UNIVERSITAT POLITÈCNICA DE CATALUNYA

Doctoral Programme

AUTOMATIC CONTROL, ROBOTICS AND COMPUTER VISION

Ph.D. Thesis

KINODYNAMIC PLANNING AND CONTROL OF CLOSED-CHAIN ROBOTIC SYSTEMS

Ricard Bordalba Llaberia

Advisors:
Lluís Ros and Josep M Porta

May 2022

Kinodynamic Planning and Control of Closed-chain Robotic Systems

by Ricard Bordalba Llaberia

A thesis submitted to the Universitat Politècnica de Catalunya
for the degree of Doctor of Philosophy

Doctoral Programme:
Automatic Control, Robotics and Computer Vision

This thesis has been completed at:
Institut de Robòtica i Informàtica Industrial, CSIC-UPC

Advisors:
Lluís Ros and Josep M Porta

Dissertation Committee:
Prof. Pablo González de Santos
Prof. Federico Thomas Arroyo
Prof. Steven LaValle

KINODYNAMIC PLANNING AND CONTROL
OF CLOSED-CHAIN ROBOTIC SYSTEMS

Ricard Bordalba Llaberia

Abstract

This work proposes a methodology for kinodynamic planning and trajectory control in robots with closed kinematic chains. The ability to plan trajectories is key in a robotic system, as it provides a means to convert high-level task commands—like “move to that location”, or “throw the object at such a speed”—into low-level controls to be followed by the actuators. In contrast to purely kinematic planners, which only generate collision-free paths in configuration space, kinodynamic planners compute state-space trajectories that also account for the dynamics and force limits of the robot. In doing so, the resulting motions are more realistic and exploit gravity, inertia, and centripetal forces to the benefit of the task. Existing kinodynamic planners are fairly general and can deal with complex problems, but they require the state coordinates to be independent. Therefore, they are hard to apply to robots with loop-closure constraints whose state space is not globally parameterizable. These constraints define a nonlinear manifold on which the trajectories must be confined, and they appear in many systems, like parallel robots, cooperative arms manipulating an object, or systems that keep multiple contacts with the environment. In this work, we propose three steps to generate optimal trajectories for such systems. In a first step, we determine a trajectory that avoids the collisions with obstacles and satisfies all kinodynamic constraints of the robot, including loop-closure constraints, the equations of motion, or any limits on the velocities or on the motor and constraint forces. This is achieved with a sampling-based planner that constructs local charts of the state space numerically, and with an efficient steering method based on linear quadratic regulators. In a second step, the trajectory is optimized according to a cost function of interest. To this end we introduce two new collocation methods for trajectory optimization. While current methods easily violate the kinematic constraints, those we propose satisfy these constraints along the obtained trajectories. During the execution of a task, however, the trajectory may be affected by unforeseen disturbances or model errors. That is why, in a third step, we propose two trajectory control methods for closed-chain robots. The first method enjoys global stability, but it can only control trajectories that avoid forward singularities. The second method, in contrast, has local stability, but allows these singularities to be traversed robustly. The combination of these three steps expands the range of systems in which motion planning can be successfully applied.

KINODYNAMIC PLANNING AND CONTROL OF CLOSED-CHAIN ROBOTIC SYSTEMS

Ricard Bordalba Llaberia

Resum

Aquest treball proposa una metodologia per a la planificació cinetodinàmica i el control de trajectòries en robots amb cadenes cinemàtiques tancades. La capacitat de planificar trajectòries és clau en un robot, ja que permet traduir instruccions d'alt nivell—com ara "mou-te cap aquella posició" o "llença l'objecte amb aquesta velocitat"—en senyals de referència que puguin ser seguits pels actuadors. En comparació amb els planificadors purament cinemàtics, que només generen camins lliures de col·lisions a l'espai de configuracions, els planificadors cinetodinàmics obtenen trajectòries a l'espai d'estats que són compatibles amb les restriccions dinàmiques i els límits de força del robot. Els moviments que en resulten són més realistes i aprofiten la gravetat, la inèrcia i les forces centrípetes en benefici de la tasca que es vol realitzar. Els planificadors cinetodinàmics actuals són força generals i poden resoldre problemes complexos, però assumeixen que les coordenades d'estat són independents. Per tant, no es poden aplicar a robots amb restriccions de clausura cinemàtica en els quals l'espai d'estats no admeti una parametrització global. Aquestes restriccions defineixen una varietat diferencial sobre la qual cal mantenir les trajectòries, i apareixen en sistemes com ara els robots paral·lels, els braços que manipulen objectes coordinadament o els sistemes amb extremitats en contacte amb l'entorn. En aquest treball, proposem tres passos per generar trajectòries òptimes per a aquests sistemes. En un primer pas, determinem una trajectòria que evita les col·lisions amb els obstacles i satisfà totes les restriccions cinetodinàmiques, incloses les de clausura cinemàtica, les equacions del moviment o els límits en les velocitats i en les forces d'actuació o d'enllaç. Això s'aconsegueix mitjançant un planificador basat en mostratge aleatori que utilitza cartes locals construïdes numèricament, i amb un mètode eficient de navegació local basat en reguladors quadràtics lineals. En un segon pas, la trajectòria s'optimitza segons una funció de cost donada. A tal efecte, introduïm dos nous mètodes de col·locació per a l'optimització de trajectòries. Mentre els mètodes existents violen fàcilment les restriccions cinemàtiques, els que proposem satisfan aquestes restriccions al llarg de les trajectòries obtingudes. Durant l'execució de la tasca, tanmateix, la trajectòria pot veure's afectada per pertorbacions imprevistes o per errors deguts a incerteses en el model dinàmic. És per això que, en un tercer pas, proposem dos mètodes de control de trajectòries per robots amb cadenes tancades. El primer mètode gaudeix d'estabilitat global, però només permet controlar trajectòries que no travessin singularitats directes del robot. El segon mètode, en canvi, té estabilitat local, però permet travessar aquestes singularitats de manera robusta. La combinació d'aquests tres passos amplia el ventall de sistemes en els quals es pot aplicar amb èxit la planificació cinetodinàmica.

Acknowledgements

I would like to thank my supervisors, Lluís Ros and Josep M Porta, for their valuable guidance and support during these years. I wouldn't have made it this far if it hadn't been for their constant dedication, and for always having their door open for questions and discussions.

I am grateful and indebted to Moritz Diehl (University of Freiburg), who welcomed me to work with his group during several months in what became a very fruitful research stay.

I also wish to express my gratitude to the members of the dissertation committee for the time they have devoted to reading the thesis, and for their encouraging comments on the research reported.

I very much appreciate the great group of colleagues and friends that I met at IRI, who were part of countless lunches, board game sessions and football matches, which made my office days more enjoyable.

Finally, I am also thankful to my family, for the encouragement that I received during my work, and specially to Alba, who is always on my side.

This work has been partially supported by an FPI grant (BES-2015-074238) of the Spanish Ministry of Economy, Industry and Competitivity under the research projects:

- **DPI2017-88282-P** KINODYN: Kinodynamic planning of efficient and agile robot motions.
- **DPI2014-57220-C2-2-P** RobCab: Control strategies for cable-driven robot for low-gravity simulation.

Contents

1 Introduction

"Dynamics opens a world of opportunity for robotics.
Robots that move dynamically can go where other robots can't go,
handle larger payloads with smaller footprint and smaller robot mass,
and move faster to get work done more quickly".
— Marc Raibert, CEO of Boston Dynamics

1.1 Motivation

Since its formalization in the early nineties by Donald et al. [1] the kinodynamic planning problem remains as one of the most challenging open problems in robotics. The problem entails finding feasible trajectories connecting two given states of a robot, each defined by a configuration and a velocity of the underlying mechanical structure. To ensure feasibility, the trajectory should fulfill all kinematic constraints of the system (like loop-closure or velocity constraints), satisfy the equation of motion, avoid the collisions with obstacles, and be controllable with the limited force of the actuators. In many contexts, moreover, the trajectory should also be optimal in some sense, minimizing the time, energy, or control effort required for its execution for example. All of these constraints and cost functions are relevant in many factory and home environments in which Robotics is called to play a role in the near future.

The ability to plan trajectories is key in a robotic system. Above all, it endows the system with a means to convert high-level commands—like "move to a certain location", or "throw the object at a given speed"—into appropriate reference signals to be followed by the actuators. By accounting for the robot dynamics and force limits at the planning stage, moreover, the motions

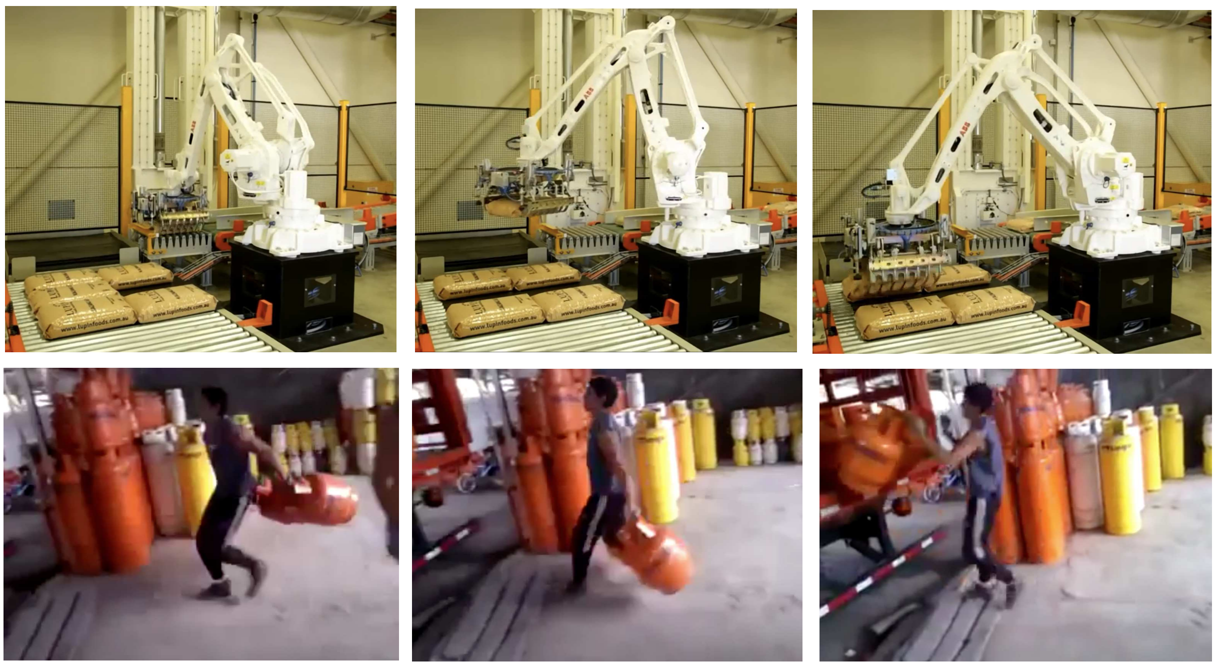


Figure 1.1: Top: A robot moving a load from one conveyor to another using a straight-to-the-goal motion. Bottom: A man loading two heavy gas bottles on a truck. While the robot trajectory is suitable for the shown payload, the same trajectory may be unusable to lift heavier loads, or may require a more powerful robot. Note how the man resorts to dynamics instead, to make optimal use of his limited strength through a swinging motion.

are easier to control, and they often look more graceful, or physically natural [2], as they tend to exploit gravity, inertia, and centripetal forces to the benefit of the task (Fig. 1.1).

The kinodynamic planning problem can be viewed as a full motion planning problem in the state space, as opposed to a purely kinematic problem that only requires the planning of a path in configuration space (C-space). This makes the problem harder, as the dimension of the state space is twice that of the C-space, and the obstacle region is virtually larger, involving states that correspond to an actual collision, but also those from which a future collision is inevitable due to the system momentum. The planning of steering motions is considerably more difficult as well. While direct motions suffice in the C-space, steering motions in the state space need to conform to the vector fields defined by the equations of motion and force limits of the robot.

Despite the previous difficulties, existing kinodynamic planners are fairly general and can deal with relatively complex problems. The vast majority of such planners, however, have an important limitation: they assume that the state space is globally parametrizable, i.e., that the states can be represented by a unique set of *independent* coordinates that are valid over the whole motion range. While parametric state spaces arise frequently, for example in robots with

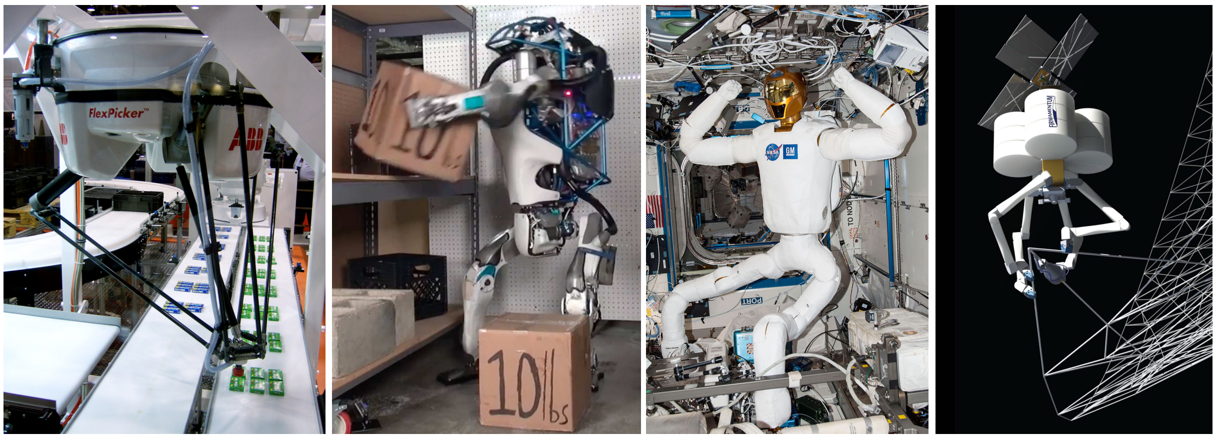


Figure 1.2: Example systems involving closed kinematic chains. The chains may be intrinsic to the robot structure, as in parallel robots (left picture), or they may result from manipulation constraints during a task, as in multi-limbed systems transporting an object, or keeping their feet attached to the environment (right pictures). From left to right: A Delta parallel robot [3], the Atlas robot from Boston Dynamics lifting a heavy load[4], the Robonaut 2 robot with two legs clamped to the International Space Station [5], and SpiderFab Bot, a conceptual design for self-fabricating space systems [6]. Pictures courtesy of ABB, Boston Dynamics, NASA, and Tethers Unlimited, Inc, respectively.

tree topology moving in free space, kinematic loop-closure constraints relating the state space coordinates in intricate ways may also appear. This occurs in many robotic systems, including parallel manipulators, robot arms manipulating an object, or in multi-limbed systems keeping contacts with the environment (Fig. 1.2). In these situations the system is said to be closed-chain, and its state space becomes a nonlinear manifold defined implicitly by the loop-closure constraints. This manifold is a zero-measure set in a larger ambient space, which complicates the design of a motion planner able to explore the manifold efficiently. Moreover, if the dynamic model of the robot is not properly handled, the obtained trajectories may deviate substantially from the manifold, leading to unrealistic trajectories, or to a failure to reach the goal. Even if the trajectory is kept on the manifold, the design of a robust controller able track the trajectory is also problematic. Closed kinematic chains exhibit so-called forward singularities, which make traditional computed-torque controllers fail in their vicinity, being unable to correct errors along certain directions, or producing large control efforts that can harm the robot structure. It is probably for all these reasons that, to date, mature algorithms for kinodynamic planning and control have not been developed for closed kinematic chains. Our purpose in this work is to help filling this gap to the largest possible extent.

1.2 Objectives

The goal of this thesis is to provide reliable algorithms to solve the following two problems for general closed-chain robotic systems:

- **Kinodynamic motion planning:** Given the kinematic and dynamic models of the robot, and a geometric model of its environment, find a trajectory that is both feasible and optimal to bring the robot from a start to a goal state. By feasible we mean that the trajectory must avoid the collisions with obstacles while satisfying all kinematic and dynamic constraints of the robot, including loop-closure constraints, the equations of motion, or any limits on the velocities or on the motor or constraint forces. By optimal we mean that the trajectory should minimize, at least locally, a given cost function of interest.

- **Tracking control:** Given a solution trajectory for the earlier problem, find a feedback controller that is able to track the trajectory in the presence of unmodeled disturbances or model errors, even across forward singularities of the robot.

Although the current knowledge in motion planning and control may allow solving these problems in particular systems (for example by exploiting explicit state-space parameterizations when they exist) our goal is to find a general solution for the entire class of systems we consider.

1.3 Assumptions and Scope

For the purpose of this work, a robotic system is a multibody system composed of rigid bodies and lower-pair joints, where some of the joints are actuated. We restrict our attention to closed-chain systems, i.e., those that must fulfill a number of loop-closure constraints modeling cyclic sequences of bodies and joints that must remain closed during a task. Such constraints can be inherent to the robot structure, as in parallel robots, or may result from manipulation constraints needed to fulfill the task, as in multi-limbed systems transporting an object, or keeping they feet attached to the environment (Fig. 1.2). In all cases we shall consider the constraints to be permanent, as opposed to intermittent constraints that arise, for example, when the robot makes or breaks contact with the ground. Robots subject to impact dynamics will also be excluded from our study.

While other holonomic constraints could also be handled by our framework, we concentrate on loop-closure constraints due to the growing interest they arouse [7–12]. We also leave nonholonomic constraints out of our scope, although, as detailed in Chapter 6, they could be accommodated in most of our methods with small modifications.

Our main focus is on robots with, at least, as many actuators as the number of degrees of freedom to be controlled. In most cases the actuator forces are limited to a prescribed range, and limits may also be imposed in the internal constraint forces if required. While the former account for limited motor capacities, the latter ensure the resistance and smooth functioning of the robot parts.

Our entire approach is model-based. We assume that proper models of the robot and its environment are available. This implies that the robot dimensions and dynamic parameters, as well as the geometry and location of all obstacles, will be known with sufficient accuracy. Thus, the problems of system identification, calibration, or environment modeling are out of the scope of this work.

The goal of our planning algorithms will be to compute the actions required to bring the robot from a start to a goal state with minimum cost. The output of the algorithms will be a time history of such actions and the corresponding trajectory in state space. The obtained trajectory has to respect all kinematic and dynamic constraints imposed by the problem.

The goal of our controllers will be to stably track the trajectory in the presence of unmodeled perturbations or model errors. The controllers will be closed loop, i.e., they will consist of a feedback law providing, for each time and state in a neighborhood of the trajectory, appropriate corrective actions to get the robot in synchrony with the trajectory. Obstacles, motor torque limits, and constraint force limits will not be handled by the controllers. Instead, we assume these constraints to be enforced at the planning stage, leaving sufficient clearance or force margins so as to be able to reject unmodeled disturbances or model errors during trajectory tracking.

1.4 Outlook at the Dissertation

1.4.1 Approach

Sampling-based planners are among the most popular methods for solving the general motion planning problem [13]. A strong point of these planners is they do not construct explicit representations of the obstacle region. Instead, their search is conducted by probing the state space with a sampling scheme, which leads to highly-efficient resolution- or probabilistically-complete algorithms. The simultaneous treatment of loop-closure and dynamic constraints, however, has remained open for these methods [12], and the trajectories they obtain tend to be jerky or far-from-optimal in many cases. With the aim of obtaining optimal trajectories, a recent family of planners directly resort to trajectory optimization methods [14–16]. These methods are powerful, but their convergence can only be ensured if they depart from good approximations of the solution, or if the problem is sufficiently relaxed so as to afford poorer approximations.

The planners in [14–16] opt for the latter and relax the obstacle-avoidance and loop-closure constraints by adding them as penalty terms in the cost function, so these constraints may not be satisfied exactly in their computed trajectories. The equations of motion and motor force limits are also neglected, and only velocity or acceleration penalty terms are added to favor the emergence of smooth motions.

In this Thesis we try to combine the benefits of sampling-based and optimization approaches while avoiding their drawbacks. To obtain optimal solutions without relaxing the kinodynamic constraints, we solve the kinodynamic planning problem by applying two modules in sequence:

- A sampling-based **trajectory planner** that searches for a collision-free trajectory connecting the start and goal states while satisfying the kinodynamic constraints of the robot. This planner does not obtain an optimal solution in general, but if finds a feasible one if it exists and enough computing time is available.

- A **trajectory optimizer** that locally improves the trajectory returned by the planner until it minimizes a cost function of interest. This optimizer enforces the same kinodynamic constraints as the planner, and admits a large variety of cost functions to optimize, for example, the time, energy, or control effort employed during the trajectory, or the smoothness of the control actions. Since the optimizer departs from a feasible trajectory, its convergence to a minimum-cost solution is greatly facilitated.

To construct these two modules we have extended state-of-the-art methods for randomized kinodynamic planning and trajectory optimization to be able to deal with closed kinematic chains. When used in conjunction, the two modules provide a probabilistically-complete kinodynamic planner for closed-chain robotic systems.

Once a trajectory has been obtained with the earlier modules, two strategies are proposed to control its tracking. On the one hand, it is well known that computed-torque methods generate feedback laws with global basins of attraction towards the desired trajectory [17], which makes them very attractive when large perturbations need to be counteracted during the motion. However, we show that these controllers do not behave well near forward singularities, so to be able to employ them we have extended our kinodynamic planning algorithms to compute singularity-free motions. On the other hand, restricting the motions to the singularity-free C-space implies a loss of motion capabilities in general, so we have designed an additional controller that, despite it shows local convergence only, allows a stable traversal of forward singularities even in the presence of unforeseen perturbations or model errors. To obtain this controller we have extended the theory of linear quadratic regulators to deal with closed kinematic chains.

1.4.2 Organization

The rest of the thesis is organized as follows:

Chapter 2 gives a detailed description of the formulation required to model closed-chain robotic systems. Both the kinematic and dynamic equations of these systems are presented, together with the main motion spaces and singularities they define. Existing dynamics algorithms to obtain the equations of motion using spatial vector algebra are also summarized.

Chapter 3 develops the trajectory planner we propose. Unlike previous sampling-based planners, ours can satisfy all kinematic and dynamic constraints simultaneously. The planner does not produce optimal solutions, but it is probabilistically complete in its fully randomized version, so it always returns a solution if one exists and enough computing time is available. The chapter also shows how the planner can be used to obtain singularity-free motions, and demonstrates its performance in several test cases.

