designed in the workspace to validate the algorithm presented in Section 4.2. They are shown in Figure 4.4. The path on the left is a straight line path having $x = 0.5$ m, $z = 0.4$ m and y spanning for 0.5 m from $y_s = 0.25$ m to $y_e = -0.25$ m. In terms of orientation the end-effector is aligned with the base reference for roll and yaw, while the pitch is $\theta = \pi$ rad along the whole path. The path on the right is an ellipse-like line with $\Lambda = 1.45$, generated with spline interpolation between the following control

points:

$$\begin{aligned}
\mathbf{x}_A &= \begin{bmatrix} 0.5 & 0 & 0.8 & 0 & \pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_B &= \mathbf{x}_A + \begin{bmatrix} 0 & -0.3 & -0.2 & 0 & \pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_C &= \mathbf{x}_A + \begin{bmatrix} 0 & 0 & -0.3 & 0 & \pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_D &= \mathbf{x}_A + \begin{bmatrix} 0 & 0.3 & -0.2 & 0 & \pi/2 & 0 \end{bmatrix}^T \\
\mathbf{x}_E &= \mathbf{x}_A
\end{aligned} \tag{4.64}$$

Both tasks constrain six dimensions, leaving one degree of freedom for redundancy resolution. In addition, the robot has to track the path as fast as possible. With respect to time-optimal planning of non-redundant mechanisms, in this case, the objective is to exploit the robot's kinematic redundancy to contribute maximizing the assigned performance index in a kineto-dynamic planning scenario, where the time law has to be defined too.

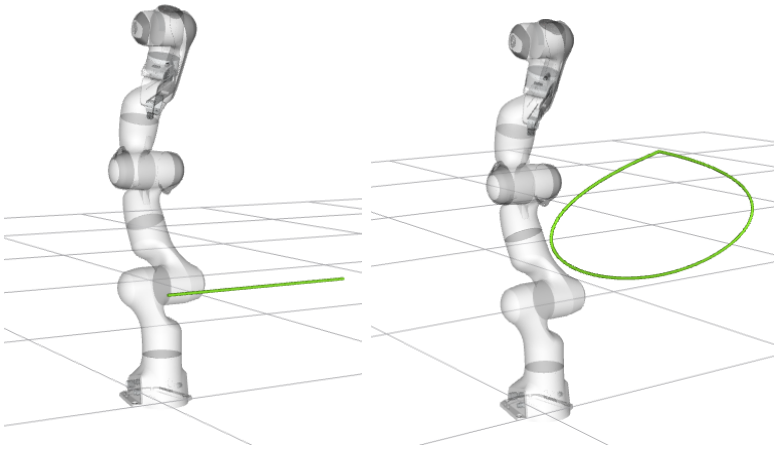


Figure 4.4: Workspace paths used for the experiments

Already in Section 2.7.4.3, we have argued that the workspace trajectory discretization plays an important role for the feasibility of the dynamic programming solution on real hardware. In case

timestamps are already available and the characteristics of the motor controllers are known, equation (2.142) can be used to determine the minimum displacement between waypoints that ensures feasibility. Unfortunately, in the case of time-optimal planning of redundant robots, the time derivative of the workspace points is a result of the optimization process, and, in the case of the Panda, not enough details are available on the profile used by motor controllers to connect waypoints with each other. For these reasons, in our experiments, the number of waypoints is treated as one of the parameters to tune for the algorithm. Then, some a-posteriori considerations can be made to guess the number of waypoints for new workspace paths.

In the setup considered in this dissertation, workspace paths are generated with MATLAB and exported to bagfiles, to be directly exploitable by ROS and MoveIt!, where the dynamic programming algorithm is implemented.

4.3.3 Planning results

The execution of the dynamic programming algorithm that we introduced in Section 4.2 requires a set of parameters to be defined beforehand, that are crucial for the performance of the algorithm, both in terms of quality of the joint-space solution and planning time. As noted in Section 4.1, this is common to other techniques (see, for example, [8]). The quality can be certainly associated to the vicinity to the globally-optimal solution, but, as commented in the previous sections, the trajectory smoothness is also an important factor. When the DP-planned trajectories are sent to the real robot, some discrepancies have to be expected with respect to the plan. They depend on the precision of the dynamic parameters and the model itself, but also on the type of command that is used to control the robot. Hence, another quality measure is associated with the path tracking precision in terms of maximal displacement in the joint or Cartesian space.

The parameters that we can tune to eventually achieve better

results are:

- number of waypoints in the assigned workspace path, as discussed in Section 4.3.2;
- pseudo-velocity limit to be used in the grid computation algorithm; in principle, this should be small enough not to decrease the pseudo-velocity resolution and large enough to contain the maximum pseudo-velocity that the phase-space trajectory can reach, that is not known beforehand;
- pseudo-velocity resolution, determining, on one side, the accuracy in the definition of the phase-space trajectory and, on the opposite, the computational complexity of the dynamic programming algorithm;
- redundancy parameter resolution, determining, on one side, the capability of the redundant manipulator to exploit its null-space to optimize the performance index at hand, and, on the opposite, such as before, the computational complexity of the algorithm.

The redundancy parameter is always selected as the joint position q_4, because it simplifies the analytical inverse kinematics that we perform, as in Section 2.8.2, with *IKFast*.

With respect to pure redundancy resolution at kinematic level, addressed in Chapter 2 and pure time-optimal planning for non-redundant robots, addressed in Chapter 3, here we have a search space of an increased dimension. Practically, we cannot use an arbitrarily fine discretization and more trade-offs should be considered. Therefore, our solutions might be, in some cases, far from the true global optimum, and we should rely on a more relaxed condition of *resolution-optimality*. However, the theoretical results of calculus of variations that we recalled in Section 4.1.1 provide some information about how a time-optimal trajectory for a redundant robot should look like. We can use such results to

provide an indication of "how optimal" the dynamic programming solutions are.

In Section 4.2, we said that the specific discrete approximation for, e.g., equations (4.41) and (4.42), is a user choice. In our experiments, that are based on a single-threaded implementation, we select a simple backward Euler approximation so as to speed up the computation and consume less memory. However, with an high-performance implementation, more complex discrete approximations could be used.

4.3.3.1 Straight line path

Algorithm 8 is first executed on the straight line path presented in Section 4.3.2 with different values for number of waypoints, pseudo-velocity (PV) limit, pseudo-velocity resolution and redundancy parameter (RP) resolution. The results for different sets of parameters are reported in Table 4.2. Together with the cost, the trajectory percentage for which at least one actuator saturates is reported. Since a necessary condition for global optimality is that at least one actuator saturates for each waypoint, globally-optimal solutions should report a saturation of 100%. In our dynamic programming algorithm, saturation is never exact because of discretization, thus an actuator is considered in saturation if its velocity, acceleration, jerk, torque or torque rate is above 90% of its capacity, in agreement with Table 4.1. The first waypoint is excluded in the computation of the saturation percentage because it never saturates by construction, i.e. the manipulator always starts its motion with all quantities equal to zero.

The first aspect that we may notice, by looking at Table 4.2, is that the optimal cost consistently is in a neighborhood of 0.6 s, with a variability of a few milliseconds, which is acceptable for a mechanical system. By increasing the number of waypoints only (see plans 4-6), the cost may increase or decrease. We should remark that more waypoints correspond to a larger number of constraints along the path and less freedom in deviating from the

Plan ID	Length (m)	Number of waypoints	PV limit	RP res. (deg)	PV res.	Cost (s)	Saturation (%)
1	0.5	10	1.4	0.500	0.02	0.655	100
2	0.5	10	1.4	0.250	0.02	0.594	89
3	0.5	10	1.4	0.250	0.01	0.592	89
4	0.5	10	1.4	0.125	0.02	0.574	89
5	0.5	15	1.4	0.125	0.02	0.645	100
6	0.5	20	1.4	0.125	0.02	0.602	95

Table 4.2: Results of Algorithm 8 on the straight line path for different sets of parameters

true straight line. On the other hand, more waypoints allow to reduce the error related to the linear approximation that we make at each step by using a simple Euler integration scheme. As far as the saturation is concerned, ideally, it should be 100% for infinite waypoint, pseudo-velocity and redundancy parameter resolution. However, some combinations of non-infinite values may still yield maximum saturation, as for plans 1 and 5. As expected, with a fixed number of waypoints, the cost decreases for finer resolutions of the redundancy parameter and the pseudo-velocity (see plans 1-4), although the improvement due to the latter, in our experiments, is negligible (see plans 2 and 3).

The actuation limits of the Panda robot (see Table 4.1), in our experiments, are always such that constraints (4.56)-(4.57) activate before constraints (4.58)-(4.60). This is also due to the fact that the planning is performed by considering zero load at the end-effector, while jerk limits are very large. The resolution-optimal joint-space solution for plan 6 is reported in Figure 4.5 and Figure 4.6. The computation time needed to find such a solution is 219 minutes on a 64-bit Ubuntu 18.04 LTS OS running on an Intel® Core™ i7-2600 CPU @ 3.40GHz ×8. No multi-core execution model has been used in the tests.

The reader may notice that, at first, joint 1's acceleration saturates to reach the maximum velocity, which persists for a large portion

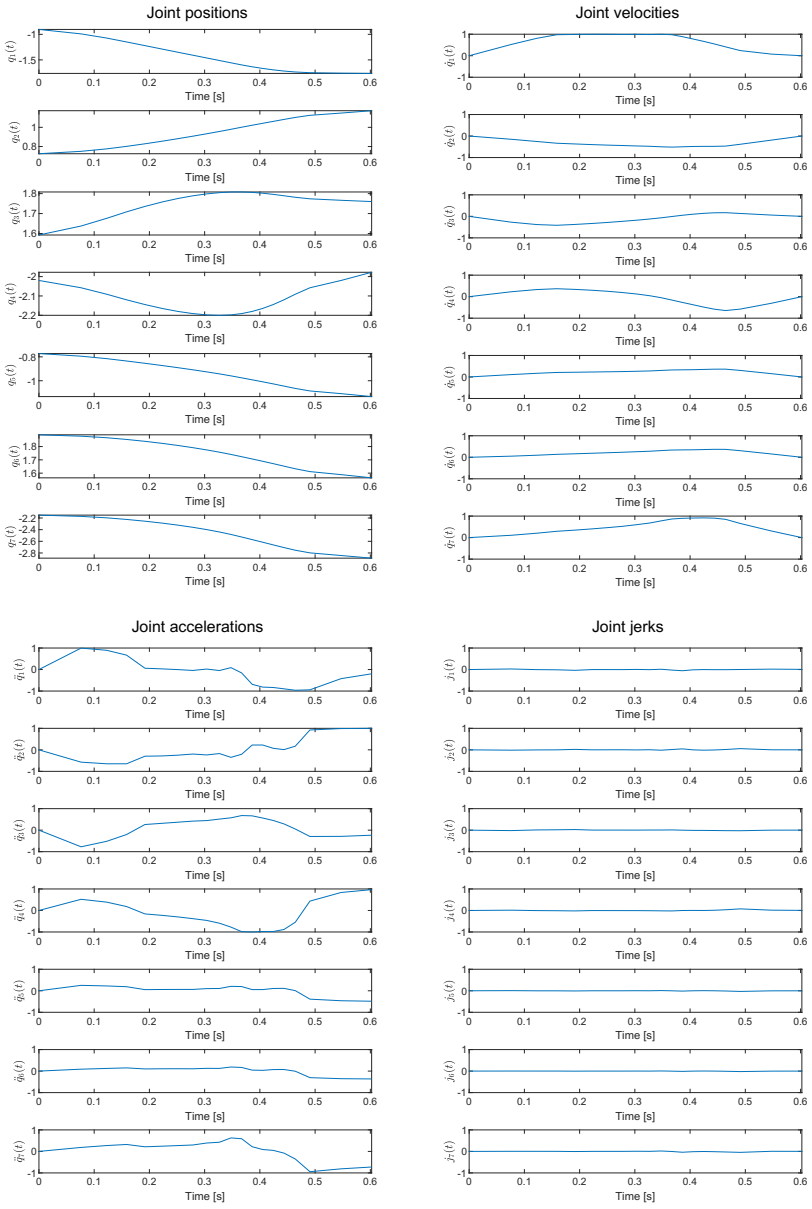


Figure 4.5: Optimal joint positions, velocities, accelerations and jerks (plan 6), normalized in $[-1, 1]$, for the straight line path