Chapter 4 presents two new methods for trajectory optimization that locally minimize a general class of integral cost functions. While existing methods tend to produce kinematic errors in closed-chain systems, those we propose eliminate these errors along the discrete trajectory or even the continuous one depending on the method. The chapter can be seen as an extension of direct collocation methods to deal with closed kinematic chains.

Chapter 5 reviews computed-torque methods and shows how they can be adapted to closed-chain robotic systems. It then explains how forward singularities seriously affect such controllers and proposes a new control method that is immune to such singularities.

Chapter 6 finally summarizes the main contributions of this work and highlights points that deserve further attention.

2
Closed-chain Systems

This chapter provides preliminary background for the rest of the thesis. We start by describing three spaces that are necessary to model the motions of a closed-chain system: the C-space, the state space, and the acceleration space. While path planning typically operates in the C-space, our kinodynamic planning and control problems are naturally solved in the state space, so an understanding of this space is necessary to construct our algorithms. The acceleration space, in turn, is useful to deal with the special geometry of accelerations in closed-chain systems. We then describe C-space and forward singularities, which are critical configurations that are problematic for our purposes. While the former generate nonsmoothnesses in the configuration and state spaces, the latter complicate the control of the robot substantially. The equations of motion of a closed-chain robot are then presented along with the solutions to the forward and inverse dynamics. The chapter finally includes a summary of recursive dynamics algorithms, which are efficient tools to compute the terms of the dynamic equations.

2.1 Kinematic Spaces

2.1.1 The Configuration Space

Let us describe our robot configuration by means of a tuple $\boldsymbol{q}$ of n_q generalized coordinates that determine the position and orientation of all links at a given instant of time. There is much freedom in choosing the form of $\boldsymbol{q}$ (for example one can resort to minimal, reference point, or natural coordinates [18]), but we will here assume that $\boldsymbol{q}$ contains, among other coordinates, the actuated coordinates of the robot. We restrict our attention to robots with closed kinematic chains, in which $\boldsymbol{q}$ must satisfy a system of n_e nonlinear equations

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0} \tag{2.1}$$

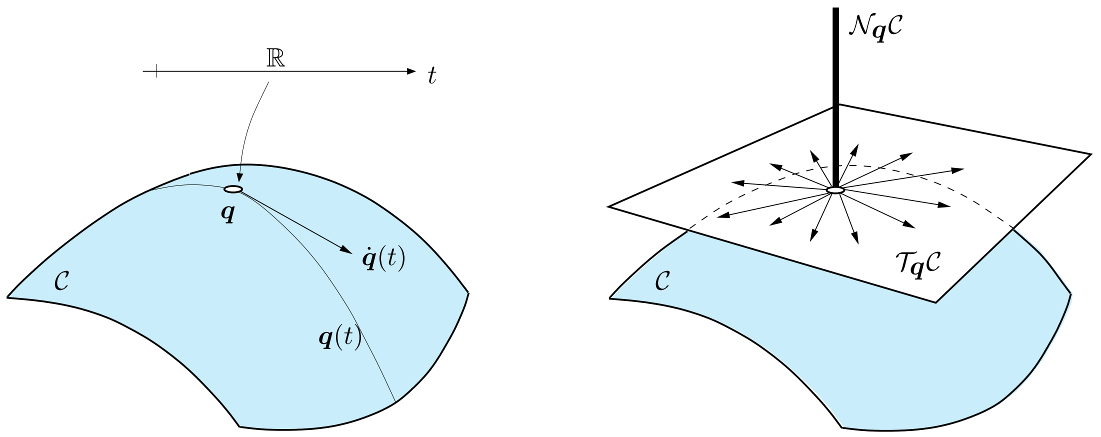

$\mathbb{R}$
t
$\boldsymbol{q}$
$\dot{\boldsymbol{q}}(t)$
$\mathcal{C}$
$\boldsymbol{q}(t)$
$\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$
$\mathcal{C}$
$\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$

Figure 2.1: The tangent and normal spaces to $\mathcal{C}$ at a point $\boldsymbol{q}$. When $\boldsymbol{\Phi_q}$ is full rank, $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ can be viewed as the set of velocity vectors $\dot{\boldsymbol{q}}(t)$ for all possible curves $\boldsymbol{q}(t)$ going through $\boldsymbol{q}$ for some t. The normal space $\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$ is the orthogonal complement of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ relative to $\mathbb{R}^{n_q}$.

enforcing the closure conditions of the chains. Equation (2.1) is holonomic, as it does not include the time derivative of $\boldsymbol{q}$, and so it is usually called the position constraint of the system. The configuration space of the robot, or C-space for short, is then the set

$$\mathcal{C} = \{\boldsymbol{q} : \boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0}\},$$

which may be quite complex in general. Around the points $\boldsymbol{q} \in \mathcal{C}$ in which the Jacobian $\boldsymbol{\Phi_q} = \partial\boldsymbol{\Phi}/\partial\boldsymbol{q}$ is full rank, however, $\mathcal{C}$ is a smooth manifold of dimension $d_{\mathcal{C}} = n_q - n_e$.

By differentiating Eq. (2.1) with respect to time, we obtain the velocity constraint

$$\boldsymbol{\Phi_q}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{0}, \tag{2.2}$$

which characterizes the set of feasible velocity vectors $\dot{\boldsymbol{q}}$ at $\boldsymbol{q} \in \mathcal{C}$:

$$\mathcal{T}_{\boldsymbol{q}}\mathcal{C} = \{\dot{\boldsymbol{q}} \in \mathbb{R}^{n_q} : \boldsymbol{\Phi_q}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{0}\}.$$

This set is also known as the tangent space of $\mathcal{C}$ at $\boldsymbol{q}$, and when $\boldsymbol{\Phi_q}(\boldsymbol{q})$ is full rank $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ is $d_{\mathcal{C}}$-dimensional and well-defined, so each $\dot{\boldsymbol{q}} \in \mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ corresponds to the time derivative of some curve $\boldsymbol{q}(t) \subset \mathcal{C}$ (Fig 2.1). In such a situation, one also defines the normal space to $\mathcal{C}$ at $\boldsymbol{q}$,

$$\mathcal{N}_{\boldsymbol{q}}\mathcal{C} = \{\boldsymbol{v} \in \mathbb{R}^{n_q} : \boldsymbol{v} = \boldsymbol{\Phi_q}(\boldsymbol{q})^{\top}\boldsymbol{\lambda} \quad \text{for some} \quad \boldsymbol{\lambda} \in \mathbb{R}^{n_e}\},$$

which contains all vectors $\boldsymbol{v}$ that are orthogonal to $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$. Note that, while $\mathcal{T}_{\boldsymbol{q}}\mathcal{C} = \mathrm{Ker}\,(\boldsymbol{\Phi_q})$, $\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$ coincides with the row space of $\boldsymbol{\Phi_q}$.

2.1.2 The State Space

Now let

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$$

denote the system formed by Eqs. (2.1) and (2.2), where

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{q}) \\ \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \dot{\boldsymbol{q}} \end{bmatrix},$$

$\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n_x}$ is the robot state, and $n_x = 2n_q$. The state space of the robot is then the set

$$\mathcal{X} = \{\boldsymbol{x} : \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}\}.$$

If $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})$ is full rank, the Jacobian

$$\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) & \boldsymbol{0} \\ \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) & \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) \end{bmatrix}$$

is also full rank at $\boldsymbol{x} \in \mathcal{X}$, so $\mathcal{X}$ is a smooth manifold in a neighborhood of $\boldsymbol{x}$, whose dimension is $d_{\mathcal{X}} = n_x - 2n_e$. This implies that the tangent space of $\mathcal{X}$ at $\boldsymbol{x}$,

$$\mathcal{T}_{\boldsymbol{x}}\mathcal{X} = \{\dot{\boldsymbol{x}} \in \mathbb{R}^{n_x} : \boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x})\, \dot{\boldsymbol{x}} = \boldsymbol{0}\},$$

is also well-defined and $d_{\mathcal{X}}$-dimensional, a property we often exploit in this thesis.

2.1.3 The Acceleration Space

For later developments it will also be necessary to consider the time derivative of Eq. (2.2),

$$\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} - \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{0}, \tag{2.3}$$

where $\boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}}$. This equation is called the acceleration constraint and we can use it to define

$$\mathcal{A}_{\boldsymbol{x}} = \{\ddot{\boldsymbol{q}} : \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} = \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}})\},$$

which provides the set of feasible accelerations of the robot at $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$.

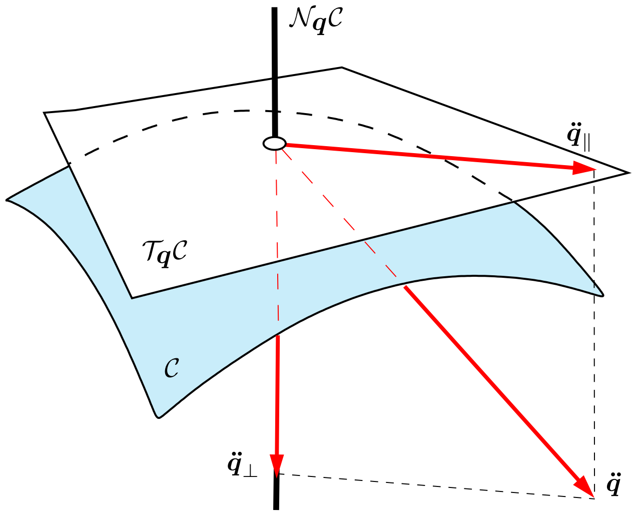

$\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$
$\ddot{\boldsymbol{q}}_{\|}$
$\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$
$\mathcal{C}$
$\ddot{\boldsymbol{q}}_{\perp}$
$\ddot{\boldsymbol{q}}$

Figure 2.2: Decomposition of $\ddot{\boldsymbol{q}}$ into its normal and parallel components. The motor actions can only modify $\ddot{\boldsymbol{q}}_{\|}$, as $\ddot{\boldsymbol{q}}_{\perp}$ is fully determined by the geometry of $\mathcal{C}$ and the value of the current state.

Note that, since $\mathcal{A}_{\boldsymbol{x}}$ is the solution of a linear system of equations, it must be an affine space with the same dimension as Ker $(\boldsymbol{\Phi}_{\boldsymbol{q}})$. In particular, any $\ddot{\boldsymbol{q}} \in \mathcal{A}_{\boldsymbol{x}}$ can be uniquely decomposed into a sum of the form

$$\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}_{\perp} + \ddot{\boldsymbol{q}}_{\|}$$

where $\ddot{\boldsymbol{q}}_{\perp}$ and $\ddot{\boldsymbol{q}}_{\|}$ are vectors of $\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$ and $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ respectively (Fig. 2.2). The component $\ddot{\boldsymbol{q}}_{\perp}$ is given by

$$\ddot{\boldsymbol{q}}_{\perp} = \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{+}\, \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}),$$

where $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{+}$ is the Moore-Penrose pseudoinverse of $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})$. The component $\ddot{\boldsymbol{q}}_{\|}$, in turn, can be expressed as

$$\ddot{\boldsymbol{q}}_{\|} = \boldsymbol{\Lambda}(\boldsymbol{q})\, \boldsymbol{\alpha},$$

where $\boldsymbol{\Lambda}(\boldsymbol{q})$ is an $n_q \times d_{\mathcal{C}}$ matrix that has, by columns, a basis of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$, and $\boldsymbol{\alpha}$ is a vector of $\mathbb{R}^{d_{\mathcal{C}}}$. Thus, the elements of $\mathcal{A}_{\boldsymbol{x}}$ can be expressed parametrically in terms of $\boldsymbol{\alpha}$,

$$\ddot{\boldsymbol{q}} = \underbrace{\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{+}\, \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}})}_{\ddot{\boldsymbol{q}}_{\perp}} + \underbrace{\boldsymbol{\Lambda}(\boldsymbol{q})\, \boldsymbol{\alpha}}_{\ddot{\boldsymbol{q}}_{\|}}, \tag{2.4}$$

which shows that $\mathcal{A}_{\boldsymbol{x}}$ is of dimension $d_{\mathcal{C}}$. From this equation we also see that $\ddot{\boldsymbol{q}}_{\perp}$ solely depends on the geometry of $\mathcal{C}$ and on the value of the current state, so $\ddot{\boldsymbol{q}}_{\|}$ is the only component of $\ddot{\boldsymbol{q}}$ that can be modified by the motor actions of the robot.

2.2 Singularities

Singularities are critical configurations in which the solution of the forward or the inverse instantaneous kinematic problems is undetermined. Depending on their nature, these configurations yield dexterity or velocity control losses, but they also have an impact on the robot dynamics. This section aims at briefly introducing the two types of singularities that are most relevant to our work. An exhaustive analysis and computation of all possible singularities is out of our scope, and we refer the reader to [19–21] for details.

2.2.1 C-space Singularities

The first type of singularity to be distinguished is the C-space singularity, which is a configuration in which the Jacobian $\mathbf{\Phi}_{\boldsymbol{q}}$ is rank deficient. In such a configuration $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ is ill defined, as some of its vectors $\dot{\boldsymbol{q}}$ do not correspond to the time derivative of any parametric curve $\boldsymbol{q}(t) \subset \mathcal{C}$. These singularities typically correspond to bifurcations, cusps, ridges, or dimension changes of $\mathcal{C}$ (Fig. 2.3). In this thesis, however, we shall assume that $\mathbf{\Phi}_{\boldsymbol{q}}$ is full rank for all $\boldsymbol{q} \in \mathcal{C}$ so $\mathcal{C}$, and thus $\mathcal{X}$, will both be smooth manifolds with well defined tangent spaces. This assumption is not too restrictive, as C-space singularities only arise for nongeneric robot dimensions and thus they can be removed by proper mechanical design. Figure 2.4 illustrates this point with an example. The shown robot presents a C-space singularity when all links are aligned [19], but the alignment can only occur for specific link lengths (left figure) so generic variations in such lengths will remove the singularity (right figure). Even so, if our robot presents C-space singularities we will still be able to plan its motions by actively avoiding such configurations (Section 3.8).

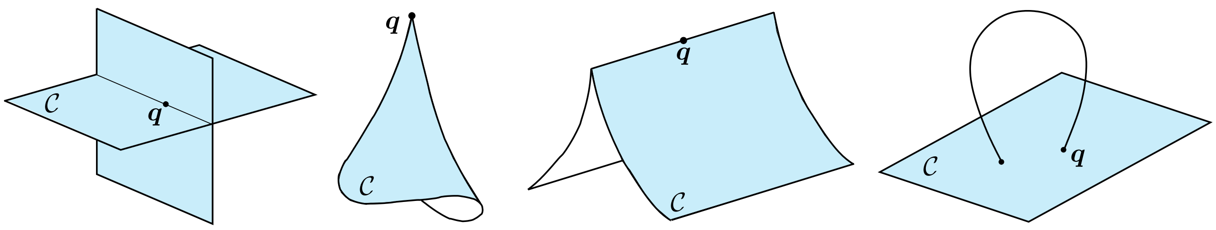


Figure 2.3: Examples of C-space singularities. They correspond to points $\boldsymbol{q}$ where $\mathcal{C}$ may lose the manifold structure, such as bifurcations, cusps, ridges, or dimension changes [19].

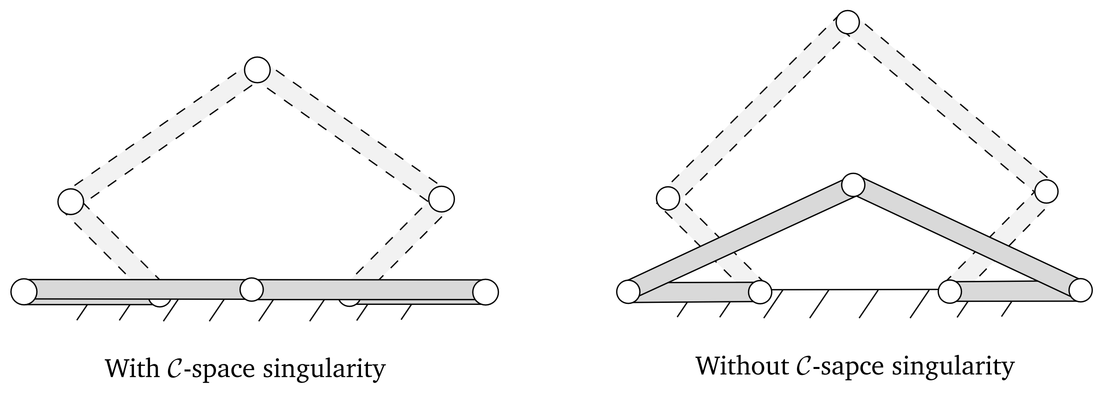

With $\mathcal{C}$-space singularity
Without $\mathcal{C}$-sapce singularity

Figure 2.4: Left: a planar five-bar robot presents a C-space singularity when the link lengths allow the alignment of all joints [19]. Right: a slight variation of the lengths removes the possibility of such an alignment.

2.2.2 Forward Singularities

The removal of C-space singularities does not rule out a second type of critical configuration called forward singularity, which depends on the choice of actuated coordinates. To define this singularity, let us decompose the $\boldsymbol{q}$ vector as follows

$$\boldsymbol{q} = \begin{bmatrix}\boldsymbol{Q}_u & \boldsymbol{Q}_r\end{bmatrix} \begin{bmatrix}\boldsymbol{q}_u \\ \boldsymbol{q}_r\end{bmatrix}, \tag{2.5}$$

where $\boldsymbol{q}_u$ and $\boldsymbol{q}_r$ respectively contain the n_u actuated coordinates and the $n_r = n_q - n_u$ remaining coordinates of $\boldsymbol{q}$, and $[\boldsymbol{Q}_u \quad \boldsymbol{Q}_r]$ is a permutation matrix used to allow an arbitrary ordering of $\boldsymbol{q}$. If we obtain the time derivative of Eq. (2.5) and insert it into Eq. (2.2), the velocity constraint then takes the form

$$\underbrace{\begin{bmatrix}\boldsymbol{\Phi}_{\boldsymbol{q}_u}(\boldsymbol{q}) & \boldsymbol{\Phi}_{\boldsymbol{q}_r}(\boldsymbol{q})\end{bmatrix}}_{\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})} \begin{bmatrix}\dot{\boldsymbol{q}}_u \\ \dot{\boldsymbol{q}}_r\end{bmatrix} = \boldsymbol{0}. \tag{2.6}$$

Suppose additionally that the forward instantaneous kinematic problem (FIKP) is defined as follows:

> Given a point $\boldsymbol{q} \in \mathcal{C}$, and some $\dot{\boldsymbol{q}}_u \in \mathbb{R}^{n_u}$, find all vectors of the form $\dot{\boldsymbol{q}} = (\dot{\boldsymbol{q}}_u, \dot{\boldsymbol{q}}_r)$ that satisfy Eq. (2.6), i.e., solve
>
> $$\boldsymbol{\Phi}_{\boldsymbol{q}_r}(\boldsymbol{q})\, \dot{\boldsymbol{q}}_r = -\boldsymbol{\Phi}_{\boldsymbol{q}_u}(\boldsymbol{q})\, \dot{\boldsymbol{q}}_u \tag{2.7}$$
>
> for $\dot{\boldsymbol{q}}_r$, assuming $\dot{\boldsymbol{q}}_u$ is a known value.

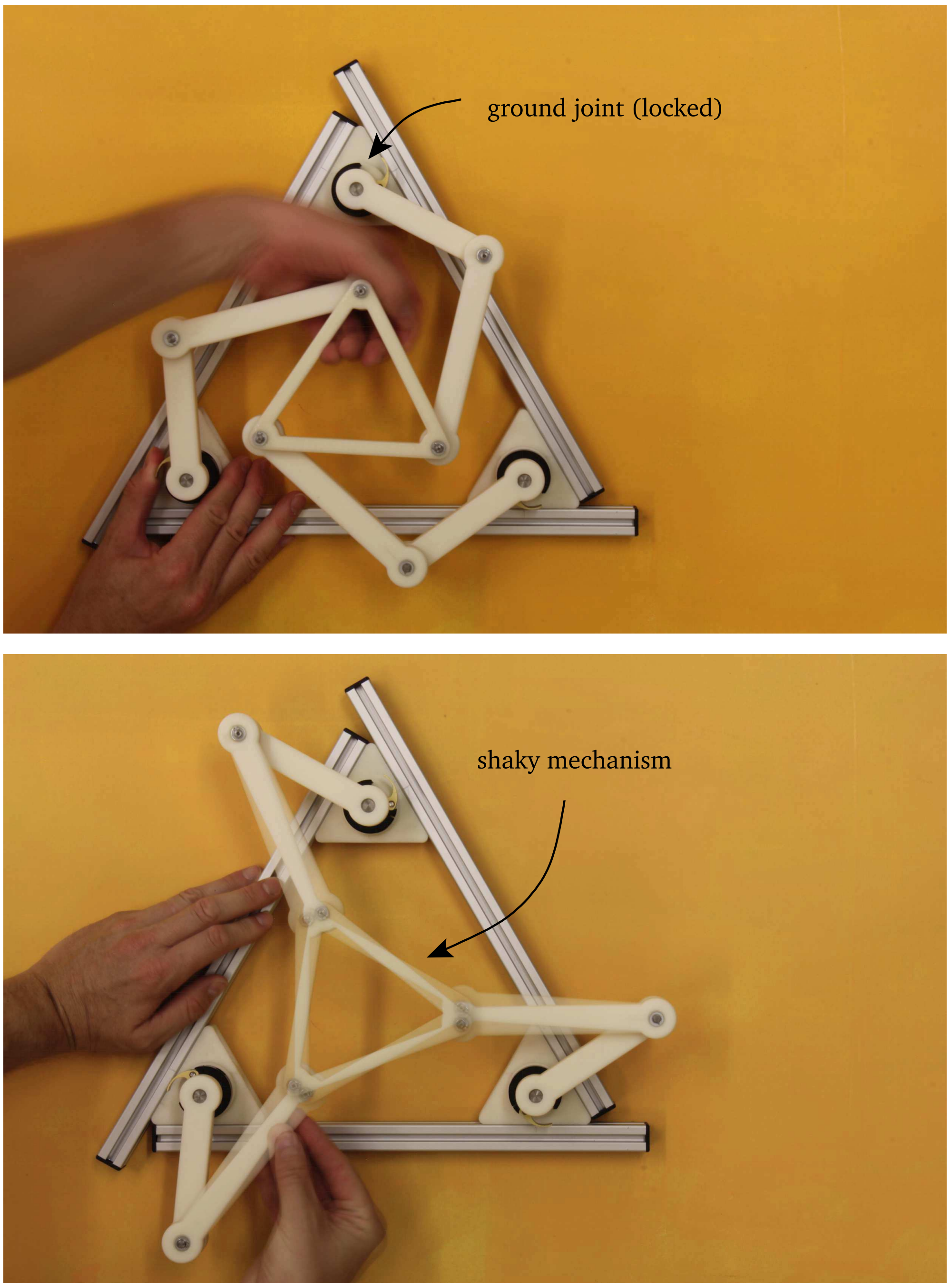


Figure 2.5: Forward singularity in a parallel 3-$\underline{R}$RR mechanism. Assuming that the ground joints are actuated, the 3-RRR mechanism exhibits a forward singularity when the lines of support of the three distal links are concurrent. Top: away from a forward singularity, the mechanism is rigid when the ground joints are locked, so the mechanism can counteract any force applied to its platform in principle. Bottom: In a forward singularity, instead, the mechanism is shaky after locking such joints. See https://youtu.be/xV3m6ioilnc for an animation.

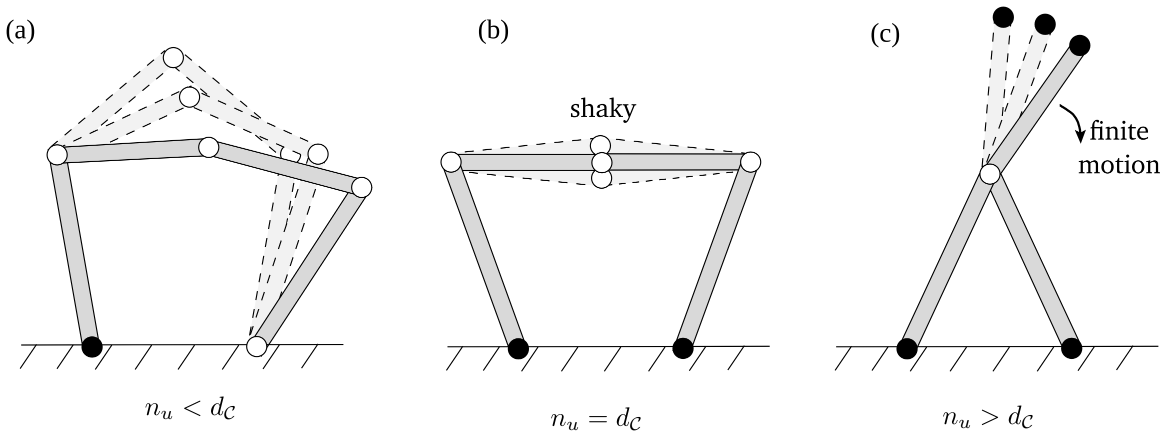


Figure 2.6: Examples of forward singularities in a five-bar robot, for the three cases $n_u < d_{\mathcal{C}}$, $n_u = d_{\mathcal{C}}$, and $n_u > d_{\mathcal{C}}$. The black circles indicate the actuated joints, which are assumed to be locked in all cases. In (a) $n_u < d_{\mathcal{C}}$, so the mechanism is not rigid, even away from the shown configuration. In (b) and (c) the mechanism is shaky, or even can move in (c), but away from such configurations it would be rigid.

Then, forward singularities are defined as the points $\boldsymbol{q} \in \mathcal{C}$ in which Eq. (2.7) has infinitely-many solutions for some $\dot{\boldsymbol{q}}_u \in \mathbb{R}^{n_u}$. In such points, thus, a feasible actuator velocity $\dot{\boldsymbol{q}}_u$ does not determine the robot velocity $\dot{\boldsymbol{q}}$, so we lose control (or observability) of $\dot{\boldsymbol{q}}$ by controlling (or observing) $\dot{\boldsymbol{q}}_u$.

From the theory of linear systems we know that Eq. (2.7) will have infinitely-many solutions for some $\dot{\boldsymbol{q}}_u \in \mathbb{R}^{n_u}$ if, and only if,

$$\text{rank}(\boldsymbol{\Phi}_{\boldsymbol{q}_r}) < n_r. \tag{2.8}$$

The fact that $\boldsymbol{\Phi}_{\boldsymbol{q}_r}$ loses rank at a forward singularity implies that the kernel of $\boldsymbol{\Phi}_{\boldsymbol{q}_r}$ is of dimension one or higher, so if we set $\dot{\boldsymbol{q}}_u = \boldsymbol{0}$ in Eq. (2.7), there will be infinitely-many values of $\dot{\boldsymbol{q}}_r$ satisfying the equation. Physically, this implies that the robot is shaky when we lock its actuators. Fig. 2.5 illustrates this phenomenon in a particular mechanism.

It is worth noting that a forward singularity is not necessarily a C-space singularity, as the rank deficiency of $\boldsymbol{\Phi}_{\boldsymbol{q}_r}$ does not imply the rank deficiency of $\boldsymbol{\Phi}_{\boldsymbol{q}}$. However, the converse is true: a C-space singularity is always a forward singularity.

While the condition in (2.8) allows us to detect whether a point $\boldsymbol{q} \in \mathcal{C}$ is a forward singularity, this condition can be checked in different ways depending on the shape of $\boldsymbol{\Phi}_{\boldsymbol{q}_r}$. This matrix is always of size $n_e \times n_r$, but it can be fat ($n_e < n_r$), square ($n_e = n_r$), or tall ($n_e > n_r$), depending on the number n_u of actuated coordinates. By noting that $n_e = n_q - d_{\mathcal{C}}$ and $n_r = n_q - n_u$, the following applies:

- If $n_u < d_{\mathcal{C}}$, $\mathbf{\Phi}_{\boldsymbol{q}_r}$ is fat and it has rank $\left(\mathbf{\Phi}_{\boldsymbol{q}_r}\right) < n_r$ irrespective of $\boldsymbol{q}$, so the robot is always in a forward singularity.
- If $n_u = d_{\mathcal{C}}$, $\mathbf{\Phi}_{\boldsymbol{q}_r}$ is square and $\boldsymbol{q} \in \mathcal{C}$ is a forward singularity if and only if
$$\det\left(\mathbf{\Phi}_{\boldsymbol{q}_r}(\boldsymbol{q})\right) = 0.$$
- If $n_u > d_{\mathcal{C}}$, $\mathbf{\Phi}_{\boldsymbol{q}_r}$ is tall and $\boldsymbol{q} \in \mathcal{C}$ is a forward singularity whenever
$$\det\left(\mathbf{\Phi}_{\boldsymbol{q}_r}(\boldsymbol{q})^\top \cdot \mathbf{\Phi}_{\boldsymbol{q}_r}(\boldsymbol{q})\right) = 0.$$

Fig. 2.6 provides an example of a singular configuration for each one of the earlier cases.

2.2.3 A Geometric Interpretation of Forward Singularities

Before closing this section, we must recall a geometric interpretation of forward singularities that will be used in Section 2.3.3 to discuss the solvability of the inverse dynamic problem. From the results in [22][Section 2.2.3] it can be shown that, when $\boldsymbol{q} \in \mathcal{C}$ is not a forward singularity, the projection of all vectors $\dot{\boldsymbol{q}} \in \mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ onto their $\boldsymbol{q}_u$ coordinates spans a vector space of dimension $d_{\mathcal{C}}$, whereas such a projection is lower-dimensional at a forward singularity. That is, if $\mathbf{\Lambda}$ is an $n_q \times d_{\mathcal{C}}$ matrix that has, by columns, a vector basis of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$, then

$$\text{rank}(\mathbf{\Lambda}^\top \boldsymbol{Q}_u) = d_{\mathcal{C}}$$

if and only if $\boldsymbol{q}$ is not a forward singularity, and

$$\text{rank}(\mathbf{\Lambda}^\top \boldsymbol{Q}_u) < d_{\mathcal{C}}$$

otherwise.

Figure 2.7 illustrates this interpretation using simple manifolds, in each of the cases $n_u < d_{\mathcal{C}}$, $n_u = d_{\mathcal{C}}$, and $n_u > d_{\mathcal{C}}$. In Fig. 2.7 (a) we assume $\mathcal{C}$ is a sphere defined in the (x, y, z) space, and y is the only actuated coordinate, so $d_{\mathcal{C}} = 2$ and $n_u = 1$. Irrespective of the position of $\boldsymbol{q}$ on $\mathcal{C}$, the projection of all vectors $\dot{\boldsymbol{q}} \in \mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ onto their y coordinate spans a vector space of dimension at most one in this case (the y axis). Since the sphere is two-dimensional, and the y axis is one-dimensional, all points in the sphere are forward singularities. In Fig. 2.7 (b), the C-space is the same sphere, but now we actuate both x and y, so $n_u = d_{\mathcal{C}} = 2$. The projection of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ onto the (x, y) plane is two-dimensional in a generic point $\boldsymbol{q}_1$, but only one-dimensional in $\boldsymbol{q}_2$, so $\boldsymbol{q}_2$ is a forward singularity. In Fig. 2.7 (c), the C-space is a curve lying in a plane Π, and x and y are both actuated, so this case corresponds to a robot with redundant actuation ($n_u = 2$ and $d_{\mathcal{C}} = 1$). In a generic configuration like $\boldsymbol{q}_1$, the projection of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ onto the (x, y) plane yields a one-dimensional vector space, so $\boldsymbol{q}_1$ is not a forward singularity. Points like $\boldsymbol{q}_2$ in which the $\dot{\boldsymbol{q}}$ vectors project to the zero vector are forward singularities.

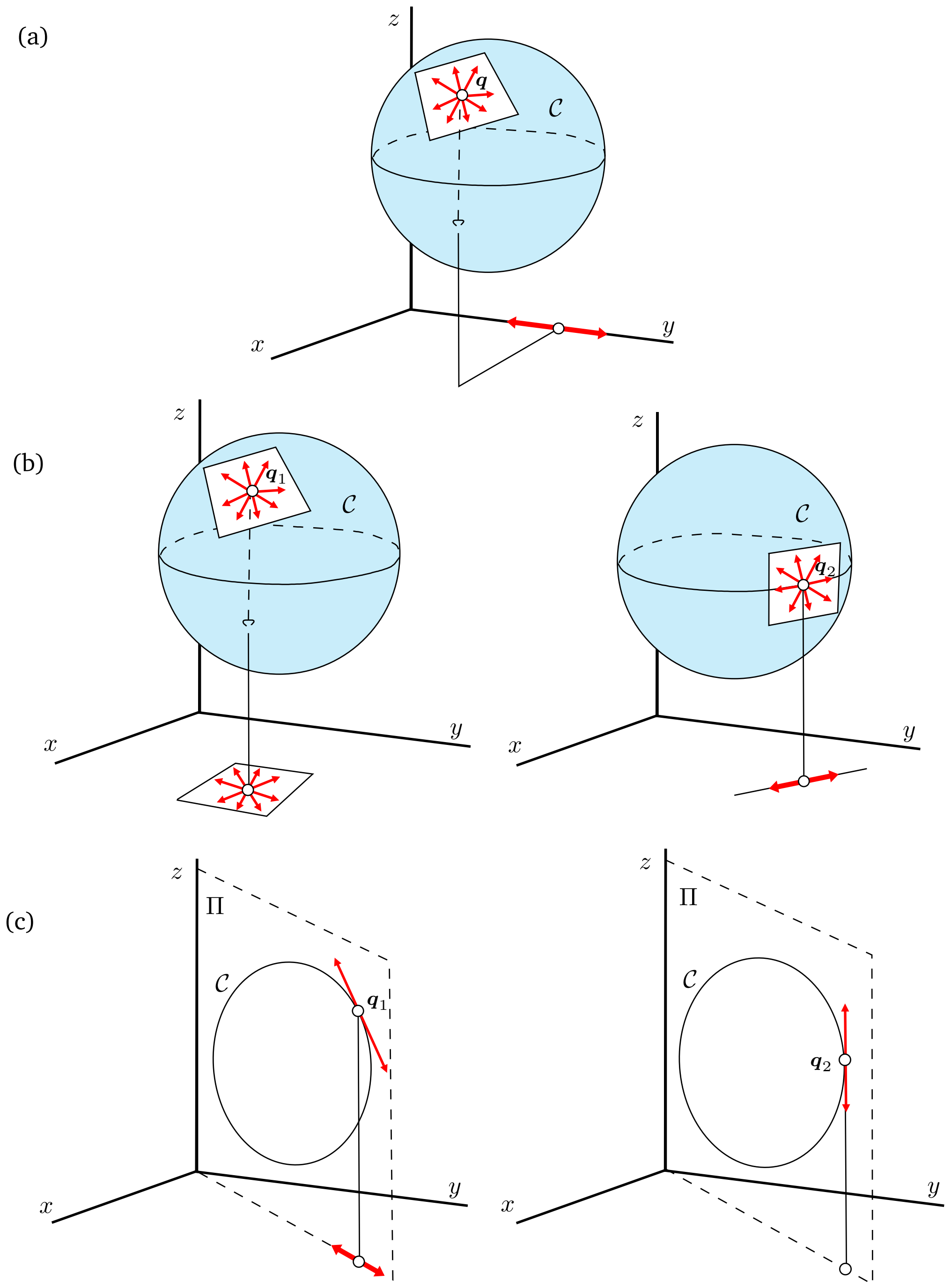


Figure 2.7: Geometric interpretation of forward singularities. At such singularities, the projection of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$ onto the space $\mathbb{R}^{n_u}$ of the $\boldsymbol{q}_u$ coordinates spans a subspace of dimension smaller than $d_{\mathcal{C}}$. In (a) $d_{\mathcal{C}} = 2$ but only the y coordinate is actuated, so $n_u < d_{\mathcal{C}}$ and all $\boldsymbol{q} \in \mathcal{C}$ are forward singularities. In (b) and (c) both x and y are actuated, so these figures depict the case of a robot with $n_u = d_{\mathcal{C}}$ and $n_u > d_{\mathcal{C}}$ respectively, in which $\boldsymbol{q}_1$ is a regular configuration and $\boldsymbol{q}_2$ is a forward singularity.

2.3 Dynamic Model

2.3.1 Lagrange's Equation with Multipliers

The kinematic equations in Section 2.1 characterize the feasible motions of the robot, but not the relationship between the actual motions and the forces that generate them. Such a relationship is given by the dynamic model of the robot, which can be formulated using Lagrange's equation with multipliers [23, 24]. In an inertial reference frame, this equation takes the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial K(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}} - \frac{\partial K(\boldsymbol{q},\dot{\boldsymbol{q}})}{\partial \boldsymbol{q}} + \frac{\partial U(\boldsymbol{q})}{\partial \boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \boldsymbol{\lambda} = \boldsymbol{Q}_u\, \boldsymbol{u} + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}}), \tag{2.9}$$

where $K(\boldsymbol{q},\dot{\boldsymbol{q}})$ and $U(\boldsymbol{q})$ are the kinetic and potential energy functions of the robot, $\boldsymbol{\lambda} \in \mathbb{R}^{n_e}$ is a vector of Lagrange multipliers, $\boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is the generalized force of friction, and $\boldsymbol{u}$ is the action vector. Assuming that each actuator modifies just one coordinate of $\boldsymbol{q}$, and that the robot contains n_u actuators, we have

$$\boldsymbol{u} = (u_1, \dots, u_{n_u}) \in \mathbb{R}^{n_u},$$

where u_i is the force or torque acting on the i-th coordinate of $\boldsymbol{q}_u$.

Using the fact that the kinetic energy takes the form

$$K(\boldsymbol{q},\dot{\boldsymbol{q}}) = \frac{1}{2}\,\dot{\boldsymbol{q}}^\top\, \boldsymbol{M}(\boldsymbol{q})\,\dot{\boldsymbol{q}},$$

where $\boldsymbol{M}(\boldsymbol{q})$ is a symmetric and positive-definite mass matrix, Eq. (2.9) can be expressed as

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \boldsymbol{\lambda} = \boldsymbol{Q}_u\, \boldsymbol{u} + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}}), \tag{2.10}$$

where

$$\boldsymbol{G}(\boldsymbol{q}) = \frac{\partial U}{\partial \boldsymbol{q}},$$

and $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is the so-called Coriolis matrix. The elements of $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ can be computed using the following formula from [24]:

$$\boldsymbol{C}_{i,j}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \frac{1}{2}\sum_{k=1}^{n_q}\left(\frac{\partial \boldsymbol{M}_{i,j}}{\partial q_k} + \frac{\partial \boldsymbol{M}_{i,k}}{\partial q_j} - \frac{\partial \boldsymbol{M}_{k,j}}{\partial q_i}\right).$$

Equation (2.10) can now be used to solve the following problems for a given state $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$:

- Forward dynamics: given $\boldsymbol{u}$, compute $\ddot{\boldsymbol{q}}$.
- Inverse dynamics: given $\ddot{\boldsymbol{q}}$, compute $\boldsymbol{u}$.

The solution to the first problem allows us to define a state space model for the robot, which will be necessary in Chapters 3, 4 and 5 to develop our trajectory planning and LQR control methods. The solution to the second problem, in turn, will be used in Chapter 5 to present a computed-torque controller for closed-chain robotic systems. Let us see how the two problems can be solved.

2.3.2 Forward Dynamics

Note that when $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\boldsymbol{u}$ have a known value, Eq. (2.10) is a system of n_q equations in $n_q + n_e$ unknowns (the n_q coordinates of $\ddot{\boldsymbol{q}}$ and the n_e Lagrange multipliers) so we need additional equations to be able to determine $\ddot{\boldsymbol{q}}$. These are provided by Eq. (2.3), which ensures that $\ddot{\boldsymbol{q}}$ will be a feasible acceleration. Equations (2.10) and (2.3) then give rise to the system

$$\begin{bmatrix} \boldsymbol{M}(\boldsymbol{q}) & \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \\ \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix}, \tag{2.11}$$

where

$$\boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) = \boldsymbol{Q}_u\, \boldsymbol{u} + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\, \dot{\boldsymbol{q}}.$$

Note that, since we assumed $\boldsymbol{\Phi}_{\boldsymbol{q}}$ to be full rank for all $\boldsymbol{q} \in \mathcal{C}$, the matrix on the left-hand side of Eq. (2.11) is invertible, so we can write

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{I}_{n_q} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}(\boldsymbol{q}) & \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \\ \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q}) & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{bmatrix}, \tag{2.12}$$

where $\boldsymbol{I}_{n_q}$ is the $n_q \times n_q$ identity matrix. Equation (2.12) already provides the solution to the forward dynamics, but for planning and control purposes it is often transformed into a first-order ordinary differential equation of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}). \tag{2.13}$$

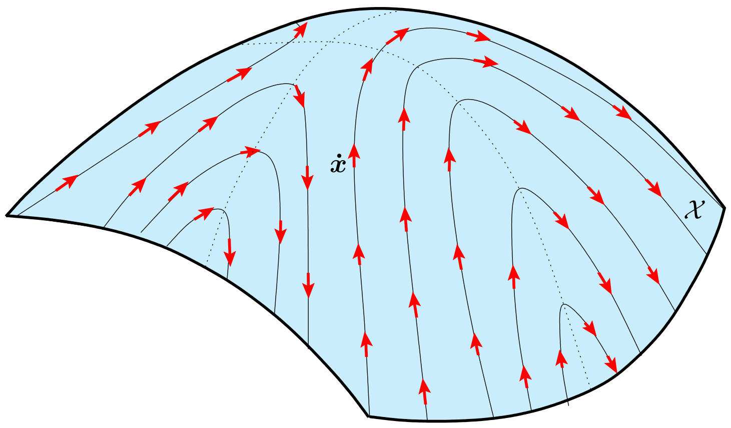


Figure 2.8: For each value of $\boldsymbol{u}$, Eq. (2.13) defines a vector field over $\mathcal{X}$.

Geometrically, this equation defines an action-dependent vector field over $\mathcal{X}$ (Fig. 2.8) whose integral curves correspond to the feasible trajectories of the robot. Thus, a feasible trajectory $\boldsymbol{x}(t)$ will be one that, for all t, satisfies the differential algebraic equation

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}(t)) = \boldsymbol{0} & \text{(2.14a)} \\ \dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t)) & \text{(2.14b)} \end{cases}$$

for some control function $\boldsymbol{u}(t)$.

In many works, the trajectory $\boldsymbol{x}(t)$ that results from $\boldsymbol{u}(t)$ is obtained by simply integrating Eq. (2.14b) numerically, without enforcing Eq. (2.14a) during the process. This is done on the grounds that Eq. (2.14a) is already accounted for implicitly by Eq. (2.14b), but note that Eq. (2.14b) only contains the acceleration constraint in Eq. (2.3) and not the position and velocity constraints in Eqs. (2.1) and (2.2). Thus, any integration method applied to Eq. (2.14b) alone will generate trajectories $\boldsymbol{x}(t)$ that, sooner or later, will inevitably drift away from $\mathcal{X}$, which results in physically inconsistent motions not fulfilling the assembly constraints of the robot. To avoid this problem one must rely on numerical methods that actively take Eq. (2.14a) into account [25–30]. In particular, in this thesis we make an extensive use of integration in local coordinates [25]. The idea consists in using local charts of $\mathcal{X}$ to express Eq. (2.14b) in some set of local parameters of $\mathcal{X}$, to then perform the numerical integration in such parameters and map the result back to $\mathcal{X}$. In our implementations, we use the version in [30] of this technique, which constructs the necessary maps using tangent-space parameterizations. Details on these techniques will be recalled as needed in the following chapters.

2.3.3 Inverse Dynamics

To solve for $\boldsymbol{u}$ in Eq. (2.10), let $\boldsymbol{\Lambda}$ be the $n_q \times d_{\mathcal{C}}$ matrix we defined in Section 2.2.3, which contains, by columns, a basis of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$. This matrix can be obtained from the QR decomposition of $\boldsymbol{\Phi}_{\boldsymbol{q}}^{\top}$ for example [31, 32].