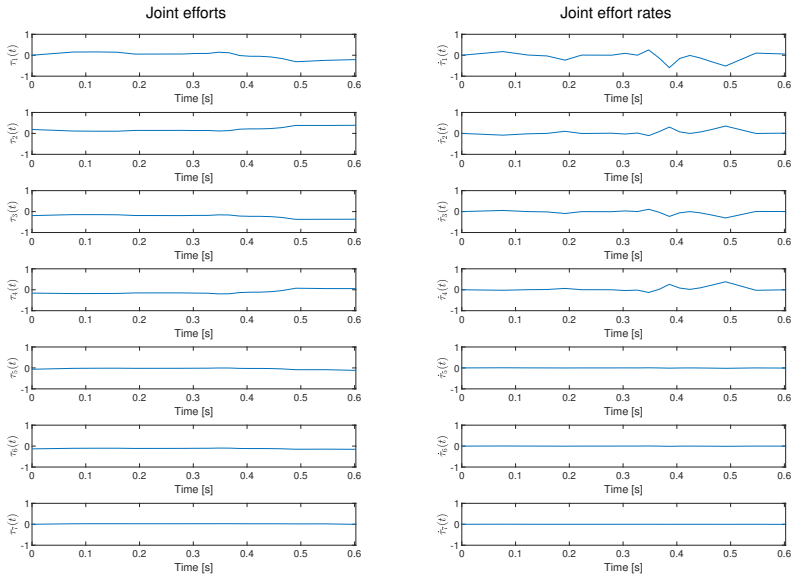


Figure 4.6: Optimal joint torque and torque rates (plan 6) for the straight line path

of the trajectory, between $t = 0.16$ s and $t = 0.37$ s. Here, there is a rather simultaneous saturation of joint 4's acceleration and joint 7's velocity, up until $t = 0.44$ s. In the last segment, joint 1's acceleration saturates again, followed by joint 2's acceleration to perform the final braking. For some segments of the trajectory, two actuators are in saturation at the same time, which is expected for a redundant manipulator, as discussed in Section 4.1.1.

The state space grids for the straight line path with 20 waypoints (plan 6) and joint position q_1 are reported in Figure 4.7. Because of the manipulator geometry and joint limits, only 4 of the 8 grids have feasible configurations. The redundancy parameter, i.e. q_4, is represented along the y-axis in its physical domain (see Table 4.1). The inverse kinematics solutions are returned by *IKFast* in a way that grids are not homogeneous. Although it is easy in this case to separate the extended aspects apart, we will not have any

advantage in doing so, as position constraints are likely to activate before velocity ones (see Section 2.8.3).

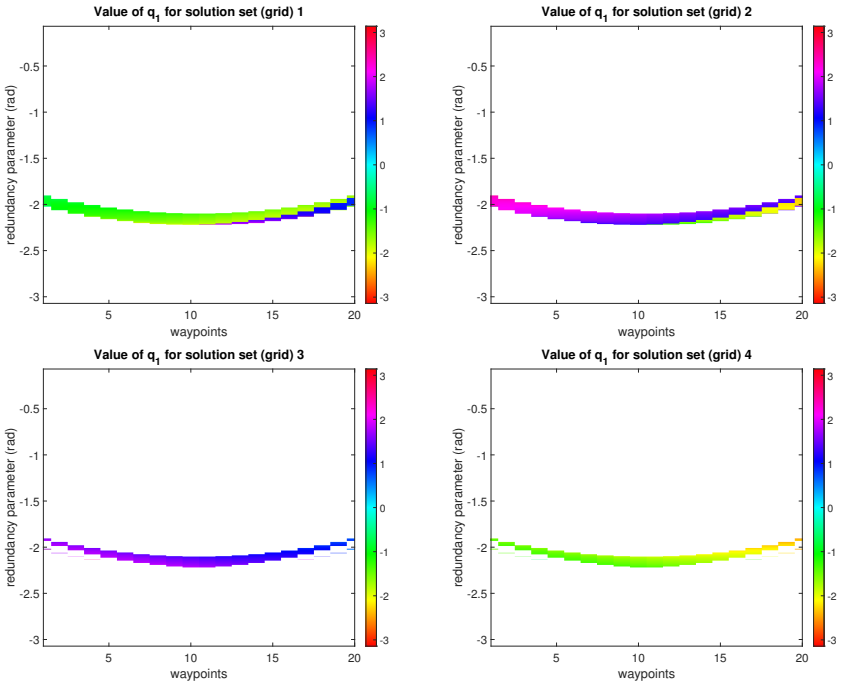


Figure 4.7: Panda grids representing q_1 (in radians) for the straight line path

The phase space trajectory (PST) is reported in Figure 4.8. As noted above, for redundant manipulators, the PST is a function of two independent variables that are λ (the progress along the path) and $\mathbf{v}$, the redundancy parameter(s). The 3D view provides an indication of how the dynamic programming algorithm exploits the redundancy parameter to increase the pseudo-velocity and consequently compute a better solution with respect to a pre-assigned joint-space path. Although it gets very close to the upper limit, the pseudo-velocity does not saturate, meaning that the plan is valid. On the contrary, should the pseudo-velocity hit the limit defined in Table 4.2, an artificial constraint would have been inserted, which requires the plan to be re-computed with an higher

bound.

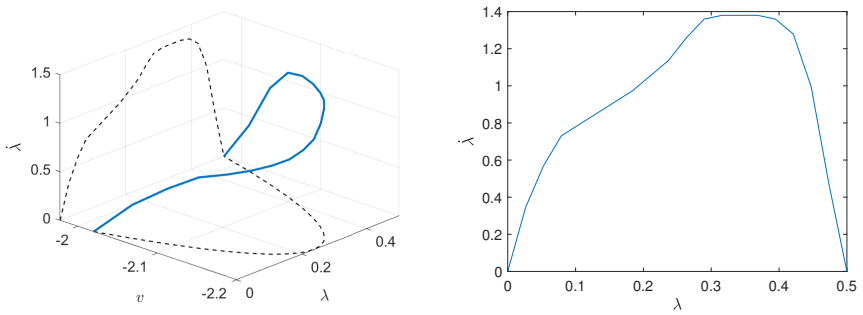


Figure 4.8: On the left, phase-space trajectory (blue) with projections (black) on planes λ-$\dot{\lambda}$ and λ-v; on the right, a front view of the former projection (or phase-plane trajectory)

4.3.3.2 Ellipse-like path

The best parameters for the straight line path are directly re-used for the ellipse-like path, to assess their consistency across different paths. Since the path is longer ($\Lambda = 1.45$ m), a higher number of waypoints should be used to keep a similar spatial resolution. The parameters used for this use case are reported in Table 4.3, together with the optimal cost and percentage of saturation. The resolution-optimal joint-space solution is reported in Figure 4.9. As for the straight line trajectory, velocity and acceleration limits activate first and therefore jerk, torque and torque rate curves are not reported here for the sake of brevity. The computation time needed to find such a solution is about 34 hours on a 64-bit Ubuntu 18.04 LTS OS running on an Intel® Core™ i7-2600 CPU @ 3.40GHz ×8. No multi-core execution model has been used in the tests.

In this case, the joint accelerations are much less smooth than before, which complicates the individuation of the switching points along the trajectory, although some segments show a clear saturation, especially for joint 2 at $t \in [0.08, 0.31]$, $t \in [0.90, 1.09]$ and

Length (m)	Number of waypoints	PV limit	RP res. (deg)	PV res.	Cost (s)	Saturation (%)
1.45	60	1.4	0.125	0.02	1.856	90

Table 4.3: Results of Algorithm 8 on the ellipse-like path with same parameters as plan 6 of Table 4.2

$t \in [1.34, 1.55]$. In the other segments, there is some saturation of velocity, for joints 1, 5 and 7 and acceleration for the same joints.

The state space grids are reported in Figure 4.10. Even in this case, grids are not homogeneous, but Algorithm 8 can still find the optimal solution across all extended aspects. As before, the improvement of performances given by homogeneous grids should be expected to be negligible.

The phase-space trajectory and its projection on the phase plane are drawn in the graphs of Figure 4.11. As before, the pseudo-velocity limit is not reached, meaning that no artificial constraint has been inserted. With respect to the PST of Figure 4.8, we may notice a less smooth curve. Indeed, this is quite a typical behavior of Algorithm 8 and strongly depends on the redundancy parameter discretization and the way inverse kinematics is performed. In Chapter 3, for non-redundant mechanisms, we have seen that the joint space path is directly time-parametrized. The kinematic inversion that is performed to obtain such a path is performed offline and can be designed to generate arbitrarily smooth curves. For instance, one may adopt a second-order integration scheme to guarantee differentiable joint positions. In this case, for a given manipulator, the switching points that are either classified beforehand (such as with the shooting method in the phase plane) or found a-posteriori (such as with dynamic programming) are only due to the workspace path geometry. In the case of Algorithm 8, the joint space path cannot be arbitrarily smooth because of the state space discretization. As a consequence, sharp motions can be required in the joint space in addition to those related to the workspace path geometry. In other words, artificial switching

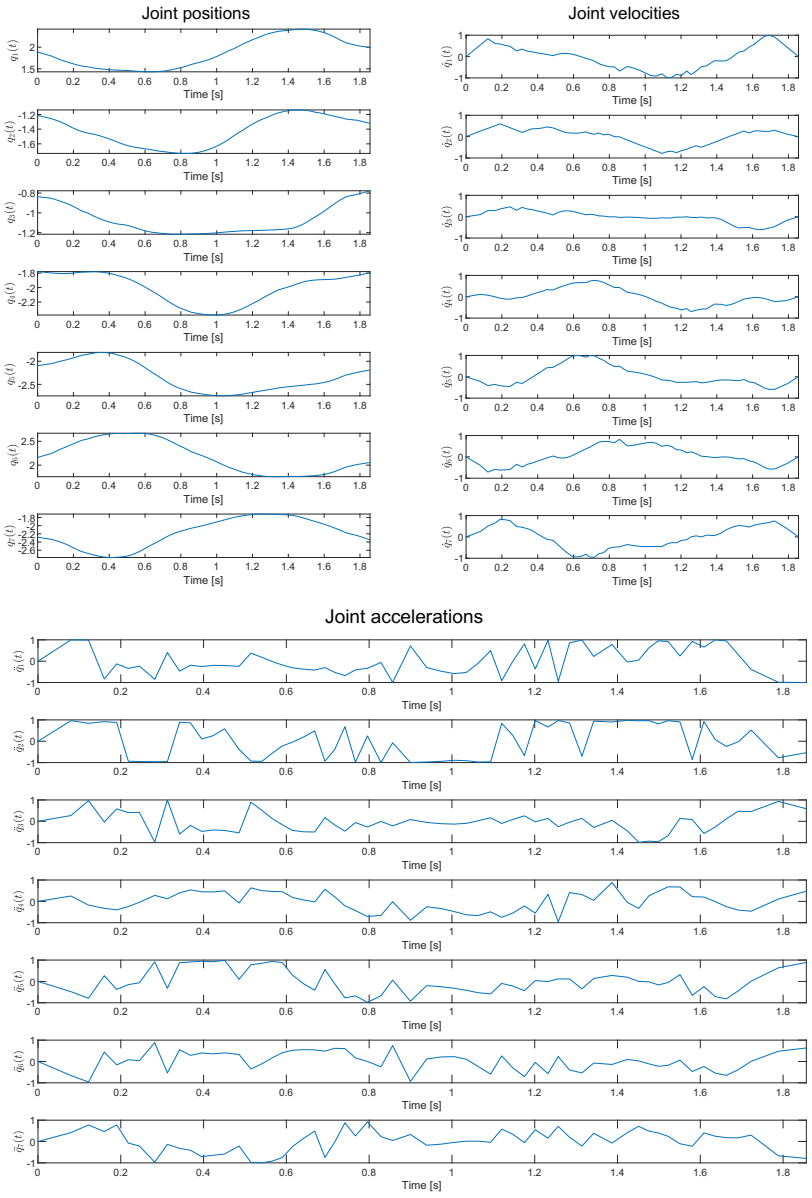


Figure 4.9: Optimal joint positions, velocities and accelerations, normalized in $[-1, 1]$, for the plan of Table 4.3