If we multiply the equation by $\boldsymbol{\Lambda}^{\top}$ and rearrange the terms we have

$$\underbrace{\boldsymbol{\Lambda}^{\top}\boldsymbol{Q}_u}_{\boldsymbol{N}(\boldsymbol{q})}\ \boldsymbol{u} = \underbrace{\boldsymbol{\Lambda}^{\top}\Big(\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}})\Big)}_{\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})} - \underbrace{\boldsymbol{\Lambda}^{\top}\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{\top}\ \boldsymbol{\lambda}}_{\boldsymbol{0}},$$

where $\boldsymbol{\Lambda}^{\top}\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{\top}\ \boldsymbol{\lambda}$ must be zero, as $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{\top}\ \boldsymbol{\lambda}$ is a vector of $\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$ that is orthogonal to the columns of $\boldsymbol{\Lambda}$. By defining $\boldsymbol{N}(\boldsymbol{q})$ and $\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ as indicated, we then see that the $\boldsymbol{u}$ values that are compatible with $\ddot{\boldsymbol{q}}$ are those that satisfy

$$\boldsymbol{N}(\boldsymbol{q})\ \boldsymbol{u} = \boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}). \tag{2.15}$$

Two situations are possible when analyzing the solvability of Eq. (2.15) at $\boldsymbol{x} = (\boldsymbol{q},\dot{\boldsymbol{q}}) \in \mathcal{X}$: either the equation will be solvable for all $\ddot{\boldsymbol{q}} \in \mathcal{A}_{\boldsymbol{x}}$, or there will be some $\ddot{\boldsymbol{q}} \in \mathcal{A}_{\boldsymbol{x}}$ for which no compatible $\boldsymbol{u}$ exists. By analogy to the open-chain case, we will refer to these situations by saying that the robot is “fully actuated at $\boldsymbol{x}$” or “underactuated at $\boldsymbol{x}$”, respectively.

It is important to characterize the states $\boldsymbol{x}$ in which the robot is underactuated, as in such states we will not be able to command all possible accelerations, thus losing the ability to control position or velocity errors along some directions. In order to do so, note that $\boldsymbol{N}(\boldsymbol{q})$ is a $d_{\mathcal{C}} \times n_u$ matrix and $\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) \in \mathbb{R}^{d_{\mathcal{C}}}$. Using Eq. (2.4), moreover, it is easy to see that $\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$ is surjective, so it spans the entire space $\mathbb{R}^{d_{\mathcal{C}}}$ as $\ddot{\boldsymbol{q}}$ varies inside $\mathcal{A}_{\boldsymbol{x}}$. This must be true as, by substituting Eq. (2.4) into $\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})$, we get

$$\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = \boldsymbol{\Lambda}^{\top}\Big(\boldsymbol{M}(\boldsymbol{q})\ (\ddot{\boldsymbol{q}}_{\perp} + \boldsymbol{\Lambda}\boldsymbol{\alpha}) + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}})\Big),$$

so

$$\boldsymbol{\tau}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = \bar{\boldsymbol{M}}(\boldsymbol{q})\ \boldsymbol{\alpha} + \boldsymbol{b}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}_{\perp}), \tag{2.16}$$

where

$$\bar{\boldsymbol{M}}(\boldsymbol{q}) = \boldsymbol{\Lambda}^{\top}\boldsymbol{M}(\boldsymbol{q})\ \boldsymbol{\Lambda},$$

and

$$\boldsymbol{b}_{\text{ID}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}_{\perp}) = \boldsymbol{\Lambda}^{\top}\Big(\boldsymbol{M}(\boldsymbol{q})\ \ddot{\boldsymbol{q}}_{\perp} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\ \dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) + \boldsymbol{\tau}_{\text{fric}}(\boldsymbol{q},\dot{\boldsymbol{q}})\Big).$$

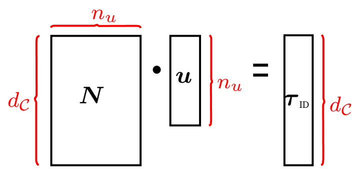
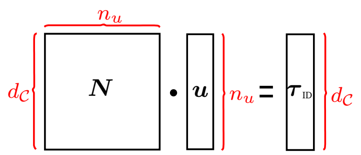
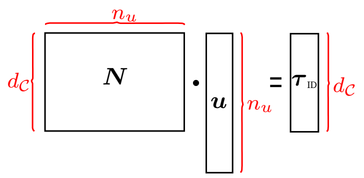
System dimensions	System shape	Rank of $\boldsymbol{N}$	Dim. of Ker $\boldsymbol{N}$	Number of solutions $\boldsymbol{u}$ for the given $\ddot{\boldsymbol{q}} \in \mathcal{A}_{\boldsymbol{x}}$	Actuation type
$n_u < d_\mathcal{C}$		n_u	0	The system is unsolvable for almost all $\ddot{\boldsymbol{q}}$, and the solution $\boldsymbol{u}$ is unique when $\boldsymbol{\tau}_{\text{ID}}$ is in $C(\boldsymbol{N})$	Underactuated
		$< n_u$	> 0	The system is unsolvable for almost all $\ddot{\boldsymbol{q}}$, and there are infinitely-many solutions $\boldsymbol{u}$ when $\boldsymbol{\tau}_{\text{ID}}$ is in $C(\boldsymbol{N})$	
$n_u = d_\mathcal{C}$		$d_\mathcal{C}$	0	A unique $\boldsymbol{u}$ exists for every $\ddot{\boldsymbol{q}} \in \mathcal{A}_{\boldsymbol{x}}$	Properly actuated
		$< d_\mathcal{C}$	> 0	For almost all $\ddot{\boldsymbol{q}}$ there is no corresponding $\boldsymbol{u}$. For those $\ddot{\boldsymbol{q}}$ that make $\boldsymbol{\tau}_{\text{ID}}$ into $C(\boldsymbol{N})$, there are infinitely-many $\boldsymbol{u}$	Underactuated
$n_u > d_\mathcal{C}$		$d_\mathcal{C}$	$n_u - d_\mathcal{C}$	For all $\ddot{\boldsymbol{q}}$, there are infinitely-many $\boldsymbol{u}$	Overactuated
		$< d_\mathcal{C}$	$> n_u - d_\mathcal{C}$	The system is unsolvable for almost all $\ddot{\boldsymbol{q}}$, and when $\boldsymbol{\tau}_{\text{ID}}$ in $C(\boldsymbol{N})$, there are infinitely-many $\boldsymbol{u}$	Underactuated

Table 2.1: Structure of the solution set of Eq. (2.15) depending on n_u and the rank of $\boldsymbol{N}$. $C(\boldsymbol{N})$ indicates the column space of $\boldsymbol{N}$.

$$\begin{array}{l}\text{At a given}\\ \text{configuration } \boldsymbol{q}\\ \text{the system is}\end{array}
\begin{cases}
\begin{array}{l}\textbf{Underactuated}\\ \text{if } \operatorname{rank}(\boldsymbol{N}(\boldsymbol{q})) < d_{\mathcal{C}}\\ \text{(irrespective of } n_u\text{)}\end{array} \\[3em]
\begin{array}{l}\textbf{Fully actuated}\\ \text{if } \operatorname{rank}(\boldsymbol{N}(\boldsymbol{q})) = d_{\mathcal{C}}\end{array}
\begin{cases}
\begin{array}{l}\textbf{Properly actuated}\\ \text{If, in addition, } n_u = d_{\mathcal{C}}\end{array} \\[2em]
\begin{array}{l}\textbf{Overactuated}\\ \text{If, in addition, } n_u > d_{\mathcal{C}}\end{array}
\end{cases}
\end{cases}$$

Figure 2.9: Actuation types of a robot at a given $\boldsymbol{q} \in \mathcal{C}$.

Since $\boldsymbol{M}(\boldsymbol{q})$ is positive definite, $\bar{\boldsymbol{M}}(\boldsymbol{q})$ must be positive-definite and thus full rank, so the mapping from $\boldsymbol{\alpha}$ to $\boldsymbol{\tau}_{\text{ID}}$ defined by Eq. (2.16) has to be surjective as we claimed. It is clear then that the robot is underactuated at $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$ if, and only if,

$$\operatorname{rank}\Big(\boldsymbol{N}(\boldsymbol{q})\Big) = \operatorname{rank}\Big(\boldsymbol{\Lambda}^{\top}\boldsymbol{Q}_u\Big) < d_{\mathcal{C}}.$$

In particular, when $n_u < d_{\mathcal{C}}$, $\boldsymbol{N}(\boldsymbol{q})$ is a tall matrix whose rank is always less than $d_{\mathcal{C}}$, so in this case the system is underactuated for all $\boldsymbol{x} \in \mathcal{X}$.

By the result in Section 2.2.3, we also see that the robot is underactuated at $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$ if, and only if, $\boldsymbol{q}$ is a forward singularity. Thus, whether the robot is underactauted or not is a property of the configuration rather than the state, and depends on the actual choice of actuated coordinates.

Fully-actuated robots are further classified into properly actuated or over-actuated depending on whether Eq. (2.15) has just one, or infinitely-many solutions for $\boldsymbol{u}$ respectively. Clearly, if $\operatorname{rank}(\boldsymbol{N}(\boldsymbol{q})) = d_{\mathcal{C}}$, the equation can only have infinitely-many solutions if the kernel of $\boldsymbol{N}(\boldsymbol{q})$ is of dimension one or higher, which occurs whenever $n_u > d_{\mathcal{C}}$. Table 2.1 and Figure 2.9 provide visual summaries of this classification.

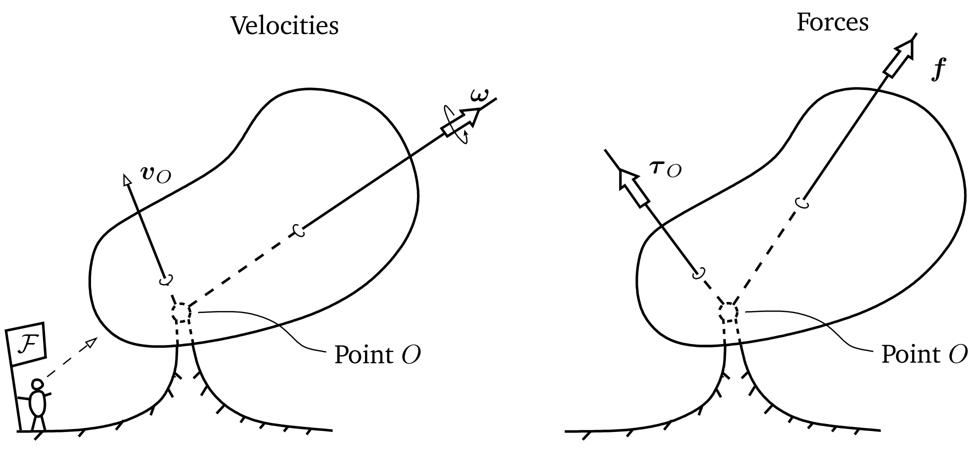


Figure 2.10: 3D vectors forming a spatial velocity (left) and a spatial force (right) on a rigid body. The coordinate system on which the quantities are expressed has its origin at some point O, and its axes aligned with three orthonormal vectors of $\mathbb{R}^3$ (not drawn). Spatial velocities and accelerations are relative to the reference frame $\mathcal{F}$ from where they are being measured.

2.4 Recursive Dynamic Algorithms

The previous section presented a compact formulation of the dynamics of closed-chain systems using the Lagrange formulation, but not the means to compute all of the terms involved. To cover this gap, we next recall various algorithms [33] for efficiently computing such terms. Most of these algorithms exploit the Newton-Euler formulation, which is based on the balance of forces and torques in every rigid body of our robot. They also rely on recursive formulations using spatial vector algebra, which is a concise notation that uses 6D vectors and matrices to represent dynamics quantities. The notation and algorithms we use follow those of Featherstone in [33], but further details can be found in [34, 35]. Equivalent Lie-theoretic algorithms can also be found in [36].

2.4.1 Spatial Quantities and Notation

We start summarizing the spatial quantities that are necessary to work with the dynamic equations. All of such quantities will be denoted with a hat and they will be expressed in some coordinate system A defined by an origin O and an othonormal basis of $\mathbb{R}^3$. Such a system will be indicated with a leading superscript, so ${}^{A}\hat{\boldsymbol{v}}$ will refer to the spatial quantity $\hat{\boldsymbol{v}}$ expressed in A coordinates. When working with spatial velocities or accelerations, however, the notion of coordinate system must not be confused with that of a reference frame. The latter refers to the body from which a particular velocity or acceleration is being measured or observed, the result being expressible in any coordinate system of choice. The reference frame $\mathcal{F}$ from which a velocity or acceleration is being measured will be mentioned explicitly or understood by context. The following quantities can be defined:

- The **spatial velocity** of a rigid body is the 6D vector

$$^{A}\hat{\boldsymbol{v}} = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v}_O \end{bmatrix},$$

  where $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity of the body and $\boldsymbol{v}_O \in \mathbb{R}^3$ is the linear velocity of the body-fixed point that instantaneously coincides with O (Fig. 2.10, left). Both $\boldsymbol{\omega}$ and $\boldsymbol{v}_O$ are assumed to be measured from some reference frame $\mathcal{F}$ and expressed in A coordinates. Relative to $\mathcal{F}$, the body can be thought of as moving with linear velocity $\boldsymbol{v}_O$ while simultaneously rotating with angular velocity $\boldsymbol{\omega}$ about an axis through O. Since the velocity of any point P on the body is $\boldsymbol{v}_P = \boldsymbol{v}_O + \boldsymbol{\omega} \times \overrightarrow{OP}$, $^{A}\hat{\boldsymbol{v}}$ actually encodes the overall velocity distribution of the body. A spatial velocity is also called a **twist** in screw theory [37, 38].

- The **spatial acceleration** of the previous body relative to $\mathcal{F}$ is simply the time derivative of $^{A}\hat{\boldsymbol{v}}$ in $\mathcal{F}$. If we define the pseudoacceleration of a point P of the body as the quantity

$$\boldsymbol{a}_P = \mathbf{a}_P - \boldsymbol{\omega} \times \boldsymbol{v}_O,$$

  where $\mathbf{a}_P$ is the conventional acceleration of P, it can be shown that the spatial acceleration of the body takes the form

$$^{A}\hat{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{a}_O \end{bmatrix},$$

  where $\boldsymbol{\alpha} = \dot{\boldsymbol{\omega}}$ and $\boldsymbol{a}_O$ is the pseudoacceleration of the body-fixed point that instantaneously coincides with O. As with spatial velocities, $\boldsymbol{\alpha}$ and $\boldsymbol{a}_O$ determine the pseudoacceleration of any point P of the body via $\boldsymbol{a}_P = \boldsymbol{a}_O + \boldsymbol{\alpha} \times \overrightarrow{OP}$. Further details on the difference between pseudoaccelerations and conventional accelerations are given in [39].

- A **spatial force** acting on a rigid body, and expressed in A coordinates, is given by

$$^{A}\hat{\boldsymbol{f}} = \begin{bmatrix} \boldsymbol{\tau}_O \\ \boldsymbol{f} \end{bmatrix},$$

where $\boldsymbol{\tau}_O$ is a pure couple and $\boldsymbol{f}$ is a 3D force acting on a line passing through O (Fig. 2.10, right). The spatial force ${}^A\hat{\boldsymbol{f}}$ is used to represent the resultant of any system of forces acting on the rigid body: while $\boldsymbol{f}$ represents the resultant force, $\boldsymbol{\tau}_O$ represents the resultant moment with respect to O. A spatial force is also called a **wrench** in [37, 38].

- The **spatial inertia** of a rigid body represents, in a single 6×6 matrix, the mass, the location of the center of mass (CM), and the moments of inertia of the body. Given the mass m, the inertia tensor $\boldsymbol{\mathcal{I}}_{\text{CM}}$ about the CM, and the position $\vec{\boldsymbol{p}}_{\text{CM}}$ of the body's CM relative to O, the spatial inertia of the body in A coordinates is given by

$$ {}^A\hat{\boldsymbol{I}} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{\text{CM}} + m\ \boldsymbol{S}(\vec{\boldsymbol{p}}_{\text{CM}})\ \boldsymbol{S}(\vec{\boldsymbol{p}}_{\text{CM}})^\top & \boldsymbol{S}(\vec{\boldsymbol{p}}_{\text{CM}}) \\ \boldsymbol{S}(\vec{\boldsymbol{p}}_{\text{CM}})^\top & m\ \boldsymbol{I}_3 \end{bmatrix}, $$

where $\boldsymbol{\mathcal{I}}_{\text{CM}}$ and $\vec{\boldsymbol{p}}_{\text{CM}}$ are both expressed in the basis of A, and $\boldsymbol{S}(\vec{\boldsymbol{p}})$ is the skew-symmetric matrix defined by

$$ \boldsymbol{S}(\vec{\boldsymbol{p}}) = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix}. $$

In some situations a spatial quantity will be expressed in some coordinate system B, and we will need to re-express it in a new coordinate system A. Different quantites obey different transformation rules to this end. The transformation of spatial velocities is given by

$$ {}^A\hat{\boldsymbol{v}} = {}^A\boldsymbol{X}_B\ {}^B\hat{\boldsymbol{v}}, $$

where

$$ {}^A\boldsymbol{X}_B = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{S}({}^A\vec{\boldsymbol{p}}_B) & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} {}^A\boldsymbol{R}_B & \boldsymbol{0} \\ \boldsymbol{0} & {}^A\boldsymbol{R}_B \end{bmatrix} = \begin{bmatrix} {}^A\boldsymbol{R}_B & \boldsymbol{0} \\ \boldsymbol{S}({}^A\vec{\boldsymbol{p}}_B)\ {}^A\boldsymbol{R}_B & {}^A\boldsymbol{R}_B \end{bmatrix}. $$

In this equation, ${}^A\boldsymbol{R}_B$ is the rotation matrix that expresses the basis of B in the basis of A, and ${}^A\vec{\boldsymbol{p}}_B$ is the position vector of the origin of B relative to the origin of A and expressed in A coordinates. The transformation of spatial accelerations follows the same previous rule. Similarly, if ${}^A\hat{\boldsymbol{f}}$ and ${}^B\hat{\boldsymbol{f}}$ refer to a same spatial force expressed in A and B coordinates, we have

$$ {}^A\hat{\boldsymbol{f}} = {}^A\boldsymbol{X}_B^*\ {}^B\hat{\boldsymbol{f}}, $$

where

$$ {}^A\boldsymbol{X}_B^* = {}^A\boldsymbol{X}_B^{-\top}. $$

The transformation rule for spatial inertias is finally given by

$$^{A}\hat{\boldsymbol{I}} = {}^{A}\boldsymbol{X}_{B}^{*} \; {}^{B}\hat{\boldsymbol{I}} \; {}^{B}\boldsymbol{X}_{A}.$$

2.4.2 Modeling Closed-Chain Systems

The first step to model a closed-chain system is to select appropriate $\boldsymbol{q}$ coordinates to define the system configuration. The main choices in multibody dynamics include relative joint coordinates, reference point coordinates, and natural coordinates [18]. In our case we adopt relative joint coordinates because they provide compact formulations that ease the modeling of actuation and friction joint forces.

We next see how to formulate the loop-closure constraints and the equation of motion of a robot. For simplicity, we assume the robot has just one kinematic loop. If more loops were present we would just collect the closure constraints for a maximal set of independent loops in the robot, and the equation of motion would involve minor modifications. We also treat revolute or prismatic joints only, and assume that one of the links is fixed to the ground. More complex joints and mobile robots could also be accommodated using the results in [33].

Formulating the Loop-closure Constraints

A kinematic loop can be regarded as a serial chain in which the base and tip links are forced to coincide. Fig. 2.11 shows such a chain in exploded view, with our notation depicted. The links are labeled from $\mathcal{L}_0$ to $\mathcal{L}_n$, where $\mathcal{L}_0$ and $\mathcal{L}_n$ refer to the base and tip links respectively. The joints are labeled from J_1 to J_n, with J_i being the joint that connects links $\mathcal{L}_{i-1}$ and $\mathcal{L}_i$. We let q_i refer to the turning angle or displacement of J_i, so the overall loop configuration is given by

$$\boldsymbol{q} = (q_1, \ldots, q_n).$$

We also define two coordinate systems on each link: one attached to the joint that is closest to the base, and one attached to the joint that is closest to the tip. On link $\mathcal{L}_{i-1}$, these two systems are called the $(i-1)$ and $(i-1)'$ coordinates respectively. As a result, on joint J_i we have the $(i-1)'$ and i coordinate systems, which coincide when $q_i = 0$. In an assembled configuration of the loop, the n coordinates of the tip link must coincide with 0 coordinates of the base link.

With the previous definitions, the spatial transformation that locates the i coordinates relative to the $(i-1)$ ones is given by

$$^{i-1}\boldsymbol{X}_i = \boldsymbol{X}_{\mathcal{L}_{i-1}} \cdot \boldsymbol{X}_z(q_i) \tag{2.17}$$

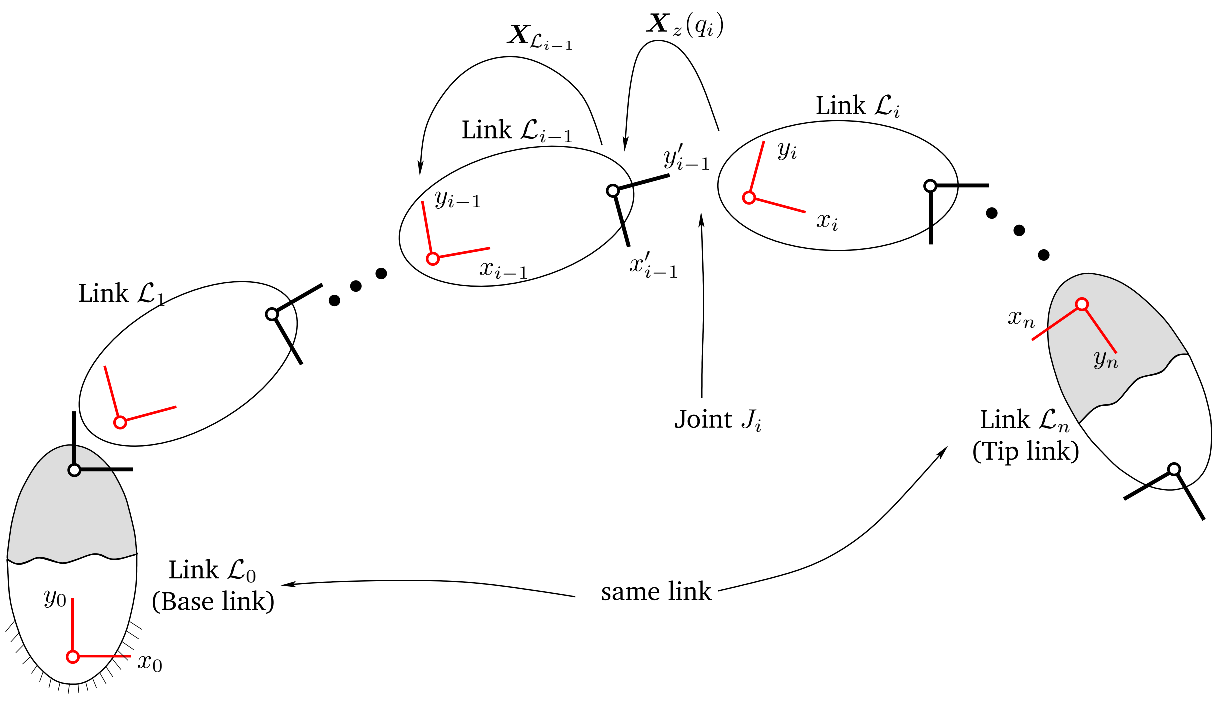


Figure 2.11: A kinematic loop in exploded view. The tip link is a copy of the base link. In an assembled configuration, these two links are forced to coincide.

where $\boldsymbol{X}_z(q_i)$ is the joint transformation about the z_i axis given in Table 2.2, and $\boldsymbol{X}_{\mathcal{L}_{i-1}}$ is a constant spatial transformation that locates the $(i-1)'$ coordinates relative to the $(i-1)$ ones. Then, the transformation of the whole loop is given by

$$^0\boldsymbol{X}_n(\boldsymbol{q}) = {}^0\boldsymbol{X}_1 \cdot {}^1\boldsymbol{X}_2 \cdot \ldots \cdot {}^{n-1}\boldsymbol{X}_n,$$

which can be decomposed as

$$^0\boldsymbol{X}_n(\boldsymbol{q}) = \begin{bmatrix} {}^0\boldsymbol{R}_n & \boldsymbol{0} \\ \boldsymbol{S}({}^0\vec{\boldsymbol{p}}_n)\,{}^0\boldsymbol{R}_n & {}^0\boldsymbol{R}_n \end{bmatrix}.$$

Thus, the loop closure constraint can be imposed by setting

$$^0\boldsymbol{R}_n = \boldsymbol{I}_3 \tag{2.18}$$

and

$$^0\vec{\boldsymbol{p}}_n = \boldsymbol{0}.$$

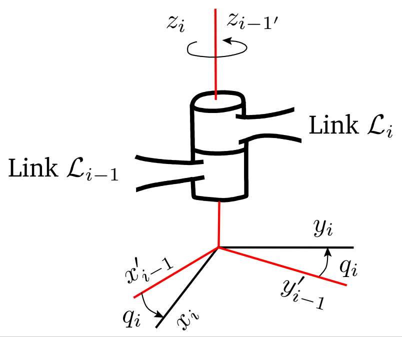
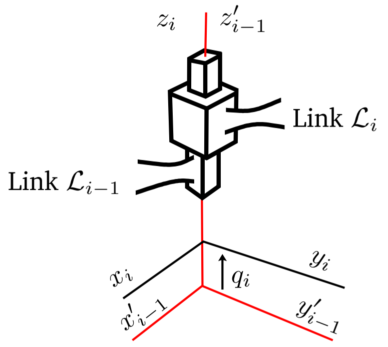
Joint type	$\boldsymbol{X}_z(q_i)$	${}^i\hat{\boldsymbol{S}}_i$
 z_i $z_{i-1'}$ Link $\mathcal{L}_i$ Link $\mathcal{L}_{i-1}$ y_i q_i x'_{i-1} y'_{i-1} q_i x_i	$\left[\begin{array}{cccccc} c(q_i) & -s(q_i) & 0 & 0 & 0 & 0 \\ s(q_i) & c(q_i) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & c(q_i) & -s(q_i) & 0 \\ 0 & 0 & 0 & s(q_i) & c(q_i) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$	$\left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 0 \\ 0 \\ 0 \end{array}\right]$
 z_i z'_{i-1} Link $\mathcal{L}_i$ Link $\mathcal{L}_{i-1}$ y_i x_i q_i x'_{i-1} y'_{i-1}	$\left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & -q_i & 0 & 1 & 0 & 0 \\ q_i & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$	$\left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 1 \end{array}\right]$

Table 2.2: Expressions of the joint transform $\boldsymbol{X}_z(q_i)$ and unit joint velocity ${}^i\hat{\boldsymbol{S}}_i$, where $s(q_i) = \sin(q_i)$ and $c(q_i) = \cos(q_i)$

Eq. (2.18) is a system of 3×3 scalar equations of which only three are independent. One can take, for example, the equations corresponding to the elements $(3, 2)$, $(1, 3)$ and $(2, 1)$ of ${}^0\boldsymbol{R}_n$ as representative of the 9 equations [40]. Thus, Eq. (2.1) can be formulated as

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0} \tag{2.19}$$

with

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \begin{bmatrix} {}^0\boldsymbol{R}_{n(3,2)} \\ {}^0\boldsymbol{R}_{n(1,3)} \\ {}^0\boldsymbol{R}_{n(2,1)} \\ {}^0\vec{\boldsymbol{p}}_{n(1)} \\ {}^0\vec{\boldsymbol{p}}_{n(2)} \\ {}^0\vec{\boldsymbol{p}}_{n(3)} \end{bmatrix}, \tag{2.20}$$

where the subindices refer to the selected elements of ${}^0\boldsymbol{R}_n$ and ${}^0\vec{\boldsymbol{p}}_n$.

The velocity constraint in Eq. (2.2) could be obtained by differentiating Eq. (2.19) symbolically with respect to time, but we can also obtain Eq. (2.2) in closed form as follows.

Note on the one hand that, since the tip and base links must coincide in Fig. 2.11, the absolute velocity of link $\mathcal{L}_n$ (i.e., relative to $\mathcal{L}_0$) must be null:

$$ {}^0\hat{\boldsymbol{v}}_n = \mathbf{0}. \tag{2.21}$$

On the other hand, ${}^0\hat{\boldsymbol{v}}_n$ can be expressed in terms of $\dot{\boldsymbol{q}} = (\dot{q}_1, \ldots, \dot{q}_n)$ using

$$ {}^0\hat{\boldsymbol{v}}_n = {}^0\hat{\boldsymbol{S}}_1\, \dot{q}_1 + \cdots + {}^0\hat{\boldsymbol{S}}_n\, \dot{q}_n, \tag{2.22}$$

where ${}^0\hat{\boldsymbol{S}}_i$ is the unit spatial velocity of $\mathcal{L}_i$ relative to $\mathcal{L}_{i-1}$ (i.e., assuming $\dot{q}_i = 1$), but expressed in 0 coordinates. This velocity can be obtained using the coordinate transformation

$$ {}^0\hat{\boldsymbol{S}}_i = {}^0\boldsymbol{X}_i\; {}^i\hat{\boldsymbol{S}}_i, \tag{2.23}$$

where ${}^i\hat{\boldsymbol{S}}_i$ is the same velocity but expressed in i coordinates (Table 2.2). Then, Eq. (2.22) can be compactly written as

$$ {}^0\hat{\boldsymbol{v}}_n = \boldsymbol{J}_n\, \dot{\boldsymbol{q}}, \tag{2.24}$$

where

$$ \boldsymbol{J}_n = \begin{bmatrix} {}^0\hat{\boldsymbol{S}}_1 & \cdots & {}^0\hat{\boldsymbol{S}}_n \end{bmatrix} \tag{2.25}$$

is the screw Jacobian of the kinematic loop. If we finally combine Eqs. (2.21) and (2.24), we obtain the desired closed-form expression for the velocity constraint:

$$ \boldsymbol{J}_n\, \dot{\boldsymbol{q}} = \mathbf{0}. \tag{2.26}$$

Note that, since we assumed the loop to be assembled when obtaining Eq. (2.26), it must be $\boldsymbol{\Phi}_{\boldsymbol{q}} = \boldsymbol{J}_n$ when $\boldsymbol{\Phi}(\boldsymbol{q}) = \mathbf{0}$, so Eq. (2.26) can really be used as a proper velocity constraint.

The acceleration constraint can be obtained in a similar manner. We know that the spatial acceleration of the tip link relative to the base link must be null, i.e.,

$$ {}^0\hat{\boldsymbol{a}}_n = \mathbf{0}. \tag{2.27}$$

Also, the time derivative of Eq. (2.24) gives

$$ {}^0\hat{\boldsymbol{a}}_n = \boldsymbol{J}_n\, \ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_n\, \dot{\boldsymbol{q}}, \tag{2.28}$$

where

$$ \dot{\boldsymbol{J}}_n = \begin{bmatrix} {}^0\dot{\hat{\boldsymbol{S}}}_1 & \cdots & {}^0\dot{\hat{\boldsymbol{S}}}_n \end{bmatrix}, $$

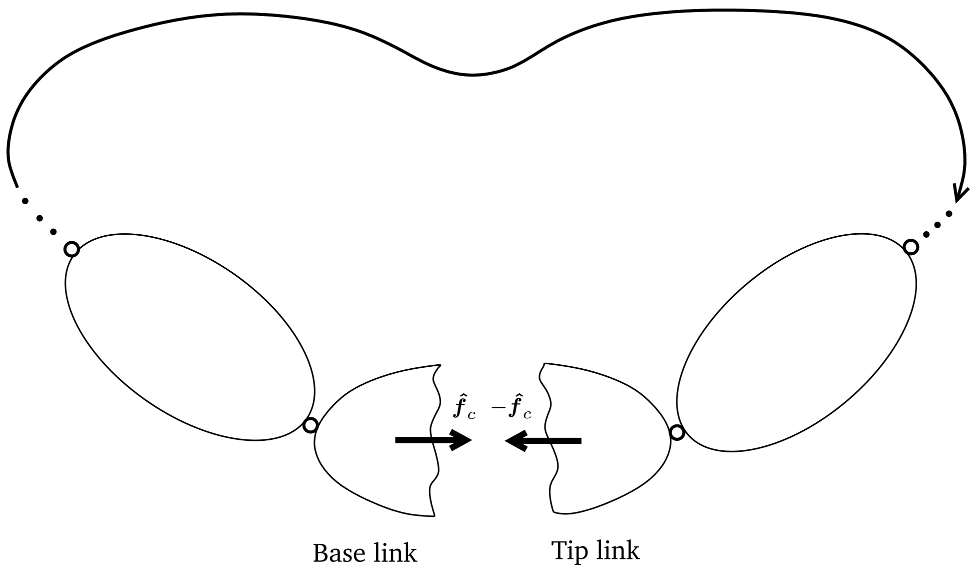


Figure 2.12: A kinematic loop can be thought of as being cut at some link, but subject to the spatial constraint forces $\hat{\boldsymbol{f}}_c$ and $-\hat{\boldsymbol{f}}_c$.

and the columns of $\dot{\boldsymbol{J}}_n$ satisfy ${}^0\dot{\hat{\boldsymbol{S}}}_i = \hat{\boldsymbol{v}}_i \times {}^0\hat{\boldsymbol{S}}_i$ [33]. By combining Eqs. (2.27) and (2.28) we see that the acceleration constraint is

$$\boldsymbol{J}_n\, \ddot{\boldsymbol{q}} = \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}),$$

where

$$\boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -\dot{\boldsymbol{J}}_n\, \dot{\boldsymbol{q}}.$$

From Eq. (2.28), we also see that $\boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the negative of the spatial acceleration of the tip link when $\ddot{\boldsymbol{q}} = \boldsymbol{0}$, which is a quantity that is recursively obtained during the calculation of $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\boldsymbol{G}(\boldsymbol{q})$, as part of the Newton-Euler method of inverse dynamics later described in Section 2.4.3. This observation is crucial to avoid redundant calculations.

Formulating the Equation of Motion

To formulate the equation of motion, we view the loop in Fig. 2.11 as being cut at some link, but subject to the spatial constraint forces $\hat{\boldsymbol{f}}_c$ and $-\hat{\boldsymbol{f}}_c$ that the half links exert on each other (Fig 2.12). Geometrically, the half links play the same role as the base and tip links in Fig. 2.11, but each has half the inertia of the original link. Under this view, thus, the equation of motion of the loop will be that of the equivalent serial chain shown in Fig 2.12, with an additional generalized force $\boldsymbol{\tau}_c$ modeling the joint effect of $\hat{\boldsymbol{f}}_c$ and $-\hat{\boldsymbol{f}}_c$:

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u}) + \boldsymbol{\tau}_c. \tag{2.29}$$

Note here that $\boldsymbol{M}(\boldsymbol{q})$, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}$ and $\boldsymbol{G}(\boldsymbol{q})$ only depend on the inertial properties of the links, and thus they can be computed assuming $\boldsymbol{\tau}_c = \boldsymbol{0}$ in the open chain. Efficient algorithms to obtain these quantities are given in the following two sections. The term $\boldsymbol{\tau}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u})$, in turn, is easy to formulate, as its i-th coordinate is simply the sum of the motor force or torque applied at the i-th joint (which is null if the joint is passive) and the friction force or torque at the joint. Finally, [33] shows that

$$\boldsymbol{\tau}_c = \boldsymbol{J}_n^{\top}\,\hat{\boldsymbol{f}}_c,$$

so by comparing Eq. (2.29) with Eq. (2.10), and knowing that $\boldsymbol{\Phi}_{\boldsymbol{q}} = \boldsymbol{J}_n$ when $\boldsymbol{q} \in \mathcal{C}$, we have $\hat{\boldsymbol{f}}_c = -\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers.

2.4.3 Computing the Kinematic, Gravity and Coriolis Terms

In the previous section we showed that the gravity and Coriolis terms of the loop mechanism, $\boldsymbol{G}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}$, correspond to those of the serial chain obtained from cutting the loop at a link and assuming $\boldsymbol{\tau}_c = \boldsymbol{0}$. This allows us to use the recursive Newton-Euler algorithm (RNEA) to compute the sum of such terms [33]. This algorithm solves the inverse dynamics of an open chain, and thus it obtains

$$\boldsymbol{\tau}_{\text{open}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) = \boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}),$$

which is the vector of joint torques that produce $\ddot{\boldsymbol{q}}$ in such a chain. Thus, if we apply the algorithm using $\ddot{\boldsymbol{q}} = \boldsymbol{0}$, we will obtain the desired sum of Coriolis and gravity terms:

$$\boldsymbol{\tau}_{\text{open}}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{0}) = \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) \tag{2.30}$$

In addition, we will see that the kinematic functions $\boldsymbol{\Phi}(\boldsymbol{q})$, $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})$ and $\boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}})$ can also be obtained recursively when applying the RNEA for $\ddot{\boldsymbol{q}} = \boldsymbol{0}$. We next explain the RNEA by dividing it in two parts.

Forward Propagation of Link Velocities and Accelerations

In the first part, the RNEA computes the absolute velocities and accelerations of each link $\mathcal{L}_i$ in the open chain, starting from the fixed base and going forward towards the tip. For each link $\mathcal{L}_i$, we express these quantities in i coordinates.

Since ${}^0\hat{\boldsymbol{v}}_0 = \mathbf{0}$, the absolute velocity of $\mathcal{L}_i$ can be obtained using the recurrence

$$ {}^i\hat{\boldsymbol{v}}_i = {}^i\boldsymbol{X}_{i-1}\, {}^{i-1}\hat{\boldsymbol{v}}_{i-1} + {}^i\hat{\boldsymbol{S}}_i\, \dot{q}_i, \tag{2.31} $$

where ${}^i\hat{\boldsymbol{S}}_i\, \dot{q}_i$ is the velocity of $\mathcal{L}_i$ relative to $\mathcal{L}_{i-1}$, and ${}^i\hat{\boldsymbol{S}}_i$ is given in Table 2.2. By differentiating Eq. (2.31) with respect to time we also obtain the recurrence

$$ {}^i\hat{\boldsymbol{a}}_i = {}^i\boldsymbol{X}_{i-1}\, {}^{i-1}\hat{\boldsymbol{a}}_{i-1} + {}^i\hat{\boldsymbol{S}}_i\, \ddot{q}_i + {}^i\dot{\hat{\boldsymbol{S}}}_i\, \dot{q}_i, \tag{2.32} $$

Thus, the acceleration of $\mathcal{L}_i$ can also be obtained recursively starting from ${}^0\hat{\boldsymbol{a}}_0 = \mathbf{0}$ and using ${}^i\dot{\hat{\boldsymbol{S}}}_i = \hat{\boldsymbol{v}}_i \times {}^i\hat{\boldsymbol{S}}_i$ [33]. However, since our goal is to compute $\boldsymbol{\tau}_{\text{open}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbf{0})$ in Eq. (2.30), we have to set $\ddot{\boldsymbol{q}} = \mathbf{0}$ in Eq. (2.32), and the recurrence simplifies to

$$ {}^i\hat{\boldsymbol{a}}_i = {}^i\boldsymbol{X}_{i-1}\, {}^{i-1}\hat{\boldsymbol{a}}_{i-1} + {}^i\dot{\hat{\boldsymbol{S}}}_i\, \dot{q}_i. \tag{2.33} $$

In this forward pass, we can also compute ${}^0\boldsymbol{X}_i$ recursively using

$$ {}^0\boldsymbol{X}_i = {}^0\boldsymbol{X}_{i-1}\, {}^{i-1}\boldsymbol{X}_i, \tag{2.34} $$

which allows us to formulate $\boldsymbol{\Phi}(\boldsymbol{q})$ as in Eq. (2.20) once ${}^0\boldsymbol{X}_n$ is obtained. Moreover, we can use Eqs. (2.34) and (2.23) to obtain the loop Jacobian $\boldsymbol{J}_n$ of Eq. (2.25):

$$ \boldsymbol{J}_n = \begin{bmatrix} {}^0\boldsymbol{X}_1\, {}^1\hat{\boldsymbol{S}}_1 & \cdots & {}^0\boldsymbol{X}_n\, {}^n\hat{\boldsymbol{S}}_n \end{bmatrix}. $$

Finally, by recalling that $\boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the negative spatial acceleration of the tip link in 0 coordinates when $\ddot{\boldsymbol{q}} = \mathbf{0}$ (Section 2.4.2), we can use Eqs. (2.33) and (2.34) to obtain

$$ \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -{}^0\boldsymbol{X}_n\, {}^n\hat{\boldsymbol{a}}_n. $$

Backward Propagation of Forces

In the second part, the force transmitted across each joint J_i is recursively computed, starting from the tip and going backwards towards the base. For each link $\mathcal{L}_i$, we express these quantities in i coordinates.

From the Newton-Euler equation for a rigid body [33], note that the net force acting on link $\mathcal{L}_i$ is given by

$$ {}^i\hat{\boldsymbol{f}}_{i,\text{net}} = {}^i\hat{\boldsymbol{I}}_i\, {}^i\hat{\boldsymbol{a}}_i + {}^i\hat{\boldsymbol{v}}_i \times {}^i\hat{\boldsymbol{I}}_i\, {}^i\hat{\boldsymbol{v}}_i, \tag{2.35} $$

where ${}^{i}\hat{\boldsymbol{I}}_i$ is the spatial inertia of the link, and ${}^{i}\hat{\boldsymbol{v}}_i$ and ${}^{i}\hat{\boldsymbol{a}}_i$ were already computed in the forward pass using Eqs. (2.31) and (2.33), respectively.

The net force acting on the last link of the serial chain can be decomposed into a sum of the externally-applied force ${}^{n}\hat{\boldsymbol{f}}_{n,\text{ext}}$, which includes gravity, and the force ${}^{n}\hat{\boldsymbol{f}}_n$ transmitted across joint J_n (Fig. 2.13):

$$ {}^{n}\hat{\boldsymbol{f}}_{n,\text{net}} = {}^{n}\hat{\boldsymbol{f}}_{n,\text{ext}} + {}^{n}\hat{\boldsymbol{f}}_n. $$

Thus, the force transmitted across J_n is

$$ {}^{n}\hat{\boldsymbol{f}}_n = {}^{n}\hat{\boldsymbol{f}}_{n,\text{net}} - {}^{n}\hat{\boldsymbol{f}}_{n,\text{ext}}. $$

Note that the constraint force $\hat{\boldsymbol{f}}_c$ closing the loop is not included in this equation, as this procedure computes the dynamic terms assuming $\boldsymbol{\tau}_c = \boldsymbol{0}$.

For the remaining links, the net force on link $\mathcal{L}_i$ is the sum of the externally-applied force, and the forces across joints J_i and J_{i+1},

$$ {}^{i}\hat{\boldsymbol{f}}_{i,\text{net}} = {}^{i}\hat{\boldsymbol{f}}_{i,\text{ext}} + {}^{i}\hat{\boldsymbol{f}}_i - {}^{i}\boldsymbol{X}^{*}_{i+1}\;{}^{i+1}\hat{\boldsymbol{f}}_{i+1}, $$

so the force transmitted across joint J_i is

$$ {}^{i}\hat{\boldsymbol{f}}_i = {}^{i}\hat{\boldsymbol{f}}_{i,\text{net}} - {}^{i}\hat{\boldsymbol{f}}_{i,\text{ext}} + {}^{i}\boldsymbol{X}^{*}_{i+1}\;{}^{i+1}\hat{\boldsymbol{f}}_{i+1}. $$

Finally, once the forces exerted by each joint are available, the i-th component of $\boldsymbol{\tau}_{\text{open}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{0})$ is obtained using the reciprocal product

$$ \tau_{\text{open},\, i} = {}^{i}\hat{\boldsymbol{S}}_i^{\top}\;{}^{i}\hat{\boldsymbol{f}}_i. $$

2.4.4 Computing the Mass Matrix

The mass matrix $\boldsymbol{M}(\boldsymbol{q})$ of a serial chain can be computed using the composite-rigid-body algorithm (CRBA) that we next outline [33]. Since all computations are performed using the 0 coordinates, we will often omit the leading superscript 0 for simplicity.