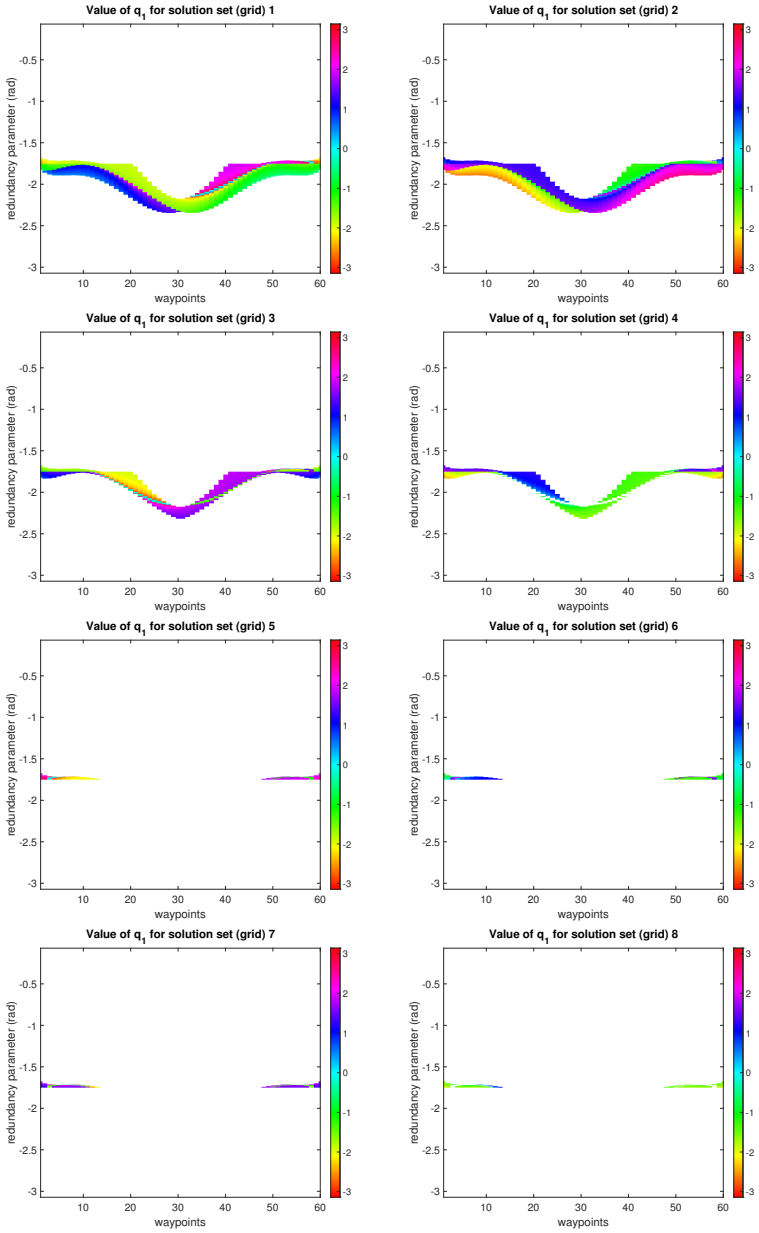


Figure 4.10: Panda grids representing q_1 (in radians) for the ellipse-like path

points can be generated as a result of the state space discretization. Such an effect can be reduced by selecting suitable values of the spatial resolution in the workspace (number of waypoints) and in the joint space (redundancy parameter resolution), as seen for plan 6 of the straight line path, and eliminated, at the limit, only when the discretization step tends to zero. In these conditions, it is clear that the global optimality is compromised in favor of a more relaxed condition of resolution-optimality.

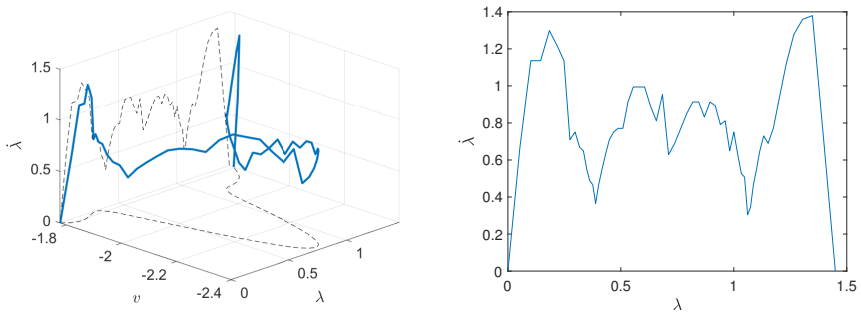


Figure 4.11: On the left, phase-space trajectory (blue) with projections (black) on planes λ-$\dot{\lambda}$ and λ-v; on the right, a front view of the former projection (or phase-plane trajectory)

4.3.4 Execution results

The execution of Algorithm 8 on either trajectory generates several control signals, as seen in figures 4.5, 4.6 and 4.9. Before they can be used as commands for the Panda robot, it is necessary to interpolate them at the frequency of the controller, i.e. 1 kHz. Ideally, since the planning we performed considers the manipulator's dynamic model, commands should be provided as open-loop torques. However, the torque signals in, e.g. Figure 4.6, are characterized by an average frequency of 33 Hz. Increasing it at the planning stage means expanding the planning time considerably, at the point that our dynamic programming algorithm becomes useless in any practical situation. On the other hand,

we should expect that interpolating the open-loop torques in a post-processing stage will introduce errors that ultimately make the performance degrade in terms of tracking accuracy. Also, as noted in Section 4.3.1, the external controller interface requires net torques to be sent to the actuators, meaning that the efforts in Figure 4.6 should be further post-processed to remove gravity and friction contributions.

Alternatively, the robot can be commanded with joint positions and control torques be generated with the internal joint impedance controller, configured with maximum stiffness. Position control has already been used in other works, e.g. [8], dealing as well with execution of time-optimal joint space solutions, with satisfactory results.

A high-level description of the control setup is shown in Figure 4.12. Algorithm 8 is implemented in ROS/MoveIt! and deployed as a node. Workspace paths $\mathbf{x}(\lambda)$ are provided in bagfiles and optimal joint space solutions, i.e. curves $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$ and $\ddot{\mathbf{q}}(t)$ are also stored in bagfiles. The controller manager of `ros_control` is configured to work with the Panda robot hardware interface (provided through `franka_ros`) and to load a robot-agnostic joint trajectory controller (JTC). Internally, the JTC performs interpolation on the input signals to deliver smoother signals to the robot driver. The supported interpolation types are linear, cubic and quintic, depending on the provided input, position only, position and velocity, or position, velocity and acceleration respectively. In our setup, all derivatives are provided, thus a quintic interpolation is performed.

Notoriously, cubic and quintic interpolations guarantee that the interpolated curve passes through the provided samples, but local high-frequency oscillations are generated among them. Since the JTC has no information about the specific robot, such oscillations are likely to violate the kinematic and dynamic limits [14]. Several techniques exist to perform interpolation of control signals and still generate curves that are within bounds. In this case, we exploit some basic signal processing functions that are already

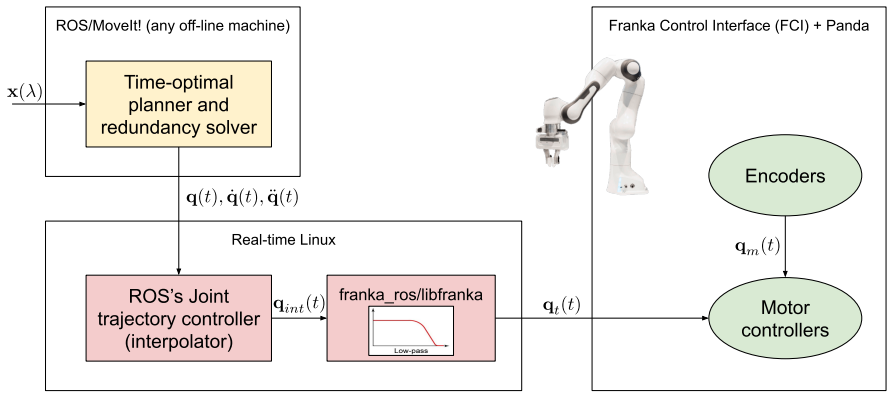


Figure 4.12: Block diagram showing the control setup

provided by *libfranka* and that we discussed in Section 4.3.1. In particular, both the low-pass filter and the rate limiter, shown in Figure 4.3, are active to deliver less oscillating commands and ensure the compliance to the actuator limits. The employment of a filter to smooth the trajectory is also beneficial to eliminate the high frequency content that excites the unmodeled joint elasticity that characterizes joints transmission in time-optimal control [58].

Since additional systems are placed between the planner and the actual control of the robot, that are not modeled in the planner itself, we should expect discrepancies between the planned solution and the measured one. However, we will see that they are negligible and that results comparable to those from other works, e.g. [8], are eventually obtained. In comparing with previous works, we should remark that our plans are calculated with the real robot capacities, as reported in Table 4.1, while conservative values might be employed elsewhere, e.g. a given percentage of the actual limit.

4.3.4.1 Straight line trajectory

A comparison between the planned joint positions and the measured ones for the straight line joint space trajectory of Figure 4.5 and Figure 4.6 is given in Figure 4.13. The maximum error in the

joint space is 0.8 deg, while the actual execution time is 0.619 s, 2.68% slower than planned. The delay introduced by the low-pass filter has been removed to provide a faithful comparison with the planned references. The same has been done for all the graphs shown in this section and in the next. A video of the execution can be found in [112].

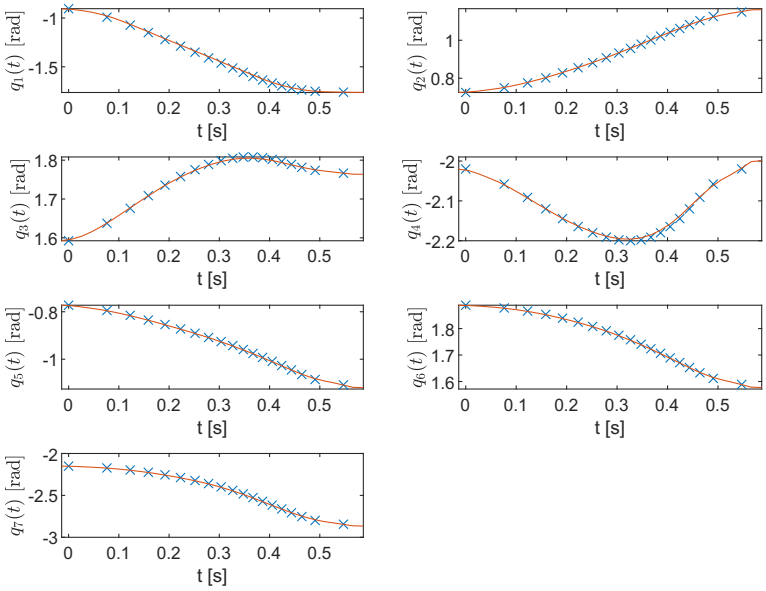


Figure 4.13: Planned (blue) vs measured (red) joint positions for the trajectory of Figure 4.5 and Figure 4.6

The same planned trajectory is scaled in time to be 10% faster and is sent to the robot again to estimate its performance. A comparison between the planned joint positions and the measured ones are reported in Figure 4.14. In this case the robot completes the task in 0.559 s, only 2.08% slower than planned, but the maximum error in the joint space increases to 4.7 deg, as it is evident from the graph of joint 1. As expected, the robot cannot be controlled beyond its capacities and the actual motion results in a remarkable degradation of the tracking performances.

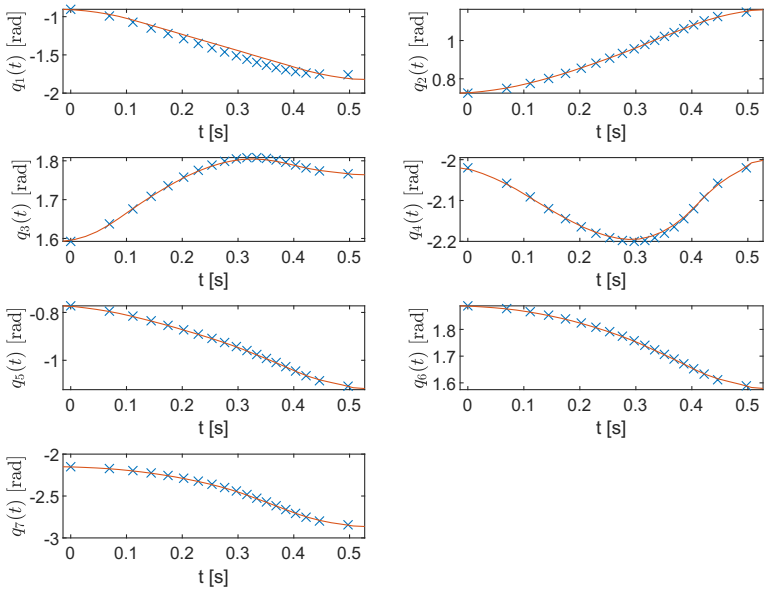


Figure 4.14: Planned (blue) vs measured (red) joint positions for a trajectory 10% faster than the optimum

4.3.4.2 Ellipse-like trajectory

The result of the execution of the more complex trajectory of Figure 4.9 is reported in Figure 4.15. The robot completes the motion in 1.859 s, 0.13% slower than planned with a maximum error in the joint space of 0.8 deg. By providing commands that are 10% faster than the optimal plan, the motion is completed in 1.70 s, 0.67% slower than planned, with a maximum error of 1.86 deg, associated to joint 1 at $t = 0.11$ s. The curves associated to this execution are shown in Figure 4.16. A video of the execution can be found in [112].

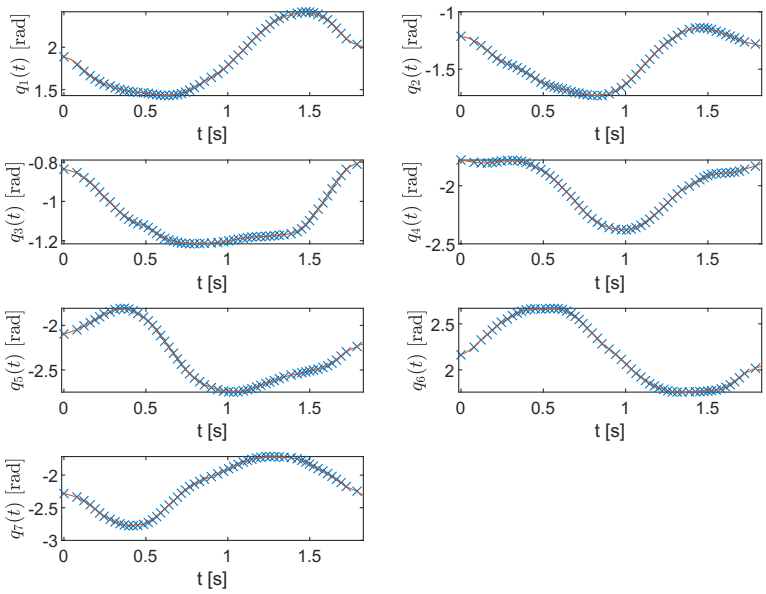


Figure 4.15: Planned (blue) vs measured (red) joint positions for the trajectory of Figure 4.9

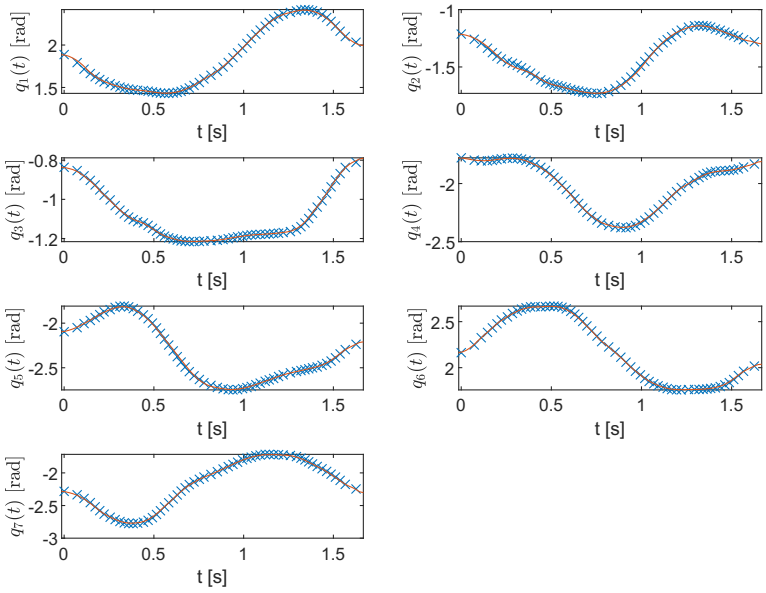


Figure 4.16: Planned (blue) vs measured (red) joint positions for a trajectory 10% faster than the optimum

Chapter 5

Conclusions

5.1 Limitations and improvements on planning

The primary objective of this dissertation was to provide an alternative method (with respect to those described in Section 4.1), based on dynamic programming, to exploit kinematic redundancy to the purpose of time-optimal planning, in an unified framework. The motivation of this work is that dynamic programming is very flexible in the accommodation of several constraints that characterize real applications and that are usually neglected to make the problem fit into a sufficiently straightforward mathematical formulation. Furthermore, even when all the problem requirements can be accommodated and the equations can be established, finding a solution is not an easy task. We have seen in Section 4.1 that some researchers just use their formulations to perform some analysis of the problem, so as to extract distinctive features, and only some of them effectively resolve the problem by coming up with a time-optimal joint-space trajectory. The most representative example is certainly given by multiple-shooting [9, 10, 11], where the underlying optimization technique is the interior-point method. Although the achievement of the globally-optimal solu-

tion is subject to an initial guess on the variables to optimize, the technique is still comparable with dynamic programming in terms of flexibility.

One more advantage of the DP algorithm presented here over others is that it is built on known results of both redundancy resolution and time-optimal planning. In this sense, it is transparent and allows one to perform additional analyses on the problem in addition to computing a time-optimal solution. In fact, the usage of state space grids, that are connected to the manifold representation, allows to understand whether a solution exists, given the geometrical constraints and joint limits, and whether a transition through one or more singularities or semi-singularities is necessary. Regarding time-optimal planning, the proposed method does not deny the curvilinear coordinate parametrization of the path, that still allows representing the solution in the phase space and inferring about its properties. Future research might concentrate on their analysis and possibly on the design of algorithms that are able to compute the maximum velocity surface and use it to provide ad-hoc techniques for time-optimal planning.

Dynamic programming is not the groundbreaking technique for time-optimal planning of redundant robots, or at least, not as defined in this dissertation. This is mainly due to its computational complexity. Although it is perfectly suited to solve one or the other problem separately, it still requires hours to days to solve a medium-complexity planning problem for redundant robots and, most importantly, it is not scalable. The challenges that we are nowadays facing require techniques to be highly flexible and efficient with respect to problems of different dimensions and complexities. In our case, the DP algorithm is not scalable for systems characterized by an higher degree of redundancy, i.e. $r > 1$, and this is an important limitation. Also, if one wanted to impose constraints on higher-order derivatives and, at the same time, preserve global-optimality, the phase space should be further augmented with additional dimensions (see, e.g. [64, 85, 77]). Again, dynamic programming is not scalable to this respect.

To outline some lines of development for the future, while trying to build upon the results of this dissertation, we may identify three main direction: methodological developments, algorithmic enhancements and technological improvements.

5.1.1 Methodological developments: 2-stages approach

We have seen that the employment of dynamic programming is beneficial for many reasons, but a trade-off is established between computation time and optimality of the solution. The trade-off is regulated by the discretization of the axes that compose the state space. We said that, in general, we should accept a more relaxed criterion of optimality, that we call resolution-optimality. Dynamic programming can find the globally-optimal solution for a given discretized state space, but, if the discretization is coarse, the optimum will be far from the "true" optimum (i.e. the globally-optimal solution of the continuous domain problem). We can think of our trajectories in Section 4.3.3 as *close* to the global optimum, but we do not have a measure of this proximity. Future research might concentrate on this aspect and adopt techniques like, for example, that of [8]. Therein, a point-to-point (PTP) globally-optimal planning is performed between two workspace poses and the joint space trajectory is stored as $\mathbf{q}^*(t)$. Then, the joint space solution is transposed in the workspace through direct kinematics, i.e. $\mathbf{x}^*(t) = \mathbf{k}\big(\mathbf{q}^*(t)\big)$. Last, the time law is removed and the path $\mathbf{x}(\lambda)$ is used as input to time-optimal planning along a pre-scribed path, yielding $\mathbf{q}(t)$. The difference $\mathbf{q}(t) - \mathbf{q}^*(t)$ can be evaluated to estimate the quality of the optimization process. The only limit of this approach is that it does not work for a generic path, as it cannot be chosen by the user but is determined by the PTP optimization.

In light of the above, it is licit to expect that some distance exists between our solutions and the global optimum. This means that there is room for 2-stages approaches, like that of Figure 1.1, to

perform better than unified approaches, like that of Figure 1.2. As pointed out several times throughout this dissertation, redundancy resolution and time-optimal planning can be easily solved independently, with an acceptable computational burden, even with dynamic programming. Future developments might regard a deepen study into 2-stages techniques to understand whether some specific objective functions can be used for redundancy resolution (e.g., acceleration capability of the manipulator) that allow minimizing the trajectory tracking time at a later stage. Also, it should be investigated whether global-optimality can be beneficial to this respect, in place of the local optimization methods discussed in Section 4.1.2.

Additional a-posteriori analyses of DP solutions (assuming they are sufficiently close to the global optimum) might yield interesting insights on which objective function to choose for redundancy resolution if a 2-stage approach was adopted. To this end, there is also room for identification and learning techniques that can exploit data collected from different use cases to infer an optimal policy.