Recall from Section 2.3 that the kinetic energy of a robot can always be expressed as

$$ K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \dot{q}_i\, \boldsymbol{M}_{i,j}\, \dot{q}_j. \tag{2.36} $$

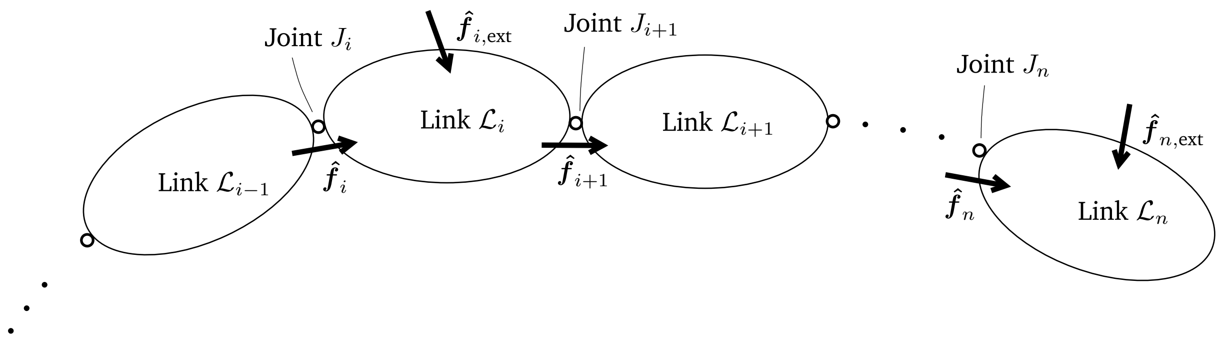


Figure 2.13: Force diagram for the open chain. The net force acting on link $\mathcal{L}_i$ can be decomposed into the sum of the forces $\hat{\boldsymbol{f}}_i$ and $-\hat{\boldsymbol{f}}_{i+1}$ transmitted across joints J_i and J_{i+1}, and the resultant of the externally-applied forces $\hat{\boldsymbol{f}}_{i,\text{ext}}$ (including gravity).

However, since $K(\boldsymbol{q}, \dot{\boldsymbol{q}})$ must be the sum of the kinetic energies of all links, we can also write

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \sum_{k=1}^{n} \hat{\boldsymbol{v}}_k^\top \, \hat{\boldsymbol{I}}_k \, \hat{\boldsymbol{v}}_k, \tag{2.37}$$

where $\hat{\boldsymbol{v}}_k$ and $\hat{\boldsymbol{I}}_k$ are the velocity and spatial inertia of link $\mathcal{L}_k$ in 0 coordinates. The velocity of link $\mathcal{L}_k$, in turn, can be written as

$$\hat{\boldsymbol{v}}_k = \sum_{i=1}^{k} {}^0\hat{\boldsymbol{S}}_1 \dot{q}_i. \tag{2.38}$$

Substituting Eq. (2.38) into Eq. (2.37) gives

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \sum_{k=1}^{n} \sum_{i=1}^{k} \sum_{j=1}^{k} \dot{q}_i \; {}^0\hat{\boldsymbol{S}}_i^\top \, \hat{\boldsymbol{I}}_k \, {}^0\hat{\boldsymbol{S}}_j \; \dot{q}_j,$$

which can be rearranged to obtain

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=\max(i,j)}^{n} \dot{q}_i \; {}^0\hat{\boldsymbol{S}}_i^\top \; \hat{\boldsymbol{I}}_k \; {}^0\hat{\boldsymbol{S}}_j \; \dot{q}_j. \tag{2.39}$$

Comparing Eq. (2.39) with Eq. (2.36), we see that

$$\boldsymbol{M}_{i,j} = {}^0\hat{\boldsymbol{S}}_i^\top \; \underbrace{\left(\sum\nolimits_{k=\max(i,j)}^{n} \hat{\boldsymbol{I}}_k \right)}_{\hat{\boldsymbol{I}}^c_{\max(i,j)}} \; {}^0\hat{\boldsymbol{S}}_j, \tag{2.40}$$

where $\hat{\boldsymbol{I}}^c_{\max(i,j)}$ is the inertia of all links that go from $\max(i,j)$ towards the tip n, treated as a single composite rigid body (hence, the name of the algorithm). This inertia can be computed recursively starting from the tip and going backwards to the base using

$$\hat{\boldsymbol{I}}^c_i = \hat{\boldsymbol{I}}_i + \sum_{j=i}^{n} \hat{\boldsymbol{I}}^c_j, \tag{2.41}$$

with $\hat{\boldsymbol{I}}^c_n = \hat{\boldsymbol{I}}_n$. Both Eqs. (2.40) and (2.41) form the basic equations of the CRBA that recursively computes the mass matrix.

2.4.5 Computing Constraint Forces

As in all mechanisms, the interaction of rigid bodies in a closed-chain system produces constraint forces across the joints that could stress the mechanism excessively. In many applications, such forces will have to remain inside prescribed bounds determined by the resistance of the robot parts, or by working constraints of the robot (e.g., in a cable-driven robot the cable tensions must remain positive in general). Thus, it is important to provide tools to compute the constraint forces in all joints, so we can use them to enforce the mentioned bounds during motion planning.

Given the configuration and velocity vectors $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, and the input vector $\boldsymbol{u}$, our aim is to compute the constraint forces transmitted across each joint of the robot. The procedure will involve three steps:

- Solution of the forward dynamics to obtain the joint acceleration vector $\ddot{\boldsymbol{q}}$.
- Forward propagation of the link velocities and accelerations determined by $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$.
- Backward propagation of the forces transmitted across the joints and extraction of the constraint forces from them.

Although the last steps are similar to those used in Section 2.4.3 to compute $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})$, $\ddot{\boldsymbol{q}}$ is not null now, and we must also include the constraint force $\hat{\boldsymbol{f}}_c$ closing the loop in the backward propagation of forces. We next explain these steps in detail.

Forward Dynamics

The first step consists in solving the forward dynamics of the closed-chain system to obtain the joint acceleration $\ddot{\boldsymbol{q}}$. This is achieved using Eq. (2.12),

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{I}_{n_q} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}(\boldsymbol{q}) & \boldsymbol{J}_n^{\top} \\ \boldsymbol{J}_n & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u}) \\ \boldsymbol{\xi}(\boldsymbol{q},\dot{\boldsymbol{q}}) \end{bmatrix}, \tag{2.42}$$

where $\boldsymbol{J}_n$, $\boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, and the term $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q})$ included in $\boldsymbol{\tau}_{\text{FD}}$ are computed following Section 2.4.3, and $\boldsymbol{M}(\boldsymbol{q})$ is obtained as in Section 2.4.4.

Forward Propagation of Link Velocities and Accelerations

In the second step, the velocities and accelerations of all bodies are recursively computed, starting from the base and working towards the tip. These quantities are expressed in i coordinates as in Section 2.4.3, with the difference that $\ddot{\boldsymbol{q}}$ is now given by Eq. (2.42). Therefore, to obtain ${}^i\hat{\boldsymbol{v}}_i$ and ${}^i\hat{\boldsymbol{a}}_i$ we resort to Eqs. (2.31) and (2.32),

$$\begin{aligned}
{}^i\hat{\boldsymbol{v}}_i &= {}^i\boldsymbol{X}_{i-1}\,{}^{i-1}\hat{\boldsymbol{v}}_{i-1} + {}^i\hat{\boldsymbol{S}}_i\,\dot{q}_i,\\
{}^i\hat{\boldsymbol{a}}_i &= {}^i\boldsymbol{X}_{i-1}\,{}^{i-1}\hat{\boldsymbol{a}}_{i-1} + {}^i\hat{\boldsymbol{S}}_i\,\ddot{q}_i + {}^i\dot{\hat{\boldsymbol{S}}}_i\,\dot{q}_i,
\end{aligned}$$

where $\ddot{q}_i$ is the ith component of the joint acceleration computed in Eq. (2.42).

Backward Propagation of Forces across the Joints

In the third step, the force ${}^i\hat{\boldsymbol{f}}_i$ transmitted across each joint J_i is recursively computed starting from the tip and working backwards towards the base. To this end, we use the Newton-Euler equation in Eq. (2.35) to compute the net force ${}^i\hat{\boldsymbol{f}}_{i,\text{net}}$ acting on link $\mathcal{L}_i$,

$${}^i\hat{\boldsymbol{f}}_{i,\text{net}} = {}^i\hat{\boldsymbol{I}}_i\,{}^i\hat{\boldsymbol{a}}_i + {}^i\hat{\boldsymbol{v}}_i \times {}^i\hat{\boldsymbol{I}}_i\,{}^i\hat{\boldsymbol{v}}_i,$$

where ${}^i\hat{\boldsymbol{v}}_i$ and ${}^i\hat{\boldsymbol{a}}_i$ are those we obtained in the earlier step. In order to find the force transmitted across each joint, note now that, unlike in Section 2.4.3, me must take into account the constraint force $\hat{\boldsymbol{f}}_c$ at the cut link. From Section 2.4.2 we know that such a force is given by

$$\hat{\boldsymbol{f}}_c = -\boldsymbol{\lambda},$$

where

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I}_{n_e} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}(\boldsymbol{q}) & \boldsymbol{J}_n^\top \\ \boldsymbol{J}_n & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ \boldsymbol{\xi} \end{bmatrix},$$

so the spatial force transmitted across joint J_n will be

$${}^n\hat{\boldsymbol{f}}_n = {}^n\hat{\boldsymbol{f}}_{n,\text{net}} - {}^n\hat{\boldsymbol{f}}_{n,\text{ext}} + {}^n\boldsymbol{X}_0^*\,\hat{\boldsymbol{f}}_c.$$

We can now compute the spatial force across the remaining joints working backwards to the base via the recursion

$$^{i}\hat{\boldsymbol{f}}_{i} = {}^{i}\hat{\boldsymbol{f}}_{i,\text{net}} - {}^{i}\hat{\boldsymbol{f}}_{i,\text{ext}} + {}^{i}\boldsymbol{X}^{*}_{i+1}\,{}^{i+1}\hat{\boldsymbol{f}}_{i+1},$$

Note that the total force transmitted across joint J_i is the sum of the constraint force ${}^{i}\hat{\boldsymbol{f}}_{i,\text{constr}}$ and the axial component ${}^{i}\hat{\boldsymbol{f}}_{i,\text{axial}}$ accounting for friction and motor joint forces:

$$^{i}\hat{\boldsymbol{f}}_{i} = {}^{i}\hat{\boldsymbol{f}}_{i,\text{constr}} + {}^{i}\hat{\boldsymbol{f}}_{i,\text{axial}}.$$

Since

$$^{i}\hat{\boldsymbol{f}}_{i,\text{axial}} = {}^{i}\hat{\boldsymbol{S}}_{i}^{\top}\,{}^{i}\hat{\boldsymbol{f}}_{i}\,{}^{i}\hat{\boldsymbol{S}}_{i},$$

the constraint force at joint J_i is finally given by

$$^{i}\hat{\boldsymbol{f}}_{i,\text{constr}} = {}^{i}\hat{\boldsymbol{f}}_{i} - {}^{i}\hat{\boldsymbol{S}}_{i}^{\top}\,{}^{i}\hat{\boldsymbol{f}}_{i}\,{}^{i}\hat{\boldsymbol{S}}_{i}.$$

If necessary, we can always express $\hat{\boldsymbol{f}}_{i,\text{constr}}$ in 0 coordinates using

$$^{0}\hat{\boldsymbol{f}}_{i,\text{constr}} = {}^{0}\boldsymbol{X}^{*}_{i}\,{}^{i}\hat{\boldsymbol{f}}_{i,\text{constr}}.$$

3 Trajectory Planning

We next develop a sampling-based planner for closed-chain robotic systems. After reviewing the state of the art on the topic, we formulate the problem to be solved, and explain why previous sampling-based methods, while powerful, find difficulties in systems with loop-closure and dynamic constraints. The state space of such systems is an implicitly-defined manifold that complicates the sampling and steering procedures, and leads to trajectories that deviate from the manifold if standard integration methods are used. To address these issues, we construct an atlas of the state space incrementally, and use this atlas to generate random states, and to steer the system towards such states. The planner we obtain is probabilistically complete, and it can avoid forward singularities if required. We finally illustrate the planner performance of the planner in significantly complex tasks involving planar and spatial robots that have to lift or throw a load using torque-limited actuators. The planner can be seen as an extension of the one in [41] to deal with closed kinematic chains, or of the one in [9] to deal with dynamic constraints.

3.1 Related work

C-space approaches

The sheer complexity of kinodynamic planning is usually managed by decomposing the problem into simpler subproblems [13]. For example, path planning approaches concentrate on finding a collision-free path in the C-space while satisfying the kinematic constraints [7, 9, 42, 43], which is already a challenging issue by itself. In these approaches, the dynamic constraints of the robot are neglected with the hope they will be enforced in a postprocessing stage. A typical approach for such stage is to resort to time scaling methods [44–49]. These methods regard the path as a function $\boldsymbol{q} = \boldsymbol{q}(s)$ in which $\boldsymbol{q}$ is the robot configuration and s is some path parameter, and then find a monotonic time scaling $s = s(t)$ such that $\boldsymbol{q}(t) = \boldsymbol{q}(s(t))$ connects the start and

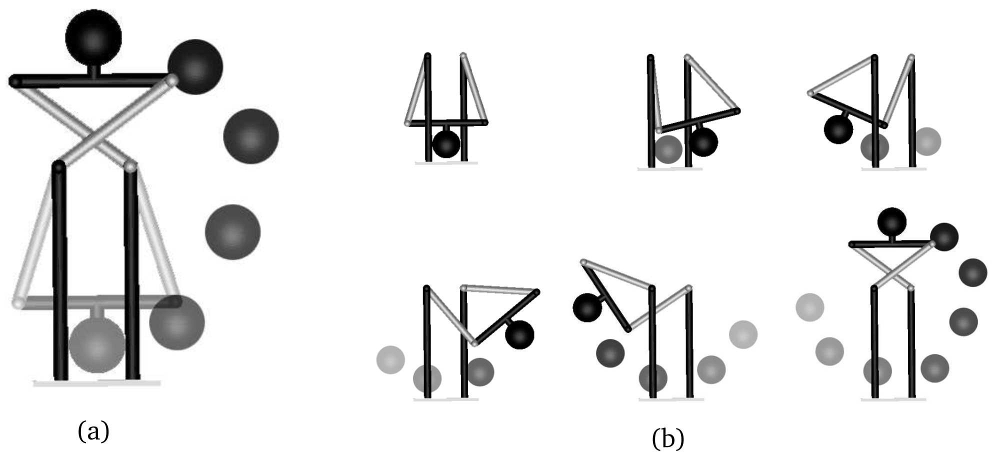


Figure 3.1: A four-bar pendulum modeling a swing boat ride. A kinematic trajectory (a) and a trajectory also fulfilling dynamic constraints due to torque limitation (b) may be quite different. The time-scaling of (a) will not be able to produce the swinging motion in (b). See youtu.be/vhBc6bH8jiw for an animated version of this figure.

goal configurations in minimum time. Although the generated trajectory is only time-optimal for the computed path, this approach makes the trajectory planning problem more tractable, so it can be solved in systems with many degrees of freedom like humanoids, legged robots, or mobile robot formations [50]. Time scaling methods, in addition, have recently been extended to compute the feasible velocities at the end of a path given an initial range of velocities [51], which is useful in the context of randomized planning [50].

Despite their advantages, the previous methods essentially work in the C-space, which makes them limited in some way or another. For instance, path planning approaches are not likely to generate the swinging paths that may be required in highly dynamic tasks like lifting a heavy load under torque limitations. Moreover, it is not difficult to find situations in which a kinematically-feasible path becomes unusable because it does not account for the system dynamics (Fig. 3.1). In other approaches, start or goal states with nonzero velocity cannot be specified, which is necessary in, for example, catching or throwing objects at a certain speed and direction. Time scaling methods, in turn, require the robot to be fully actuated. While this is rarely an issue in robot arms or humanoids under contact constraints [10, 11], parallel robots with passive joints are underactuated at forward singularities. These configurations are problematic when managed in the C-space as they can only be traversed under particular velocities and accelerations, as described in Sections 2.2 and 2.3.3. As it will turn out, however, the previous limitations do not apply if, as we do, robot trajectories are directly planned in the state space.

State space approaches

Since its introduction in [41], the rapidly-exploring random tree (RRT) has emerged as one of the most powerful methods for planning in the state space. Kinodynamic RRT planners are conceptually simple, easy to implement, and can handle dynamic constraints and actuator force limits if desired. They can also solve problems in relatively high dimensions and are probabilistically complete. A main issue of such planners, however, is that exact steering methods are not available for nonlinear dynamical systems. The usual RRT method tries to circumvent this problem by simulating random actions for a given time, and then selecting the action that gets the system closest to the target [41]. For particular systems, better solutions exist though. For instance, the approach in [52] assumes double integrator dynamics and exploits the fact that the minimum-time problem has an efficient solution in this case. The resulting planner is fast, but the full dynamics of the system can only be coped via feedback linearization, which requires the inverse dynamic problem to be solvable. As an alternative, the method in [53] linearizes the system dynamics and uses an infinite-horizon LQR controller to define a steering method, but such a controller can only be used to reach goal states at rest. In contrast, the methods in [54–56] use finite-horizon LQR controllers that can converge to arbitrary states.

Note that, as designed, none of the previous methods can handle closed kinematic chains, as they assume the state coordinates to be independent. The planner we propose in this chapter covers this gap by extending the methods in [41] and [56] to deal with dependencies on the state coordinates.

3.2 The Trajectory Planning Problem

Recall from Section 2.3.2 that the time evolution of a closed-chain system is determined by the system of differential-algebraic equations

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, & \text{(3.1a)} \\ \dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), & \text{(3.1b)} \end{cases}$$

where Eq. (3.1a) is the kinematic constraint that forces the states $\boldsymbol{x}$ to remain in $\mathcal{X}$, and Eq. (3.1b) is the differential constraint that models the robot dynamics. To model the fact that the actuator forces are limited, we assume that $\boldsymbol{u}$ can only take values inside the box

$$\mathcal{U} = [-u_{1,\text{max}}, u_{1,\text{max}}] \times [-u_{2,\text{max}}, u_{2,\text{max}}] \times \ldots \times [-u_{n_u,\text{max}}, u_{n_u,\text{max}}] \tag{3.2}$$

of $\mathbb{R}^{n_u}$, where $u_{i,\text{max}}$ denotes the limit force or torque of the i-th motor. Furthermore, we assume that the robot states $\boldsymbol{x}(t)$ can only evolve inside a region $\mathcal{X}_{\text{feas}} \subseteq \mathcal{X}$ of collision-free states respecting any desired limits on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, and maintaining all constraint forces within admissible bounds.

With the previous definitions, the trajectory planning problem can be stated as follows: Given the kinematic and dynamic models of a robot, a geometric model of the environment, and two states $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$ of $\mathcal{X}_{\text{feas}}$, find a control function $\boldsymbol{u} = \boldsymbol{u}(t)$ such that the trajectory $\boldsymbol{x} = \boldsymbol{x}(t)$ determined by Eqs. (3.1) for $\boldsymbol{x}(0) = \boldsymbol{x}_s$ fulfills $\boldsymbol{x}(t_g) = \boldsymbol{x}_g$ for some time $t_g > 0$, with $\boldsymbol{x}(t) \in \mathcal{X}_{\text{feas}}$ and $\boldsymbol{u}(t) \in \mathcal{U}$ for all $t \in [0, t_g]$.

3.3 Limitations of Prior RRT Methods

Observe that the previous formulation is more general than the one assumed in earlier RRT planners. In particular, the approaches in [9, 42, 57–59] are purely kinematic, so they disregard the dynamics in Eq. (3.1b), and the force bounds in Eq. (3.2). As a result, they only compute paths in $\mathcal{C}$, and such paths might be unfeasible dynamically. In contrast, planning approaches with dynamic constraints like [41, 53, 55, 56, 60, 61] consider Eq. (3.1b) and the bounds in (3.2), but neglect Eq. (3.1a), which impedes the handling of robots with closed kinematic chains. While [13] proposed a few extensions to deal with such chains using RRT methods, we next see that these extensions are problematic in practice.

Recall from [13] that a usual RRT is initialized at $\boldsymbol{x}_s$, and is extended by applying four steps repeatedly (see Fig. 3.2):

1. a guiding state $\boldsymbol{x}_{\text{rand}} \in \mathcal{X}$ is randomly selected;

2. the RRT state $\boldsymbol{x}_{near}$ that is closest to $\boldsymbol{x}_{\text{rand}}$ is determined according to some metric;

3. a steering method is used to compute the action $\boldsymbol{u} \in \mathcal{U}$ that brings the system as close as possible to $\boldsymbol{x}_{\text{rand}}$ in the absence of obstacles;

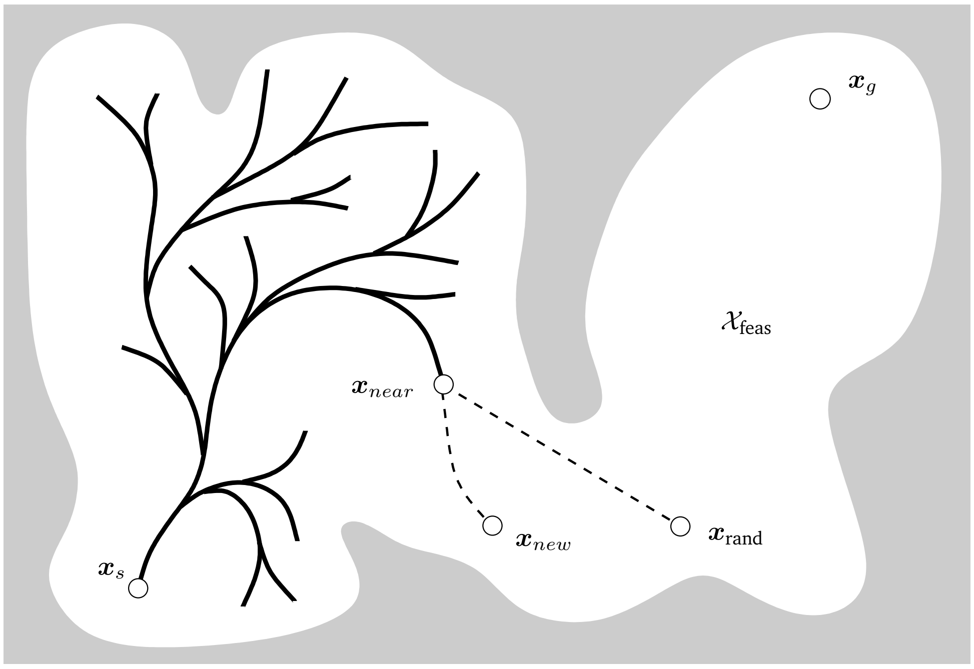


Figure 3.2: Expansion of a unidirectional RRT [41]. First, a random state $\boldsymbol{x}_{\text{rand}} \in \mathcal{X}_{\text{feas}}$ is generated. Then, the state $\boldsymbol{x}_{near}$ that is closest to $\boldsymbol{x}_{\text{rand}}$ is chosen. Finally, a trajectory trying to connect these two states is generated yielding a new state $\boldsymbol{x}_{new}$.

4. the movement that results from applying $\boldsymbol{u}$ during some time Δt is obtained by integrating Eq. (3.1b). This yields a new state $\boldsymbol{x}_{new}$, which is added to the RRT if it lies in $\mathcal{X}_{\text{feas}}$, or it is discarded otherwise. In the former case, $\boldsymbol{u}$ is stored in the new edge connecting $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{new}$. The process terminates when a tree node is close enough to $\boldsymbol{x}_g$.

It is worth noting that, in many implementations, steps 3 and 4 are repeated with $\boldsymbol{x}_{new}$ playing the role of $\boldsymbol{x}_{near}$, as long as $\boldsymbol{x}_{new}$ gets closer to $\boldsymbol{x}_{\text{rand}}$.

Three problems arise when applying the previous method to closed kinematic chains. First, the points $\boldsymbol{x}_{\text{rand}}$ are difficult to obtain in general, as $\mathcal{X}$ may be a manifold without explicit parametrizations. To circumvent this issue, [13, Section 7.4.1] proposes to randomly pick $\boldsymbol{x}_{\text{rand}}$ from the larger ambient space $\mathbb{R}^{n_x}$ (Fig. 3.3) and use, as a guiding state, the point $\boldsymbol{x}'_{\text{rand}}$ that results from projecting $\boldsymbol{x}_{\text{rand}}$ onto the tangent space of $\mathcal{X}$ at $\boldsymbol{x}_{near}$. However, while $\boldsymbol{x}'_{\text{rand}}$ is easy to compute, its pulling effect on the RRT may be small. The ambient space could be large in comparison to $\mathcal{X}$, resulting in points $\boldsymbol{x}'_{\text{rand}}$ that might often be close to $\boldsymbol{x}_{near}$, which diminishes the exploration bias of the RRT. This effect was analyzed in [9] and [42]. A second problem concerns the dynamic simulation of robot motions. Existing RRT methods would only use the dynamics in Eq. (3.1b) to generate such motions on the grounds that the kinematic constraints

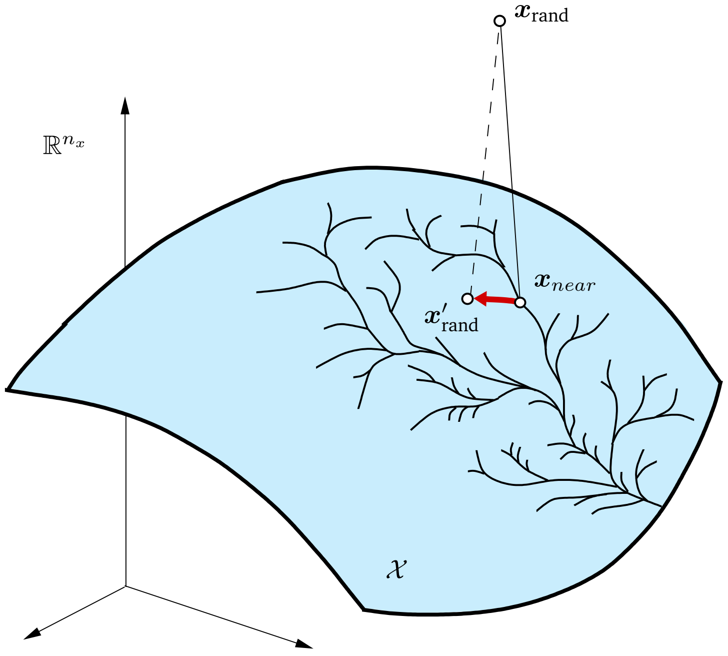

$\boldsymbol{x}_{\text{rand}}$
$\mathbb{R}^{n_x}$
$\boldsymbol{x}_{near}$
$\boldsymbol{x}'_{\text{rand}}$
$\mathcal{X}$

Figure 3.3: Generation of a guiding sample according to [13].

in Eq. (3.1a) are implicitly accounted for by Eq. (3.1b) [13, Section 13.4.3.1]. However, from multibody dynamics it is known that the motion of a closed-chain robot can only be predicted reliably if Eq. (3.1a) is actively used during the integration of Eq. (3.1b) [30, 62]. Otherwise, the inevitable errors introduced when discretizing Eq. (3.1b) will make the trajectory $\boldsymbol{x}(t)$ increasingly drift from $\mathcal{X}$ as the simulation progresses (Fig. 3.4). Such a drift may even be large enough to prevent the connection of $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$. As we will see in next chapter, the use of Baumgarte stabilization to compensate this drift [35] is also problematic, as it may lead to instabilities [63] or fictitious energy increments, and the stabilizing parameters are not easy to tune in general. A third problem, finally, concerns the steering method. A shooting strategy based on simulating several actions from $\mathcal{U}$ was proposed in [41], but this technique is inefficient when n_u is large, as the number of samples needed to properly represent $\mathcal{U}$ grows exponentially with n_u. The lack of a good steering strategy is a general problem of RRT methods, but it is more difficult to address when closed kinematic chains are present.

Purely kinematic planners like [9, 42, 57–59] do not perform dynamic simulations, and employ direct steering motions between configurations. Moreover, most of them sample in ambient space. Thus, the previous three problems would also have to be solved when trying to generalize these planners to deal with dynamic constraints. Among such planners, however, the

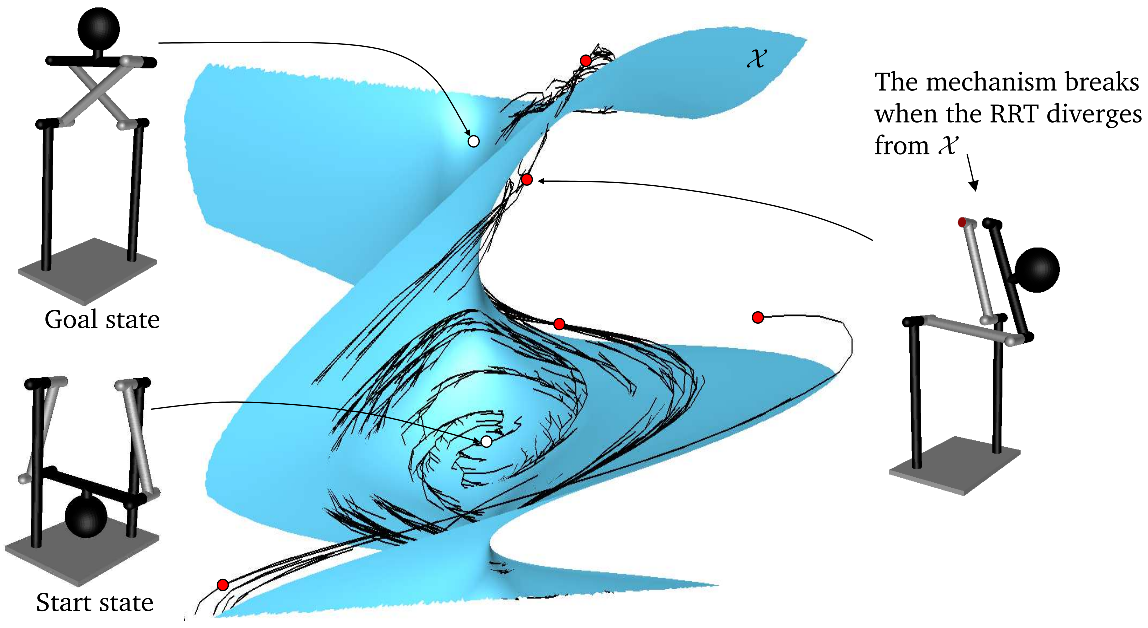

$\mathcal{X}$
The mechanism breaks when the RRT diverges from $\mathcal{X}$
Goal state
Start state

Figure 3.4: Application of the planner in [41] to obtain a dynamically feasible trajectory for a four-bar robot. The pendulum has to be moved from the start to the goal states indicated, which correspond to the white dots in the manifold $\mathcal{X}$ shown in blue. As we see, an RRT built using [41] easily diverges from $\mathcal{X}$ as the exploration proceeds (see the red points for example) which difficults the connection of the query states and generates mechanism disassembles. See youtu.be/nZrfgNN9Jhw for an animated version of this figure.

one in [9] is more amenable for generalization, as it employs atlas machinery that is applicable to mechanisms of general topology. Such a machinery can be extended to tackle the more general problem of Section 3.2. As we shall see, once an atlas of $\mathcal{X}$ is obtained, we will have the necessary tools to

1. sample the $\mathcal{X}$ manifold directly instead of its ambient space $\mathbb{R}^{n_x}$;
2. integrate Eqs. (3.1) as a differential-algebraic equation to ensure driftless motions on $\mathcal{X}$;
3. define a proper steering method for closed kinematic chains.

We develop these tools in the following two sections, and later use them as basic building blocks in our planner implementation.

3.4 Mapping and Exploring the State Space

This section addresses the sampling and drift issues by constructing an atlas of $\mathcal{X}$. We first define this atlas and show how to use it to both sample and integrate Eqs. (3.1) accurately on $\mathcal{X}$ (Section 3.4.1). However, since it may take too long to construct a full atlas, we only build a partial atlas incrementally as the RRT grows (Sections 3.4.2 and 3.4.3).

3.4.1 Atlas Construction

Formally, an atlas of $\mathcal{X}$ is a collection of charts mapping $\mathcal{X}$ entirely (Fig. 3.5(a)), where each chart is a local diffeomorphism φ_c from an open set $V_c \in \mathcal{X}$ to an open set $P_c \in \mathbb{R}^{d_\mathcal{X}}$:

$$\varphi_c : V_c \to P_c.$$

The V_c sets can be thought of as partially-overlapping tiles covering $\mathcal{X}$, in such a way that every $\boldsymbol{x} \in \mathcal{X}$ lies in at least one set V_c. The point $\boldsymbol{y} = \varphi_c(\boldsymbol{x})$ provides the local coordinates, or parameters, of $\boldsymbol{x}$ in chart c. Since each map φ_c is a diffeomorphism, it has an inverse map

$$\psi_c : P_c \to V_c$$

that provides a local parametrization of $V_c \subset \mathcal{X}$. Since we assumed that $\mathcal{X}$ is a smooth manifold, it is guaranteed that φ_c and ψ_c exist for any $\boldsymbol{x}_c \in \mathcal{X}$ [64].

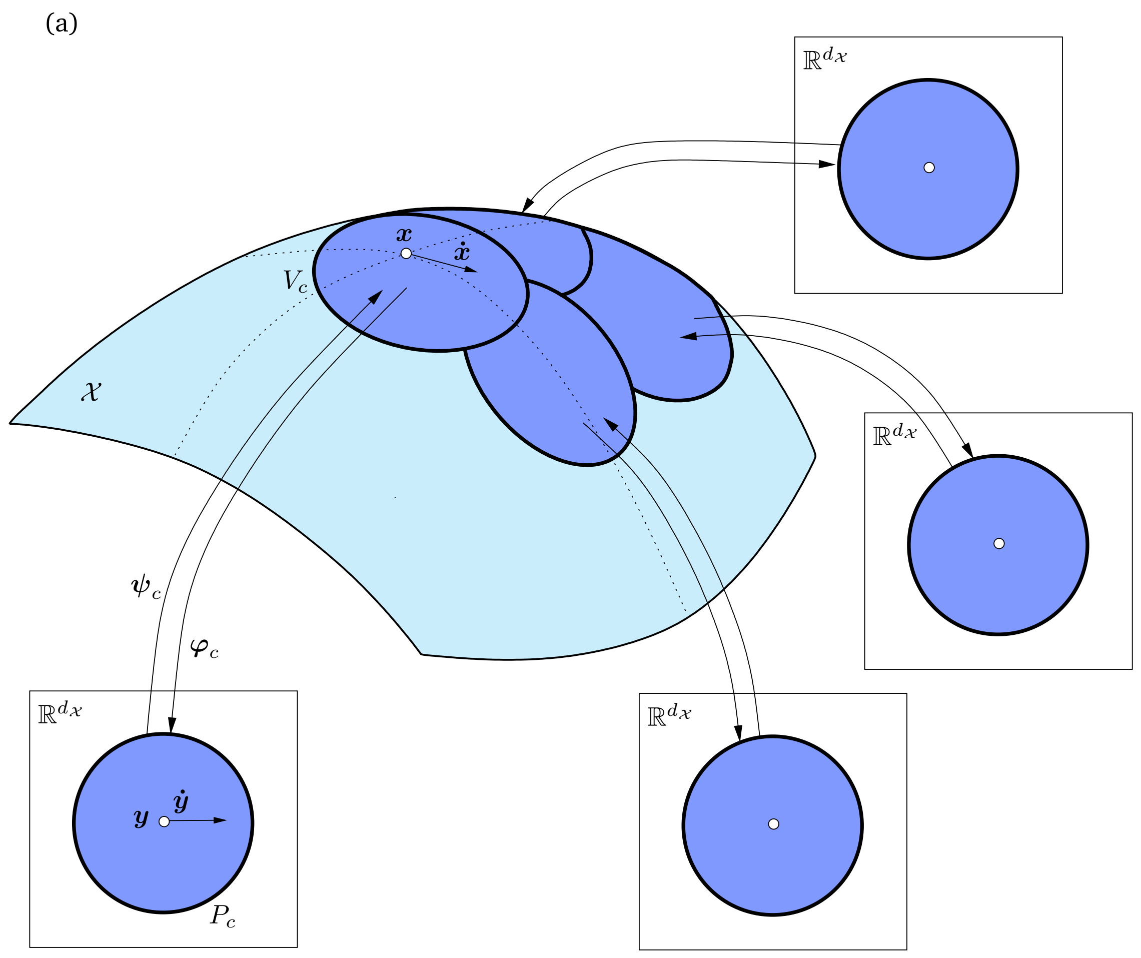
For particular manifolds, φ_c and ψ_c can be defined in closed form (for example, by expressing some variables of Eq. (3.1a) as a function of the others). For the sake of generality, however, we define them using the tangent space parametrization [65], which works for any manifold. Under this parametrization, the map $\boldsymbol{y} = \varphi_c(\boldsymbol{x})$ around a given $\boldsymbol{x}_c \in \mathcal{X}$ is obtained by projecting $\boldsymbol{x}$ orthogonally to $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$ (Fig. 3.5(b)), so this map takes the form

$$\boldsymbol{y} = \boldsymbol{U}_c^\top \, (\boldsymbol{x} - \boldsymbol{x}_c), \tag{3.3}$$

where $\boldsymbol{U}_c$ is an $n_x \times d_\mathcal{X}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. This matrix can easily be computed using the QR decomposition of the Jacobian $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_c)$. The inverse map $\boldsymbol{x} = \psi_c(\boldsymbol{y})$, in turn, is implicitly determined by the system of nonlinear equations

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, \\ \boldsymbol{U}_c^\top \, (\boldsymbol{x} - \boldsymbol{x}_c) - \boldsymbol{y} = \boldsymbol{0}, \end{cases}$$

which, when $\boldsymbol{x}$ is close to $\boldsymbol{x}_c$, can be solved for $\boldsymbol{x}$ using the Newton-Raphson method.



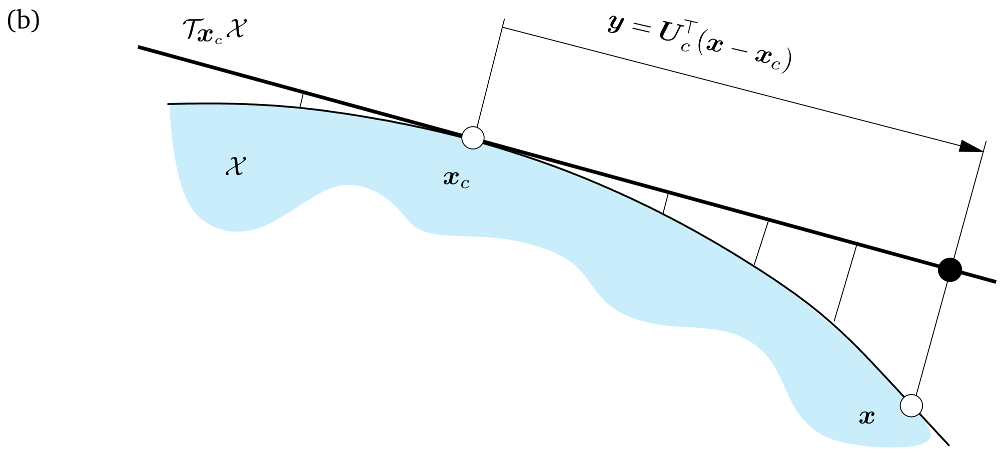

(b)
$\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$
$\boldsymbol{y} = \boldsymbol{U}_c^\top(\boldsymbol{x} - \boldsymbol{x}_c)$
$\mathcal{X}$
$\boldsymbol{x}_c$
$\boldsymbol{x}$

Figure 3.5: (a) A collection of overlapping charts with the corresponding maps φ_c and inverse maps ψ_c. When these charts cover $\mathcal{X}$ entirely, they form an atlas of $\mathcal{X}$. (b) Using the tangent space parametrization, φ_c is defined by the projection of $\boldsymbol{x}$ onto $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$.

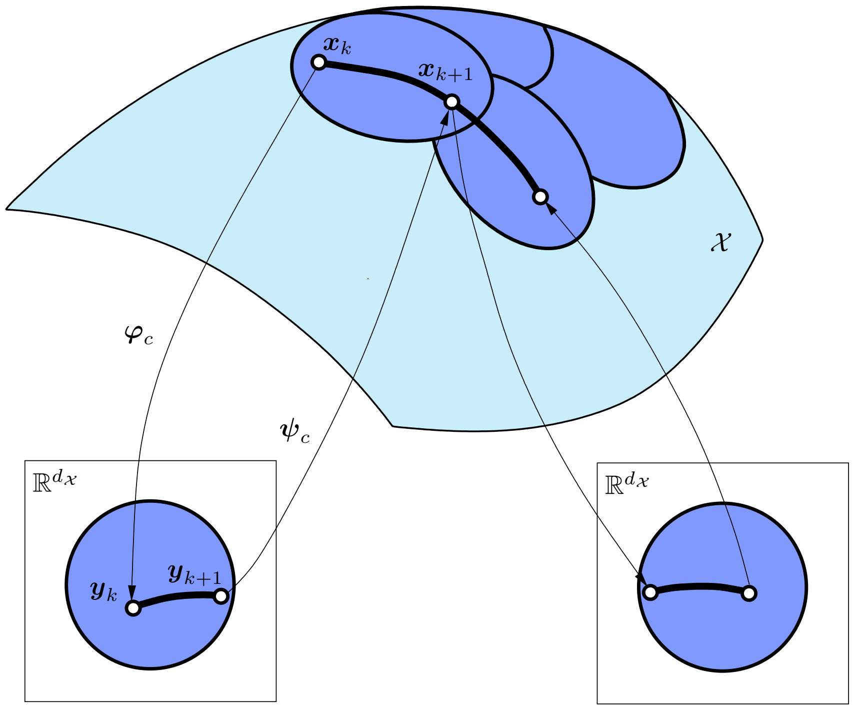


Figure 3.6: To integrate on $\mathcal{X}$, we first obtain the local coordinate of $\boldsymbol{x}_k$ using $\boldsymbol{y}_k = \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$, then compute $\boldsymbol{y}_{k+1}$ by integrating in local coordinates, and finally project $\boldsymbol{y}_{k+1}$ back to $\mathcal{X}$ using $\boldsymbol{x}_{k+1} = \boldsymbol{\psi}_c(\boldsymbol{y}_{k+1})$.

Assuming that an atlas has been created, the problem of sampling $\mathcal{X}$ boils down to generating random points $\boldsymbol{y}_{rand}$ in the P_c sets, as they can always be projected to $\mathcal{X}$ using the map $\boldsymbol{x}_{rand} = \boldsymbol{\psi}_c(\boldsymbol{y}_{rand})$. Also, the atlas allows the conversion of the vector field defined by Eq. (3.1b) into one on the P_c sets of the charts. The time derivative of Eq. (3.3), $\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \dot{\boldsymbol{x}}$, gives the relationship between the two vector fields, and allows writing

$$\dot{\boldsymbol{y}} = \boldsymbol{U}_c^\top \, \boldsymbol{g}(\boldsymbol{\psi}_c(\boldsymbol{y}), \boldsymbol{u}) = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u}), \tag{3.5}$$

which is Eq. (3.1b), but expressed in local coordinates. This equation still takes the full dynamics into account, and forms the basis of geometric methods for the integration of ordinary differential equations on manifolds [25, 26, 30]. Given a state $\boldsymbol{x}_k$ and an action $\boldsymbol{u}_k$, $\boldsymbol{x}_{k+1}$ is estimated by obtaining $\boldsymbol{y}_k = \boldsymbol{\varphi}_c(\boldsymbol{x}_k)$, then computing $\boldsymbol{y}_{k+1}$ using a discrete form of Eq. (3.5), and finally getting $\boldsymbol{x}_{k+1} = \boldsymbol{\psi}_c(\boldsymbol{y}_{k+1})$ (Fig. 3.6). The procedure guarantees that $\boldsymbol{x}_{k+1}$ will lie on $\mathcal{X}$ by construction, thus making the integration compliant with all kinematic constraints in Eq. (3.1a).