5.1.2 Algorithmic enhancements: a comparison with rapidly-exploring random trees

Time-optimal planning along pre-scribed paths is only one example of kineto-dynamic planning. In the literature, several other techniques exist to address this category of problems. Among them, randomized algorithms have attracted and still attract the attention of many researchers, especially because of their simplicity, effectiveness and theoretical guarantees. In this section, a category of them, the *Rapidly-exploring Random Trees* are briefly recalled in order to highlight the commonalities and differences they have with dynamic programming. The objective is to sketch some ideas about mixing the two techniques to design a more efficient algorithm for kineto-dynamic planning.

5.1.2.1 What is an RRT?

Let $\mathcal{C}$ be a d-dimensional state (or configuration) space, $x_{init} \in \mathcal{C}$ the initial state and $\mathcal{X}_{goal} \subset \mathcal{C}$ a set of possible goal states. The name *Rapidly-exploring Random Tree* (RRT) [113] refers to a specific class of algorithms that are able to find a sequence of states in $\mathcal{C}$ (also termed, more generally, a *solution*) connecting x_{init} with any state $x_{goal} \in \mathcal{X}_{goal}$, while avoiding obstacles. This is done by building a tree data structure $\mathcal{T}$ rooted in x_{init}, where the possible solutions are the paths connecting a tree leaf with the tree root.

Very briefly, the main steps taken by an RRT are:

- randomly sample the configuration space to get x_{rand};
- identify the node x_{near} in $\mathcal{T}$ nearest to x_{rand};
- create a new node x_{new} in the direction of x_{rand} starting from x_{near} and add it to $\mathcal{T}$.

A solution is obtained by repeating the steps above as many times as the number of nodes x_{new} necessary to reach $\mathcal{X}_{goal}$.

This very straightforward formulation can be complicated to address the following extensions:

- allow for more than one initial state;
- seek a solution that minimizes or maximizes, in a local or global sense, a given performance index;
- manage kineto-dynamic constraints, with limits on velocities, torques or other constraints;
- allow for bidirectional (equality) constraints, such as, for a manipulator, a task for the end-effector.

All these extensions are clearly required for the time-optimal planning of redundant robots along prescribed paths.

5.1.2.2 Comparison with dynamic programming

The RRTs are not *complete*, while dynamic programming is, since they do not guarantee to find a solution in a given time horizon. They satisfy a more relaxed property of completeness, that is known as *probabilistic completeness* [114], implying that the probability of an algorithm to fail in finding a solution tends to zero as the number of iterations (i.e., in this case, the number of nodes in the solution sequence) tends to infinity.

The classic RRT algorithm is not optimal, while an optimal algorithm exists, called RRT* [115]. Even in this case, RRT* is not globally-optimal in a strict sense, but more precisely is *asymptotically optimal*, meaning that the globally-optimal solution is reached asymptotically as the number of iterations tends to infinity.

In order to draw a comparison with dynamic programming, we should remark that classical DP, that is based on the Bellman's optimality principle, is indeed globally-optimal in a strict sense. In other words, it is able to optimize the integral over the path of the assigned cost function. However, in practice, when DP deals with a continuous domain, it is affected by the well-known *curse of dimensionality*, which obliges, from the practical standpoint, to work with a discretized state space. Hence, as remarked several times throughout the dissertation, the DP algorithm will only be *resolution-optimal*. As the resolution of the discretization process becomes finer and, ultimately, tends to infinity, the solution asymptotically converges to the globally-optimal one.

Concerning again the optimality, it is lastly interesting to notice that, in case the space of all possible sequences from x_{init} to $\mathcal{X}_{goal}$ is characterized by different homotopy classes, both RRT* and DP are able to asymptotically converge to the global optimum across all homotopy classes [115].

From a procedural standpoint, RRT* and DP differ in that

- RRT* randomly extracts samples from the continuous con-

figuration space and evaluates transitions in the direction of such samples only starting from the nodes that, in the tree, are considered close to it;

- DP discretizes the configuration space and evaluates transitions between all combinations of nodes between two adjacent clusters. In other words, the adjacency is determined by construction.

This distinction draws a clear dividing line between the two techniques when it comes to the enforcement of constraints. Keeping aside possible variants and extensions of both classes of algorithms, the enforcement of constraints is

- *beneficial* for DP, as it allows to limit the number of evaluations that are made between two adjacent clusters;

- *harmful* for RRT because it leads to trashing non-feasible states after that computations have been made on them. This is particularly true for RRTs working with kineto-dynamic systems, where computations involve the integration of differential equations.

In order to confirm the above, we may have a look at the computational complexity. The DP time complexity is $O(N_i N_u^{2d})$, where N_i is the time (or curvilinear coordinate) dimension and indicates the progress towards the goal state, while N_u is the number of discrete samples for the d dimensions of the configuration space. In case of constraints, as observed in Section 2.7.7, the computational complexity reduces to $O(N_i N_w^d N_u^d)$, where $N_w << N_u$ and refers to the cardinality of a *window* inside a cluster, satisfying the constraints. Ultimately, N_w, for highly-constrained systems, may just contain a few nodes. For RRT, the time complexity is $O(k \log k \log^d c)$ [115], where k is the number of samples in the final solution and it is, in general, different from N_i, while c is connected to the number of constraints in the configuration space.

The usual RRT formulation, as briefly commented above, and that of its optimal variant RRT* foresee only one initial state x_{init}, that also is the root of the tree $\mathcal{T}$. However, although this is the case of the majority of planning problems, it is not suitable for off-line planning of repeatable motions, as in manufacturing plants, where the interest is in finding both the initial and final configurations as a result of the optimization problem. However, it could be possible that RRTs can be extended to this respect to introduce a "virtual" initial state, which becomes the root of the tree. Then, the states satisfying the constraints on the initial conditions can be picked as children of the root. Alternatively, one should generate a new tree for each $x_{init} \in \mathcal{X}_{init}$, where $\mathcal{X}_{init} \subset \mathcal{C}$ represents the set of the states satisfying the initial conditions.

The performance of the RRT, as well as the optimality of its optimal variant RRT*, depend on a certain number of configuration parameters. This is indeed common to several algorithms characterized by one or more random components, such as genetic algorithms (GA), to mention a class that we have analyzed in this dissertation. However, an important difference with GA is that RRT* can guarantee global optimality (once again, asymptotically), while GA convergence to the global optimum does not have any theoretical foundation. Also, the probabilistic theory behind RRT* provides conditions by which the configuration parameters guarantee global optimality.

Regarding the performance of RRT in general, it is mainly affected by the following two parameters [114]:

- amplitude of the step between a node in the tree and its children;
- bias towards the goal state.

5.1.2.3 Planning for kineto-dynamic systems

RRT was originally designed to tackle a class of planning problems that is much broader than kineto-dynamic planning [113]. For

this reason, several adaptations have been necessary to be able to effectively plan in presence of differential state equations.

The main challenge of this kind of planning problems is to connect two known states in the configuration space through differential state equations, piloted by a control input, which means to solve a TPBVP. As discussed before, this kind of connections, between a node in the tree and a target one generated by random sampling, is the core of RRT.

In order to address the challenge, the Authors of [116] propose to apply, starting from the node in the tree, sequences of control inputs driving the system to a state that is *sufficiently close* to the target one, without reaching it exactly. The search for the best sequence of inputs, among the available ones, can be done with different techniques, such as the shooting method.

Compared to the classic RRT, RRT* foresees the following additional steps to guarantee asymptotic convergence to the globally-optimal solution [115]:

- after x_{new} has been connected to his parent x_{near}, new connections are tried with any other tree node x_{next} in a neighborhood of x_{new}, even though x_{next} is not a leaf of the tree;
- a connection from x_{new} to x_{next} is established if the cost is lower than that of the existing path connecting x_{init} to x_{next};
- in order to keep the tree structure, the connection between x_{next} and its old parent node has to be removed;
- the cost is updated for all the nodes children of x_{next}, since a lower cost path is now available toward the tree root, passing by x_{new}.

The operations above are known as *rewiring* and are particularly delicate when it comes to kineto-dynamic systems [116]. In fact, theoretically, a TPBVP should be solved for every x_{next} that is rewired toward x_{new}. Even in this case, because of the presence of

differential equations, it is usually accepted to reach a neighborhood of x_{next}, say the state $\tilde{x}_{next}$. Then, since x_{next} already had children, the entire sub-tree rooted in x_{next} must be re-computed starting from $\tilde{x}_{next}$ and this is usually done by replaying the pre-stored inputs. On one hand, this operation is computationally very expensive, as it has to be repeated for every node in the neighborhood of x_{new} yielding a lower cost, on the other, constraints are not re-checked when the nodes are re-computed, meaning that they can be violated. In [116], the Authors observe that the unfeasible solutions are later automatically excluded by the optimization process, but a formal proof is not provided.

In the case of bidirectional constraints, like an end-effector path constraint, the rewiring operation certainly yields the violation of the constraint, since the possibly negligible error between x_{next} and $\tilde{x}_{next}$ is later integrated for all the subsequent nodes, ultimately deviating from the path.

Since nodes are connected by respecting the state equation, one of the advantages of employing RRT* for kineto-dynamic systems is that the resulting solution does not need to be post-processed before it can be sent to the robot controllers.

5.1.2.4 Bi-directional constraints

As anticipated above, an interesting use case for RRT and RRT* is when the problem formulation includes bi-directional constraints, especially for kineto-dynamic systems.

The first remark is that sampling on the whole configuration space is not efficient since the large majority of the states will not comply with the bi-directional constraint and will need to be discarded. Rather, it is convenient to work directly on a subspace of $\mathcal{C}$ where all the configurations therein already satisfy the bi-directional constraint [117].

If the bi-directional constraint is given by a path in a multi-dimensional space, as for time-optimal planning of redundant ro-

bots across prescribed paths, and the geometry is not simply describable analytically, it is also very inefficient to work on a continuous state space. For this reason, in [117], the authors have accepted a discretization of the constraint (i.e. the path) and restrict the sampling only on such a discrete domain.

For instance, in the case of a path constraint $\mathbf{x}(\lambda) = \mathbf{k}(\mathbf{q})$ assigned to a redundant robot, the sampling procedure is the following:

- the domain of the curvlinear coordinate λ is discretized, say in $[0, 1]$ or $[0, \Lambda]$, being Λ the path length;
- λ_{rand} is obtained by randomly sampling the continuous domain of λ;
- the discrete value closest to the sample λ_{rand} is picked, say $\lambda(i)$, corresponding to a given $\mathbf{x}(\lambda(i)) = \mathbf{x}(i)$;
- the redundancy parameter is randomly sampled in its domain: if the redundancy parameter is one of the joints, we may call this sample $q_{j,rand}$;
- the remaining joint positions $\mathbf{q}_r$ are computed as

$$\mathbf{q}_r = \mathbf{k}^{-1}(\mathbf{x}(i), q_{j,rand}), \tag{5.1}$$

  yielding a complete random configuration $\mathbf{q}_{rand}$.

By following this procedure, the path constraint is respected by construction.

The RRT (or RRT*) can proceed as usual by identifying the nearest node in the tree $\mathbf{q}_{near}$. Since the path typically needs to be followed entirely, and it is now discretized, it is convenient to pick $\mathbf{q}_{next}$ directly from the next sample on the path (or the previous one if the end-effector is allowed to reverse its direction on the path) [117]. Thus, if $\mathbf{q}_{near} = \mathbf{q}_{near}(k)$, $\mathbf{q}_{new} = \mathbf{q}_{new}(k-1)$ or $\mathbf{q}_{new} = \mathbf{q}_{new}(k+1)$. In the case of time-optimal planning, only the latter will hold.

If the discretization of λ is fine, it is perhaps possible to avoid the exact integration of the state equation and to adopt a simpler discrete approximation, as it is done for our dynamic programming algorithm.

Clearly, this does not exclude the possibility of picking $\mathbf{q}_{rand}$ and, consequently, $\mathbf{q}_{new}$ in proximity to some intermediate node in the tree, which actually is a necessary condition for RRT* to optimize the existing solution.

By adopting the procedure described in this section, RRT* becomes very similar to DP, with the following differences:

- in RRT*, the state space, except for waypoints, is not discretized;
- in RRT*, transitions are not evaluated between all nodes in adjacent clusters, but only between the neighbors of $\mathbf{q}_{new}$.

A deepen analysis of these aspects may yield interesting developments in dynamic programming applied to planning of redundant robots, that would allow to

- design an *anytime* version of dynamic programming that adapts to the available planning horizon, by delivering solutions that asymptotically tend to the global optimum;
- generate smoother results as the discretizations of the redundancy parameters and pseudo-velocity are no longer necessary.

5.1.3 Algorithmic enhancements: heuristics

We have seen that the computational complexity of the algorithm is quadratic with respect to all the dimensions that are not the path and this cannot be avoided with the classical DP formulation. In Section 2.7.7, we analyzed the problem in greater detail to

highlight that the constraints play an important role in reducing the number of nodes that, at each stage, are enabled and subject to comparisons with neighbors. We also demonstrated that the CPU time can be greatly reduced if the properties of such constraints are exploited in the design of the algorithm.

Indeed, nodes can be excluded if they do not satisfy the constraints, but their cost could also be an indication of whether they will belong to the globally-optimal solution or not. In a forward implementation of the algorithm, for each node j, the grid stores the cumulative cost $I_j(i)$ associated with the optimal sub-trajectory connecting it with an initial state. The total cost of the trajectory passing through j will be given by

$$I_j(N_i) = I_j(i) + H_j(i) \tag{5.2}$$

where $H_j(i)$ is the cost associated to the optimal sub-trajectory that connects the node j with a final state. For each $i < N_i$, $H_j(i)$ is unknown. Further research might tackle the problem of finding a consistent heuristic that allows determining an estimation for $H_j(i)$ such that the number of enabled nodes could be further reduced, preserving global optimality. Such heuristic function could be based on the topological characteristics of the problem and the concept of $\mathcal{C}$-path homotopy class introduced in this dissertation, or on locally-optimal resolution techniques.

5.1.4 Technological improvements: parallelism

Dynamic programming, as we have defined it, i.e. like a graph search problem, is highly parallelizable. The parallelism can be implemented at several levels of granularity:

- with a bi-directional scan: forward and backward DP algorithms are run together and optimal solutions are merged at some intermediate waypoint of the path;
- each grid (corresponding to a different inverse kinematic solution, or extended aspect) can be scanned independently,

but transition points along the manifold are shared resources that should be suitably managed;

- the comparisons between a node in a cluster at $i+1$ and all the nodes in clusters at i can be performed independently, as the former is a read-write resource, while the latter are read-only resources (see Figure 5.1).

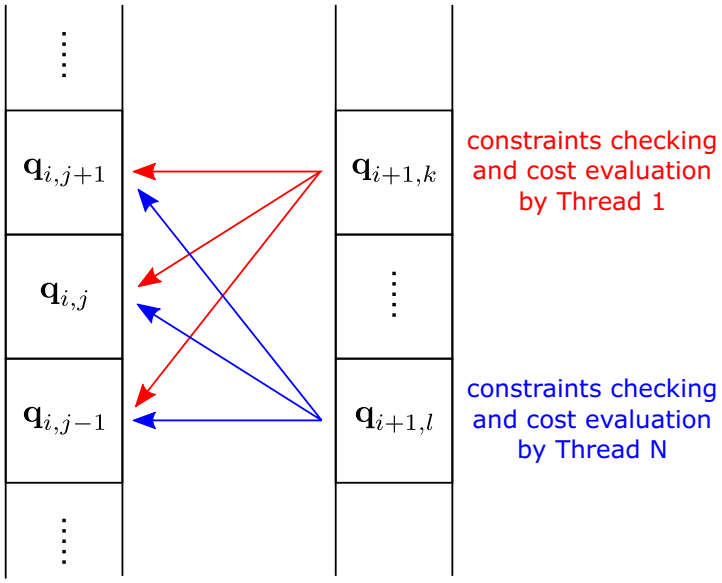


Figure 5.1: Dynamic programming parallelism at node level

The parallelism can be implemented at thread level, or the node-level parallelism, where the operations to be performed are simple and atomic, can be implemented on a dedicated hardware, like graphical processing units (GPU). In the aerospace sector, in ground centers, parallelism is also often achieved through processing clusters that allow for a much higher computational power.

5.2 Limitations and improvements on control

Our dynamic programming algorithm is focused on finding the optimal time law that is encoded in $\dot{\lambda}(\lambda)$. Since all the quantities of interest are connected to $\dot{\lambda}(\lambda)$, any kind of control signal can be generated. Unfortunately, because of the discretization of the state space, the control is only defined at the waypoints, while its behavior between them is undetermined. On the other hand, the controller period is much smaller than the planner's, thus some kind of interpolation is necessary. In [14], it is performed at planning level, so that continuous and differentiable phase-plane profiles can be generated by the dynamic programming algorithm itself, while in Section 4.3, we addressed this issue in post-processing, using interpolation, low-pass filtering and rate limiting. Clearly, the best solution might depend on the specific manipulator and primary controller provided by the robot manufacturer. Some different ways of coping with a non-modifiable primary controller have been commented in Section 3.8.

When a smooth control signal is available, the control still has to cope with uncertainties in the dynamics of the manipulator, dynamics associated to the controller itself, and torque saturation. To this end, future research may address this issue for redundant robots, considering that, compared to non-redundant ones, they provide greater flexibility, even in adapting to tracking errors. Pseudo-velocity and pseudo-acceleration are used at control level to scale down the velocity along the path (see Section 3.8), but, with a redundant manipulator, self-motion in the null-space could also be used to compensate for poor tracking. Techniques that use non-saturating actuators to counterbalance the error on the saturating ones also look promising to this respect.

In order to help the controller in keeping the tracking error bounded, the planner could be further extended to consider the controller dynamics. This is especially useful when the control has complex

dynamics due to closed-loop control (as in the case of position and velocity control, that we also used in the experiments of Section 4.3).

5.3 Application to other systems

Although the formulation of the problem with dynamic programming is quite general, we only applied it to relatively simple serial chains whose task is to follow a pre-determined path in minimum time, without exchanging forces with the surrounding environment. In reality, many other applications exist that expose some kinematic redundancy. In fact, given the flexibility of redundant robots, they are more and more frequently employed in many fields, including manufacturing, service and medical robotics, aerospace robotics. There, robots are often required to interact with the environment or with each other, or, more in general, collaborate to achieve a common goal.

In case the robot has to exert forces on external bodies, our dynamic programming algorithm can be employed with some minimal modifications. In fact, the only extension concerns the introduction of a suitable interaction model. To this end, the flexibility of dynamic programming, for a non-redundant case, has been already demonstrated in [70], but more interesting scenarios can be investigated, depending on the type of the interaction and degree of redundancy. Again, [70] is a first demonstration of applying time-optimal planning with dynamic programming to a cooperative grasping scenario, but future research may address the problem of redundancy in closed kinematic chains, when time-optimal motions are planned.

Lastly, the application of the proposed algorithm (or, more probably, extensions of it) to more complex systems, for off-line planning or benchmarking of algorithms, is of utmost scientific and practical interest. They include, among others, parallel robots, mobile manipulators and flying manipulators.

Bibliography

[1] International Space Exploration Coordination Group (ISECG). (2018, January) The global exploration roadmap. [Online]. Available: https://www.globalspaceexploration.org/wordpress/wp-content/isecg/GER_2018_small_mobile.pdf

[2] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, "A review of space robotics technologies for on-orbit servicing," *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, July 2014. [Online]. Available: https://doi.org/10.1016%2Fj.paerosci.2014.03.002

[3] J. L. Forshaw, G. S. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, R. Axthelm, S. Barraclough, A. Ratcliffe, C. Bernal, F. Chaumette, A. Pollini, and W. H. Steyn, "RemoveDEBRIS: an in-orbit active debris removal demonstration mission," *Acta Astronautica*, vol. 127, pp. 448–463, October 2016. [Online]. Available: https://doi.org/10.1016%2Fj.actaastro.2016.06.018

[4] R. N. Hoyt, J. Cushing, G. Jimmerson, J. T. Slostad, R. Dyer, and S. Alvarado, "SpiderFab: Process for on-orbit construction of kilometer-scale apertures," 2018.

[5] H. Arai, K. Tanie, and S. Tachi, "Path tracking control of a manipulator considering torque saturation," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 1, pp. 25–31, 1994. [Online]. Available: https://doi.org/10.1109%2F41.281604

[6] P. Chiacchio, "Exploiting redundancy in minimum-time path following robot control," in *1990 American Control Conference*.

IEEE, May 1990, pp. 2313–2318. [Online]. Available: https://doi.org/10.23919%2Facc.1990.4791142

[7] F. Basile and P. Chiacchio, "A contribution to minimum-time task-space path-following problem for redundant manipulators," *Robotica*, vol. 21, no. 2, pp. 137–142, February 2003. [Online]. Available: https://doi.org/10.1017%2Fs0263574702004678

[8] K. Al Khudir and A. De Luca, "Faster motion on cartesian paths exploiting robot redundancy at the acceleration level," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3553–3560, October 2018. [Online]. Available: https://doi.org/10.1109%2Flra.2018.2853806

[9] A. Reiter, A. Müller, and H. Gattringer, "Inverse kinematics in minimum-time trajectory planning for kinematically redundant manipulators," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, October 2016, pp. 6873–6878. [Online]. Available: https://doi.org/10.1109%2Fiecon.2016.7793436

[10] A. Reiter, H. Gattringer, and A. Müller, "Redundancy resolution in minimum-time path tracking of robotic manipulators," in *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*. SCITEPRESS - Science and Technology Publications, 2016, pp. 61–68. [Online]. Available: https://doi.org/10.5220%2F0005975800610068

[11] A. Reiter, A. Müller, and H. Gattringer, "On higher order inverse kinematics methods in time-optimal trajectory planning for kinematically redundant manipulators," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1681–1690, April 2018. [Online]. Available: https://doi.org/10.1109%2Ftii.2018.2792002

[12] A. Guigue, M. Ahmadi, R. Langlois, and M. J. D. Hayes, "Pareto optimality and multiobjective trajectory planning for a 7-dof redundant manipulator," *IEEE Transactions on Robotics*, vol. 26, no. 6, pp. 1094–1099, December 2010. [Online]. Available: https://doi.org/10.1109%2Ftro.2010.2068650

[13] E. Ferrentino and P. Chiacchio, "On the optimal resolution of inverse kinematics for redundant manipulators using a topological analysis," *Journal of Mechanisms and Robotics*, vol. 12, no. 3, June 2020. [Online]. Available: https://doi.org/10.1115%2F1.4045178

[14] D. Kaserer, H. Gattringer, and A. Müller, "Nearly optimal path following with jerk and torque rate limits using dynamic programming," *IEEE Transactions on Robotics*, vol. 35, no. 2, pp. 521–528, April 2019. [Online]. Available: https://doi.org/10.1109%2Ftro.2018.2880120

[15] J. P. Mallet, *Parallel robots*, 2nd ed. Springer Netherlands, 2006.

[16] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer International Publishing, 2016.

[17] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 12, pp. 868–871, 1977. [Online]. Available: https://doi.org/10.1109%2Ftsmc.1977.4309644

[18] C. A. Klein, "Use of redundancy in the design of robotic systems," in *Robotics Research: The Second International Symposium*. MIT Press, August 1985, pp. 207–214.

[19] D. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on Man Machine Systems*, vol. 10, no. 2, pp. 47–53, June 1969. [Online]. Available: https://doi.org/10.1109%2Ftmms.1969.299896

[20] J. M. Hollerbach and K. C. Suh, "Redundancy resolution of manipulators through torque optimization," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 308–316, August 1987. [Online]. Available: https://doi.org/10.1109%2Fjra.1987.1087111

[21] D. R. Baker and C. W. Wampler, "On the inverse kinematics of redundant manipulators," *The International Journal of Robotics*

Research, vol. 7, no. 2, pp. 3–21, April 1988. [Online]. Available: https://doi.org/10.1177%2F027836498800700201

[22] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano, "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy," *The International Journal of Robotics Research*, vol. 10, no. 4, pp. 410–425, August 1991. [Online]. Available: https://doi.org/10.1177%2F027836499101000409

[23] K. C. Suh and J. M. Hollerbach, "Local versus global torque optimization of redundant manipulators," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 1987, pp. 619–624. [Online]. Available: https://doi.org/10.1109%2Frobot.1987.1087955

[24] K. Kazerounian and Z. Wang, "Global versus local optimization in redundancy resolution of robotic manipulators," *The International Journal of Robotics Research*, vol. 7, no. 5, pp. 3–12, October 1988. [Online]. Available: https://doi.org/10.1177%2F027836498800700501

[25] D. P. Martin, J. Baillieul, and J. M. Hollerbach, "Resolution of kinematic redundancy using optimization techniques," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 4, pp. 529–533, 1989. [Online]. Available: https://doi.org/10.1109/70.88067

[26] J. J. Rice and J. M. Schimmels, "Multi-homotopy class optimal path planning for manipulation with one degree of redundancy," *Mechanism and Machine Theory*, vol. 149, July 2020. [Online]. Available: https://doi.org/10.1016%2Fj.mechmachtheory.2020.103834

[27] K. Gotlih, I. Troch, and K. Jezernik, "Global optimal control of redundant robot," *Robotica*, vol. 14, no. 2, pp. 131–140, March 1996. [Online]. Available: https://doi.org/10.1017%2Fs0263574700019044

[28] B. W. Choi, J. H. Won, and M. J. Chung, "Optimal redundancy resolution of a kinematically redundant manipulator

for a cyclic task," *Journal of Robotic Systems*, vol. 9, no. 4, pp. 481–503, June 1992. [Online]. Available: https://doi.org/10.1002%2Frob.4620090404

[29] Z. Wang and K. Kazerounian, "An efficient algorithm for global optimization in redundant manipulations," *Journal of Mechanisms Transmissions and Automation in Design*, vol. 111, no. 4, pp. 488–493, 1989. [Online]. Available: https://doi.org/10.1115%2F1.3259026

[30] Y. Nakamura and H. Hanafusa, "Optimal redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 1, pp. 32–42, March 1987. [Online]. Available: https://doi.org/10.1177%2F027836498700600103

[31] S.-W. Kim, K.-B. Park, and J.-J. Lee, "Redundancy resolution of robot manipulators using optimal kinematic control," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation.* IEEE Comput. Soc. Press, 1994, pp. 683–688. [Online]. Available: https://doi.org/10.1109%2Frobot.1994.351407

[32] Y. Zhang, Y. Liu, Z. Xie, and M. Moallem, "Optimal reaction control for the flexible base redundant manipulator system," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO).* IEEE, December 2019, pp. 515–520. [Online]. Available: https://doi.org/10.1109%2Frobio49542.2019.8961851

[33] Y. C. Chen and K. O'Neil, "Stabilization of pseudoinverse acceleration control of redundant mechanisms," in *Robotics 98.* American Society of Civil Engineers, April 1998, pp. 293–299. [Online]. Available: https://doi.org/10.1061%2F40337%28205%2943

[34] J. W. Burdick, "On the inverse kinematics of redundant manipulators: characterization of the self-motion manifolds," in *IEEE International Conference on Robotics and Automation.* IEEE, 1989, pp. 264–270. [Online]. Available: https://doi.org/10.1109%2Frobot.1989.99999

[35] A. P. Pashkevich, A. B. Dolgui, and O. A. Chumakov, "Multiobjective optimization of robot motion for laser cutting applications," *International Journal of Computer Integrated Manufacturing*, vol. 17, no. 2, pp. 171–183, March 2004. [Online]. Available: https://doi.org/10.1080%2F0951192031000078202

[36] C. L. Lück and S. Lee, "Self-motion topology for redundant manipulators with joint limits," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3. IEEE Comput. Soc. Press, 1993, pp. 626–631. [Online]. Available: https://doi.org/10.1109%2Frobot.1993.291835

[37] P. Borrel and A. Liegeois, "A study of multiple manipulator inverse kinematic solutions with applications to trajectory planning and workspace determination," in *Proceedings - International Conference on Robotics and Automation*. IEEE, 1986, pp. 1180–1185. [Online]. Available: https://doi.org/10.1109%2Frobot.1986.1087554

[38] P. Wenger, "A new general formalism for the kinematic analysis of all nonredundant manipulators," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE Comput. Soc. Press, 1992, pp. 442–447. [Online]. Available: https://doi.org/10.1109%2Frobot.1992.220300

[39] J. A. Pámanes, P. Wenger, and J. L. Zapata, "Motion planning of redundant manipulators for specified trajectory tasks," in *Advances in Robot Kinematics*. Springer Netherlands, 2002, pp. 203–212. [Online]. Available: https://doi.org/10.1007%2F978-94-017-0657-5_22

[40] P. Wenger, P. Chedmail, and F. Reynier, "A global analysis of following trajectories by redundant manipulators in the presence of obstacles," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE Comput. Soc. Press, 1993, pp. 901–906. [Online]. Available: https://doi.org/10.1109%2Frobot.1993.292258

[41] J. W. Burdick, "Global kinematics for manipulator planning and control," in *Intelligent Control and Adaptive Systems*,

G. Rodriguez, Ed. SPIE, February 1990, pp. 57–68. [Online]. Available: https://doi.org/10.1117%2F12.969907

[42] D. K. Pai and M. C. Leu, "Genericity and singularities of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 545–559, 1992. [Online]. Available: https://doi.org/10.1109%2F70.163780

[43] P. Wenger, "Classification of 3R positioning manipulators," *Journal of Mechanical Design*, vol. 120, no. 2, pp. 327–332, June 1998. [Online]. Available: https://doi.org/10.1115%2F1.2826976

[44] A. Guigue, M. Ahmadi, M. J. D. Hayes, R. Langlois, and F. C. Tang, "A dynamic programming approach to redundancy resolution with multiple criteria," in *IEEE International Conference on Robotics and Automation.* IEEE, 2007, pp. 1375–1380. [Online]. Available: https://doi.org/10.1109%2Frobot.2007.363176

[45] Z. Zhou and C. C. Nguyen, "Globally optimal trajectory planning for redundant manipulators using state space augmentation method," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 19, no. 1, pp. 105–117, 1997. [Online]. Available: https://doi.org/10.1023/A:1007905817998

[46] Y. Shen and K. Huper, "Optimal trajectory planning of manipulators subject to motion constraints," in *ICAR '05. Proceedings of the 12th International Conference on Advanced Robotics, 2005.* IEEE, 2005, pp. 9–16. [Online]. Available: https://doi.org/10.1109%2Ficar.2005.1507384

[47] A. Dolgui and A. Pashkevich, "Manipulator motion planning for high-speed robotic laser cutting," *International Journal of Production Research*, vol. 47, no. 20, pp. 5691–5715, July 2009. [Online]. Available: https://doi.org/10.1080%2F00207540802070967

[48] D. Reveles, J. A. Pámanes, and P. Wenger, "Trajectory planning of kinematically redundant parallel manipulators by using multiple working modes," *Mechanism and Machine*

Theory, vol. 98, pp. 216–230, April 2016. [Online]. Available: https://doi.org/10.1016%2Fj.mechmachtheory.2015.09.011

[49] J. Gao, A. Pashkevich, and S. Caro, "Optimization of the robot and positioner motion in a redundant fiber placement workcell," *Mechanism and Machine Theory*, vol. 114, pp. 170–189, August 2017. [Online]. Available: https://doi.org/10.1016%2Fj.mechmachtheory.2017.04.009

[50] D. Bussi, M. Barrera, R. Trucco, F. Salvioli, M. Rabaioli, E. Topa, A. D'Ottavio, L. Savioli, L. Ravagnolo, G. Martucci di Scarfizzi, P. Franceschetti, L. Joudrier, A. Williams, and T. Lim, "Challenges in the definition, validation and simulation of the ground operations of the ExoMars 2020 Rover surface mission at the Rover Operations Control Centre (ROCC)," in *69th International Astronautical Congress*, October 2018.

[51] PickNik Consulting. (2018) Moveit! web page. [Online]. Available: https://moveit.ros.org/

[52] Franka Emika. (2018) Panda web page. [Online]. Available: https://www.franka.de/panda

[53] W. Khalil and J. Kleinfinger, "A new geometric notation for open and closed-loop robots," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 1986, pp. 1174–1179. [Online]. Available: https://doi.org/10.1109%2Frobot.1986.1087552

[54] Franka Emika. (2018) Franka Control Interface (FCI). [Online]. Available: https://frankaemika.github.io/docs/

[55] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[56] Y. Chen, S. Y.-P. Chien, and A. A. Desrochers, "General structure of time-optimal control of robotic manipulators moving along prescribed paths," *International Journal of Control*, vol. 56, no. 4, pp. 767–782, October 1992. [Online]. Available: https://doi.org/10.1080%2F00207179208934342

[57] J.-J. E. Slotine and H. S. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 118–124, 1989. [Online]. Available: https://doi.org/10.1109%2F70.88024

[58] J. Kim and E. A. Croft, "Online near time-optimal trajectory planning for industrial robots," *Robotics and Computer-Integrated Manufacturing*, vol. 58, pp. 158–171, August 2019. [Online]. Available: https://doi.org/10.1016%2Fj.rcim.2019.02.009

[59] I. T. Pietsch, O. Becker, M. Krefft, and J. Hesselbach, "Time-optimal trajectory planning for adaptive control of plane parallel robots," in *The Fourth International Conference on Control and Automation.* IEEE, 2003, pp. 639–643. [Online]. Available: https://doi.org/10.1109%2Ficca.2003.1229182

[60] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, April 1987. [Online]. Available: https://doi.org/10.1109%2Fjra.1987.1087090

[61] Y. Chen and A. A. Desrochers, "Structure of minimum-time control law for robotic manipulators with constrained paths," in *Proceedings - IEEE International Conference on Robotics and Automation.* IEEE Comput. Soc. Press, 1989, pp. 971–976. [Online]. Available: https://doi.org/10.1109%2Frobot.1989.100107

[62] S. Ma, "Time-optimal control of robotic manipulators with limit heat characteristics of the actuator," *Advanced Robotics*, vol. 16, no. 4, pp. 309–324, January 2002. [Online]. Available: https://doi.org/10.1163%2F15685530260174502

[63] S. Behzadipour and A. Khajepour, "Time-optimal trajectory planning in cable-based manipulators," *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 559–563, June 2006. [Online]. Available: https://doi.org/10.1109%2Ftro.2006.870663

[64] D. Constantinescu and E. A. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified

paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000. [Online]. Available: https://doi.org/10.1002/(sici)1097-4563(200005)17:5⟨233::aid-rob1⟩3.0.co;2-y

[65] G. Pardo-Castellote and R. H. Cannon, "Proximate time-optimal algorithm for on-line path parameterization and modification," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 1996, pp. 1539–1546. [Online]. Available: https://doi.org/10.1109%2Frobot.1996.506923

[66] A. Casalino, A. M. Zanchettin, and P. Rocco, "Online planning of optimal trajectories on assigned paths with dynamic constraints for robot manipulators," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, October 2016. [Online]. Available: https://doi.org/10.1109%2Firos.2016.7759168

[67] L. Žlajpah and B. Nemec, "Implementation of time-optimal path-tracking control on palletizing robots," in *Proceedings - IEEE International Symposium on Industrial Electronics*. IEEE, 1999, pp. 861–866. [Online]. Available: https://doi.org/10.1109%2Fisie.1999.798726

[68] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, December 2014. [Online]. Available: https://doi.org/10.1109%2Ftro.2014.2351113

[69] J. Dong and J. A. Stori, "A generalized time-optimal bidirectional scan algorithm for constrained feed-rate optimization," *Journal of Dynamic Systems, Measurement, and Control*, vol. 128, no. 2, pp. 379–390, 2006. [Online]. Available: https://doi.org/10.1115%2F1.2194078

[70] D. Kaserer, H. Gattringer, and A. Müller, "Time optimal motion planning and admittance control for cooperative grasping," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2216–2223, April 2020. [Online]. Available: https://doi.org/10.1109%2Flra.2020.2970644

[71] S. Singh and M. C. Leu, "Optimal trajectory generation for robotic manipulators using dynamic programming," *Journal of Dynamic Systems, Measurement, and Control*, vol. 109, no. 2, pp. 88–96, 1987. [Online]. Available: https://doi.org/10.1115%2F1.3143842

[72] A. J. Cahill, M. R. James, J. C. Kieffer, and D. Williamson, "Remarks on the application of dynamic programming to the optimal path timing of robot manipulators," *International Journal of Robust and Nonlinear Control*, vol. 8, no. 6, pp. 463–482, May 1998. [Online]. Available: https://doi.org/10.1002/(sici)1099-1239(199805)8:6⟨463::aid-rnc312⟩3.0.co;2-r

[73] Y. Nakamura and K. Yamane, "Dynamics computation of structure-varying kinematic chains and its application to human figures," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 2, pp. 124–134, April 2000. [Online]. Available: https://doi.org/10.1109%2F70.843167

[74] Q.-C. Pham and O. Stasse, "Time-optimal path parameterization for redundantly actuated robots: a numerical integration approach," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 6, pp. 3257–3263, December 2015. [Online]. Available: https://doi.org/10.1109%2Ftmech.2015.2409479

[75] L. Žlajpah, "On time optimal path control of manipulators with bounded joint velocities and torques," in *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 1996, pp. 1572–1577. [Online]. Available: https://doi.org/10.1109%2Frobot.1996.506928

[76] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, June 1985. [Online]. Available: https://doi.org/10.1109%2Ftac.1985.1104009

[77] H. Pham and Q.-C. Pham, "On the structure of the time-optimal path parameterization problem with third-order constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2017. [Online]. Available: https://doi.org/10.1109%2Ficra.2017.7989084

[78] Z. Shiller and H.-H. Lu, "Computation of path constrained time optimal motions with dynamic singularities," *Journal of Dynamic Systems, Measurement, and Control*, vol. 114, no. 1, pp. 34–40, 1992. [Online]. Available: https://doi.org/10.1115%2F1.2896505

[79] K. Hauser, "Fast interpolation and time-optimization with contact," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1231–1250, August 2014. [Online]. Available: https://doi.org/10.1177%2F0278364914527855

[80] A. Obradović, J. Vuković, N. Mladenović, and Z. Mitrović, "Time optimal motions of mechanical system with a prescribed trajectory," *Meccanica*, vol. 46, no. 4, pp. 803–816, August 2010. [Online]. Available: https://doi.org/10.1007%2Fs11012-010-9339-3

[81] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, September 1985. [Online]. Available: https://doi.org/10.1177%2F027836498500400301

[82] Z. Shiller, "On singular time-optimal control along specified paths," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, pp. 561–566, 1994. [Online]. Available: https://doi.org/10.1109%2F70.313107

[83] M. Galicki and I. Pajak, "Optimal motions of redundant manipulators with state equality constraints," in *Proceedings - IEEE International Symposium on Assembly and Task Planning (ISATP'99)*. IEEE, July 1999, pp. 181–185. [Online]. Available: https://doi.org/10.1109%2Fisatp.1999.782956

[84] S.-J. Kim, D.-S. Choi, and I.-J. Ha, "A comparison principle for state-constrained differential inequalities and its application to time-optimal control," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 967–983, July 2005. [Online]. Available: https://doi.org/10.1109%2Ftac.2005.851434

[85] M. Tarkiainen and Z. Shiller, "Time optimal motions of manipulators with actuator dynamics," in *Proceedings* -

IEEE International Conference on Robotics and Automation. IEEE Comput. Soc. Press, 1993. [Online]. Available: https://doi.org/10.1109%2Frobot.1993.291873

[86] K. Shin and N. McKay, "Robust trajectory planning for robotic manipulators under payload uncertainties," *IEEE Transactions on Automatic Control*, vol. 32, no. 12, pp. 1044–1054, December 1987. [Online]. Available: https://doi.org/10.1109%2Ftac.1987.1104523

[87] H. K. Khalil, *Nonlinear Systems*. Prentice Hall, 1996.

[88] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, "Time-optimal path tracking for robots: a convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, October 2009. [Online]. Available: https://doi.org/10.1109%2Ftac.2009.2028959

[89] Y.-K. Choi, J.-H. Park, H.-S. Kim, and J. H. Kim, "Optimal trajectory planning and sliding mode control for robots using evolution strategy," *Robotica*, vol. 18, no. 4, pp. 423–428, July 2000. [Online]. Available: https://doi.org/10.1017%2Fs0263574799002118

[90] B. Craenen, A. Eiben, and E. Marchiori, "How to handle constraints with evolutionary algorithms," in *The Practical Handbook of Genetic Algorithms*. Chapman and Hall/CRC, December 2000. [Online]. Available: https://doi.org/10.1201%2F9781420035568.ch10

[91] K. Shin and N. McKay, "A dynamic programming approach to trajectory planning of robotic manipulators," *IEEE Transactions on Automatic Control*, vol. 31, no. 6, pp. 491–500, June 1986. [Online]. Available: https://doi.org/10.1109%2Ftac.1986.1104317

[92] K. M. Burjorjee, "Generative fixation: a unifed explanation for the adaptive capacity of simple recombinative genetic algorithms," PhD Thesis, Brandeis University, August 2009.

[93] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA, 1996.

[94] G. Zinni, "Analysis, design and implementation of a time-optimal planner for robotic applications," Master's thesis, Politecnico di Torino, 2019.

[95] O. Dahl and L. Nielsen, "Torque-limited path following by online trajectory time scaling," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 5, pp. 554–561, October 1990. [Online]. Available: https://doi.org/10.1109%2F70.62044

[96] Z. Shiller and H. Chang, "Trajectory preshaping for high-speed articulated systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 117, no. 3, pp. 304–310, September 1995. [Online]. Available: https://doi.org/10.1115%2F1.2799120

[97] Z. Shiller, H. Chang, and V. Wong, "The practical implementation of time-optimal control for robotic manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 12, no. 1, pp. 29–39, March 1996. [Online]. Available: https://doi.org/10.1016%2F0736-5845%2895%2900026-7

[98] J. Kieffer, A. Cahill, and M. James, "Robust and accurate time-optimal path-tracking control for robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 880–890, December 1997. [Online]. Available: https://doi.org/10.1109%2F70.650167

[99] O. Dahl, "Path-constrained robot control with limited torques - experimental evaluation," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 658–669, October 1994. [Online]. Available: https://doi.org/10.1109%2F70.326570

[100] J. Moreno-Valenzuela, "Tracking control of on-line time-scaled trajectories for robot manipulators under constrained torques," in *Proceedings - 2006 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 19–24. [Online]. Available: https://doi.org/10.1109%2Frobot.2006.1641155

[101] K. S. Eom, I. H. Suh, and W. K. Chung, "Disturbance observer based path tracking control of robot manipulator considering torque saturation," *Mechatronics*, vol. 11, no. 3, pp. 325–343, apr 2001. [Online]. Available: https://doi.org/10.1016%2Fs0957-4158%2800%2900021-0

[102] W. Niu and M. Tomizuka, "A new approach of coordinated motion control subjected to actuator saturation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 123, no. 3, pp. 496–504, July 1999. [Online]. Available: https://doi.org/10.1115%2F1.1387247

[103] H. Tam, "Minimum time closed-loop tracking of a specified path by robot," in *29th IEEE Conference on Decision and Control*. IEEE, December 1990, pp. 3132–3137. [Online]. Available: https://doi.org/10.1109%2Fcdc.1990.203368

[104] M. Galicki, "Time-optimal controls of kinematically redundant manipulators with geometric constraints," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 89–93, 2000. [Online]. Available: https://doi.org/10.1109%2F70.833194

[105] J. McCarthy and J. Bobrow, "The number of saturated actuators and constraint forces during time-optimal movement of a general robotic system," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 407–409, June 1992. [Online]. Available: https://doi.org/10.1109%2F70.143358

[106] E. Ferrentino and P. Chiacchio, "Redundancy parametrization in globally-optimal inverse kinematics," in *Advances in Robot Kinematics 2018*. Springer International Publishing, June 2018, pp. 47–55. [Online]. Available: https://doi.org/10.1007%2F978-3-319-93188-3_6

[107] S. Ma and M. Watanabe, "Time optimal path-tracking control of kinematically redundant manipulators," *JSME International Journal Series C*, vol. 47, no. 2, pp. 582–590, 2004. [Online]. Available: https://doi.org/10.1299%2Fjsmec.47.582

[108] P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano, "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy," *The International Journal of Robotics Research*, vol. 10, no. 4, pp. 410–425, August 1991. [Online]. Available: https://doi.org/10.1177%2F027836499101000409

[109] C. Gaz, M. Cognetti, A. Oliva, P. Robuffo Giordano, and A. De Luca, "Dynamic identification of the Franka Emika Panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4147–4154, October 2019. [Online]. Available: https://doi.org/10.1109%2Flra.2019.2931248

[110] Real-time linux wiki. [Online]. Available: https://rt.wiki.kernel.org/

[111] Open Source Robotics Foundation. (2020) Ros control wiki page. [Online]. Available: http://wiki.ros.org/ros_control

[112] E. Ferrentino and H. Judiss Savino. (2020) Execution of time-optimal trajectories with Franka Emika's Panda robot. [Online]. Available: https://www.youtube.com/watch?v=9xStJSPJ3bM

[113] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998, iowa State University TR 98-11.

[114] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014. [Online]. Available: https://doi.org/10.1109%2Faccess.2014.2302442

[115] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, jun 2011. [Online]. Available: https://doi.org/10.1177%2F0278364911406761

[116] J. H. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the rrt*," in *IEEE Conference on Decision and Control and European*

Control Conference. IEEE, dec 2011. [Online]. Available: https://doi.org/10.1109%2Fcdc.2011.6161521

[117] M. Cefalo and G. Oriolo, "A general framework for task-constrained motion planning with moving obstacles," *Robotica*, vol. 37, no. 3, pp. 575–598, oct 2018. [Online]. Available: https://doi.org/10.1017%2Fs0263574718001182

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005),
Fondo Sociale Europeo, Azione I.1 “Dottorati Innovativi con caratterizzazione Industriale”

UNIONE EUROPEA
Fondo Sociale Europeo