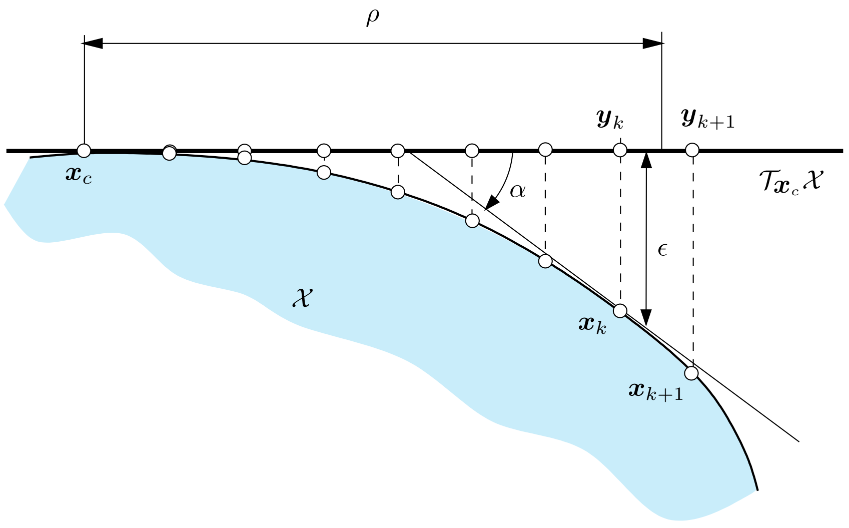


Figure 3.7: Thresholds determining the extension of the P_c set of the chart at $\boldsymbol{x}_c$. While $\boldsymbol{y}_k$ lies in P_c, $\boldsymbol{y}_{k+1}$ does not because it violates Eqs. (3.6)-(3.8).

3.4.2 Incremental Atlas and RRT Expansion

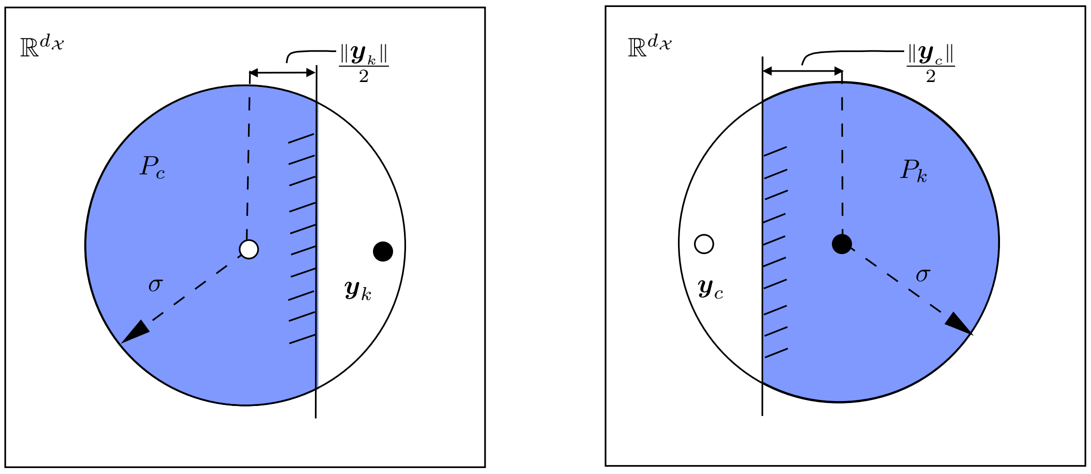
The previous approach relies on the availability of an atlas of $\mathcal{X}$. However, the construction of a full atlas is only feasible for low-dimensional state spaces. On the other hand, only part of the atlas is necessary to solve a trajectory planning problem. For these reasons, as in [9], we combine the construction of the atlas and the expansion of the RRT. In this approach, a partial atlas is used to both generate random states and to expand the RRT branches. As described next, new charts are also created as the RRT branches reach unexplored regions of the state space.

Suppose that $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are two consecutive states along an RRT branch and let $\boldsymbol{y}_k$ and $\boldsymbol{y}_{k+1}$ be their local coordinate vectors in $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. Then, a new chart at $\boldsymbol{x}_k$ is created if the inverse map $\boldsymbol{x}_{k+1} = \psi_c(\boldsymbol{y}_{k+1})$ given by Eq. (3.4) cannot be solved for $\boldsymbol{x}_{k+1}$ using the Newton-Raphson method, or if any of the following conditions is met

$$\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c\, \boldsymbol{y}_{k+1})\| > \epsilon, \tag{3.6}$$

$$\frac{\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\|}{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|} < \cos\alpha, \tag{3.7}$$

$$\|\boldsymbol{y}_{k+1}\| > \rho, \tag{3.8}$$



$\mathbb{R}^{d_\mathcal{X}}$
$\frac{\|\boldsymbol{y}_k\|}{2}$
P_c
σ
$\boldsymbol{y}_k$
$\mathbb{R}^{d_\mathcal{X}}$
$\frac{\|\boldsymbol{y}_c\|}{2}$
P_k
σ
$\boldsymbol{y}_c$

Figure 3.8: Half planes added to trim the P_c and P_k sets of two neighboring charts. Note that $\boldsymbol{y}_k = \varphi_c(\boldsymbol{x}_k)$ and $\boldsymbol{y}_c = \varphi_k(\boldsymbol{x}_c)$.

where ϵ, α, and ρ are user-defined thresholds (Fig. 3.7). These conditions are introduced to ensure that the P_c sets of the created charts capture the overall shape of $\mathcal{X}$ with sufficient detail. The first condition limits the maximal distance between $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$ and the manifold $\mathcal{X}$. The second condition ensures a bounded curvature in the part of $\mathcal{X}$ that is covered by a chart, as well as a smooth transition between neighboring charts. The third condition finally guarantees the generation of new charts as the RRT grows, even for almost flat manifolds.

3.4.3 Chart Coordination

Since the charts will be used to generate samples on $\mathcal{X}$, it is important to reduce the overlap between new charts and those already present in the atlas. Otherwise, the areas of $\mathcal{X}$ covered by several charts would be oversampled. To avoid this problem, the P_c set of each chart is initialized as a ball of radius σ centered at the origin of $\mathbb{R}^{d_\mathcal{X}}$. This ball is progressively bounded as new neighboring charts are created around the chart. If, while growing an RRT branch, a neighboring chart is created at a point $\boldsymbol{x}_k$ with parameter vector $\boldsymbol{y}_k$ in P_c, the following inequality

$$\boldsymbol{y}^\top \boldsymbol{y}_k - \frac{\|\boldsymbol{y}_k\|^2}{2} \leq 0 \tag{3.9}$$

is added as a bounding half-plane of P_c (Fig. 3.8). An analogous inequality is added to the P_k set of the chart at $\boldsymbol{x}_k$, but using $\boldsymbol{y}_c = \varphi_k(\boldsymbol{x}_c)$ instead of $\boldsymbol{y}_k$ in Eq. (3.9). Note that the radius σ of the initial ball must be larger than ρ to guarantee that the RRT branches covered by chart c will eventually trigger the generation of new charts, i.e., to guarantee that Eq. (3.8) will eventually hold. Also, since Eq. (3.8) forces the norm of $\boldsymbol{y}_k$ to be limited by ρ, the half-plane defined by Eq. (3.9) will be guaranteed to clip P_c. Consequently, the P_c sets of those charts surrounded by neighboring charts will be significantly smaller than the P_c sets of the charts at the exploration border of the atlas. As we shall see in Section 3.6.1, this favors the growth of the tree towards unexplored regions of $\mathcal{X}$.

3.5 Steering Methods

The planner can adopt different strategies to steer the system from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{\text{rand}}$. We briefly summarize the standard approach (Section 3.5.1) and then propose an alternative strategy based on linear quadratic regulators (Section 3.5.2).

3.5.1 A Randomized Steering Method

A simple strategy consists in simulating several constant actions from a discrete set $\mathcal{U}_s \subset \mathcal{U}$ during Δt, and then choosing the one that brings the robot closest to $\boldsymbol{x}_{\text{rand}}$ [41]. While the discrete set $\mathcal{U}_s$ can be fixed beforehand (to include, e.g., the zero action that allows the system to move by inertia) it is more common to let $\mathcal{U}_s$ be a set of random actions selected from $\mathcal{U}$ with uniform probability.

The advantage of this approach is that it is simple and works for any system. However, as explained in Section 3.3, it becomes inefficient as the dimension of $\mathcal{U}$ increases, as the number of samples in $\mathcal{U}_s$ needed to properly represent $\mathcal{U}$ grows exponentially with n_u. For instance, if $\mathcal{U}$ is one-dimensional, 10 evenly-spaced actions would suffice to discretize $\mathcal{U}$, whereas an equivalent sampling of a five-dimensional $\mathcal{U}$ would require 10^5 actions. To mitigate the curse of dimensionality, and following [13], we will define $\mathcal{U}_s$ using $2\,n_u$ actions in what follows. If $\mathcal{U}$ is an n_u-dimensional cuboid, this corresponds to just taking the extreme actions for each dimension, while keeping the rest at zero.

3.5.2 An LQR Steering Method

To obtain a steering method that scales better with the problem dimension, we next propose an alternative strategy based on linear quadratic regulators (LQR). While LQR techniques are classical steering methods for control systems [66], they assume the state coordinates to be independent, so they are applicable to open chain robots only. However, we next show that, using the atlas charts, they can be extended to the closed chain case. The idea is to exploit system linearizations at the various chart centers so as to obtain a sequence of control functions $\boldsymbol{u}(t)$ bringing the robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{\text{rand}}$.

System linearization at a chart center

To apply LQR techniques to our steering problem, we will linearize our system model at the chart centers $\boldsymbol{x}_c$ under the null action $\boldsymbol{u} = \boldsymbol{0}$. To do so, note that we cannot linearize Eq. (3.1b), as this would disregard the fact that the $\boldsymbol{x}$ variables are coupled by Eq. (3.1a). We must instead linearize Eq. (3.5), which expresses Eq. (3.1b) in the independent $\boldsymbol{y}$ coordinates of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$. Since $\boldsymbol{x} = \boldsymbol{x}_c$ corresponds to $\boldsymbol{y} = \boldsymbol{0}$ in the local coordinates of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, the sought linearization is

$$\dot{\boldsymbol{y}} = \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{y}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}}}_{\boldsymbol{A}} \boldsymbol{y} + \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}}}_{\boldsymbol{B}} \boldsymbol{u} + \underbrace{\tilde{\boldsymbol{g}}(\boldsymbol{0},\boldsymbol{0})}_{\boldsymbol{c}},$$

which can be written as

$$\dot{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}. \tag{3.10}$$

This system will be assumed to be controllable hereafter.

Observe that, in Eq. (3.10), the term

$$\boldsymbol{c} = \tilde{\boldsymbol{g}}(\boldsymbol{0},\boldsymbol{0}) = \boldsymbol{U}_c^\top \boldsymbol{g}(\boldsymbol{x}_c,\boldsymbol{0})$$

is not null in principle, as $(\boldsymbol{x},\boldsymbol{u}) = (\boldsymbol{x}_c,\boldsymbol{0})$ is not necessarily an equilibrium point of the system in Eq. (3.5). Moreover, by applying the chain rule, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices are given by:

$$\boldsymbol{A} = \left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{y}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}} = \boldsymbol{U}_c^\top \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_c\\ \boldsymbol{u}=\boldsymbol{0}}} \left.\frac{\partial \boldsymbol{\psi}_c}{\partial \boldsymbol{y}}\right|_{\boldsymbol{y}=\boldsymbol{0}}, \tag{3.11}$$

and

$$\boldsymbol{B} = \left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{0}}} = \boldsymbol{U}_c^\top \left.\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{x}=\boldsymbol{x}_c\\ \boldsymbol{u}=\boldsymbol{0}}}.$$

It is easy to see that, in Eq. (3.11),

$$\frac{\partial \boldsymbol{\psi}_c}{\partial \boldsymbol{y}} = \boldsymbol{U}_c.$$

To realize so, consider the parametrization

$$\boldsymbol{y} = \boldsymbol{U}_c^\top (\boldsymbol{x} - \boldsymbol{x}_c). \tag{3.12}$$

If we use the inverse mapping $\boldsymbol{x} = \boldsymbol{\psi}(\boldsymbol{y})$ to rewrite Eq. (3.12) as

$$\boldsymbol{y} = \boldsymbol{U}_c^\top \left(\boldsymbol{\psi}_c(\boldsymbol{y}) - \boldsymbol{x}_c\right), \tag{3.13}$$

and compute the partial derivative of both sides of Eq. (3.13) with respect to $\boldsymbol{y}$, we have

$$\boldsymbol{I} = \boldsymbol{U}_c^\top \frac{\partial \boldsymbol{\psi}_c}{\partial \boldsymbol{y}},$$

which, upon multiplication by $\boldsymbol{U}_c$, and using $\boldsymbol{U}_c^\top \boldsymbol{U}_c = \boldsymbol{I}$, yields $\partial \boldsymbol{\psi}_c / \partial \boldsymbol{y} = \boldsymbol{U}_c$ as we claimed.

Notice, therefore, that $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{c}$ can exactly be obtained by evaluating the original function $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ and its derivatives $\partial \boldsymbol{g}/\partial \boldsymbol{x}$ and $\partial \boldsymbol{g}/\partial \boldsymbol{u}$ at $(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x}_c, \boldsymbol{0})$. In those systems in which these derivatives are not easy to obtain in closed form, $\boldsymbol{A}$ and $\boldsymbol{B}$ can always be approximated numerically using finite differences, or by using automatic differentiation [67].

Steering on a single chart

Suppose now that both $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{\text{rand}}$ lie in a same chart c centered at $\boldsymbol{x}_c \in \mathcal{X}$ (Fig. 3.9). In this case, the problem of steering the robot from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{\text{rand}}$ can be reduced to that of steering the system in Eq. (3.10) from $\boldsymbol{y}_{near} = \boldsymbol{\varphi}_c(\boldsymbol{x}_{near})$ to $\boldsymbol{y}_{\text{rand}} = \boldsymbol{\varphi}_c(\boldsymbol{x}_{\text{rand}})$. This problem can be formulated as follows: Find the control function $\boldsymbol{u}(t) = \boldsymbol{u}^*(t)$ and time $t_f = t_f^*$ that minimize the cost function

$$J(\boldsymbol{u}(t), t_f) = \int_0^{t_f} \left(1 + \boldsymbol{u}(t)^\top \boldsymbol{R}\, \boldsymbol{u}(t)\right) \mathrm{d}t, \tag{3.14}$$

subject to the constraints

$$\begin{aligned} \dot{\boldsymbol{y}} &= \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}, \\ \boldsymbol{y}(0) &= \boldsymbol{y}_{near}, \\ \boldsymbol{y}(t_f) &= \boldsymbol{y}_{\text{rand}}. \end{aligned} \tag{3.15}$$

In Eq. (3.14), the unit term inside the integral penalizes large values of t_f, while the term $\boldsymbol{u}(t)^\top \boldsymbol{R}\ \boldsymbol{u}(t)$ penalizes high control actions. In this term, $\boldsymbol{R}$ is a symmetric positive-definite matrix that is fixed beforehand.

The problem just formulated is known as the fixed final state optimal control problem [66]. We shall solve this problem in two stages. Initially, we will obtain $\boldsymbol{u}^*(t)$ assuming that t_f is fixed, and then we will find a time t_f that leads to a minimum of $J(\boldsymbol{u}(t), t_f)$.

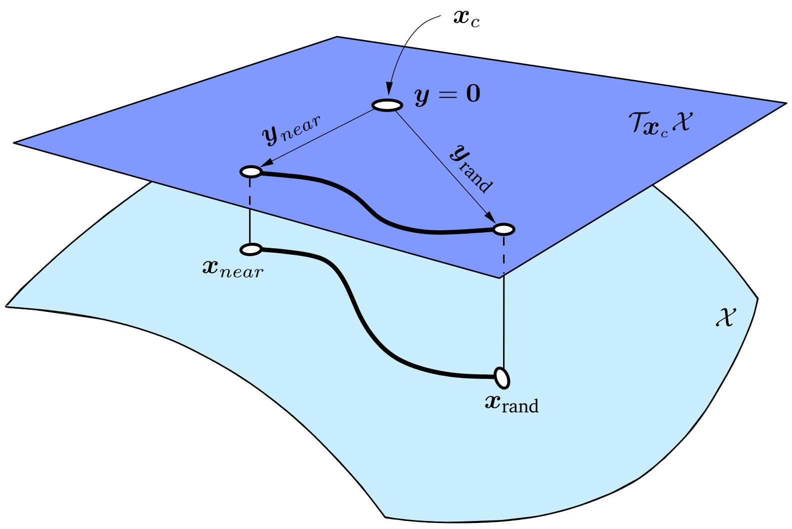

$\boldsymbol{x}_c$
$\boldsymbol{y} = \mathbf{0}$
$\boldsymbol{y}_{near}$
$\boldsymbol{y}_{\text{rand}}$
$\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$
$\boldsymbol{x}_{near}$
$\mathcal{X}$
$\boldsymbol{x}_{\text{rand}}$

Figure 3.9: When $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{\text{rand}}$ are covered by a same chart, the steering of the system can be reduced to a steering problem in $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$.

Fixed final state and fixed final time problem

If t_f is fixed, we can find the optimal action $\boldsymbol{u}(t) = \boldsymbol{u}^*(t)$ by applying Pontryagin's minimum principle. Since the function $\boldsymbol{u}^\top(t)\,\boldsymbol{R}\,\boldsymbol{u}(t)$ is convex, this principle provides necessary and sufficient conditions of optimality in our case [68]. To apply the principle, we first define the Hamiltonian function

$$H(\boldsymbol{y},\boldsymbol{u},\boldsymbol{\lambda}) = 1 + \boldsymbol{u}^\top\ \boldsymbol{R}\ \boldsymbol{u} + \boldsymbol{\lambda}^\top\ (\boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c})\,,$$

where $\boldsymbol{\lambda} = \boldsymbol{\lambda}(t)$ is an undetermined Lagrange multiplier. Then, the corresponding state and costate equations [66] are

$$\dot{\boldsymbol{y}} = \frac{\partial H}{\partial \boldsymbol{\lambda}}^\top = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{c}, \tag{3.16}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{y}}^\top = -\boldsymbol{A}^\top\boldsymbol{\lambda}. \tag{3.17}$$

For $\boldsymbol{u} = \boldsymbol{u}^*(t)$ to be an optimal control function, H must be at a stationary point relative to $\boldsymbol{u}$, i.e., it must be

$$\left.\frac{\partial H}{\partial \boldsymbol{u}}\right|^\top_{\boldsymbol{u}=\boldsymbol{u}^*(t)} = \boldsymbol{R}\ \boldsymbol{u}^*(t) + \boldsymbol{B}^\top\boldsymbol{\lambda} = \mathbf{0},$$

and thus,

$$\boldsymbol{u}^*(t) = -\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{\lambda}(t). \tag{3.18}$$

Since Eq. (3.17) is decoupled from Eq. (3.16), its solution can be found independently. It is

$$\boldsymbol{\lambda}(t) = \mathrm{e}^{\boldsymbol{A}^\top (t_f - t)}\boldsymbol{\lambda}(t_f), \tag{3.19}$$

where $\boldsymbol{\lambda}(t_f)$ is still unknown.

To find $\boldsymbol{\lambda}(t_f)$, let us consider the closed-form solution of Eq. (3.16) for $\boldsymbol{u} = \boldsymbol{u}^*(t)$:

$$\boldsymbol{y}(t) = \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{y}(0) + \int_0^t \mathrm{e}^{\boldsymbol{A}(t-\tau)} \left(\boldsymbol{B}\boldsymbol{u}^*(\tau) + \boldsymbol{c}\right) \,\mathrm{d}\tau.$$

If we evaluate this solution for $t = t_f$ and take into account Eqs. (3.18) and (3.19), we arrive at the expression

$$\boldsymbol{y}(t_f) = \boldsymbol{r}(t_f) - \boldsymbol{G}_\mathrm{r}(t_f)\, \boldsymbol{\lambda}(t_f), \tag{3.20}$$

where

$$\boldsymbol{r}(t_f) = \mathrm{e}^{\boldsymbol{A}t_f}\, \boldsymbol{y}(0) + \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}(t_f-\tau)}\, \boldsymbol{c}\, \mathrm{d}\tau, \tag{3.21}$$

and

$$\begin{aligned}\boldsymbol{G}_\mathrm{r}(t_f) &= \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}(t_f-\tau)}\, \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\, \mathrm{e}^{\boldsymbol{A}^\top (t_f-\tau)}\, \mathrm{d}\tau \\ &= \int_0^{t_f} \mathrm{e}^{\boldsymbol{A}\tau}\, \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\, \mathrm{e}^{\boldsymbol{A}^\top \tau}\, \mathrm{d}\tau. \end{aligned} \tag{3.22}$$

Given that $\boldsymbol{y}(t_f)$ is known from Eq. (3.15), we can solve Eq. (3.20) for $\boldsymbol{\lambda}(t_f)$ to obtain

$$\boldsymbol{\lambda}(t_f) = \boldsymbol{G}_\mathrm{r}(t_f)^{-1} \left(\boldsymbol{r}(t_f) - \boldsymbol{y}(t_f)\right). \tag{3.23}$$

Now, substituting Eq. (3.23) into (3.19), and the result into Eq. (3.18), we finally obtain the optimal control function for the fixed final state and fixed final time problem:

$$\boldsymbol{u}^*(t) = -\boldsymbol{R}^{-1}\boldsymbol{B}^\top \mathrm{e}^{\boldsymbol{A}^\top (t_f-t)}\, \boldsymbol{G}_\mathrm{r}(t_f)^{-1} \left(\boldsymbol{r}(t_f) - \boldsymbol{y}(t_f)\right). \tag{3.24}$$

Note that this is an open-loop control law, as $\boldsymbol{u}^*$ depends on t only. The values $\boldsymbol{r}(t_f)$ and $\boldsymbol{G}_\mathrm{r}(t_f)$ in Eq. (3.24) can be obtained by computing the integrals in Eqs. (3.21) and (3.22) numerically. The matrix $\boldsymbol{G}_\mathrm{r}(t_f)$ is known as the weighted continuous reachability Gramian, and since the system is controllable, it is symmetric and positive-definite for $t > 0$ [56], which ensures that $\boldsymbol{G}_\mathrm{r}(t_f)^{-1}$ always exists.

Finding the optimal time t_f

To find a time t_f^* for which the cost J in Eq. (3.14) attains a minimum value, we substitute the optimal control in Eq. (3.24) into Eq. (3.14), and take into account Eq. (3.22), obtaining

$$J(t_f) = t_f + \big(\boldsymbol{y}(t_f) - \boldsymbol{r}(t_f)\big)^\top \boldsymbol{G}_{\mathrm{r}}(t_f)^{-1} \big(\boldsymbol{y}(t_f) - \boldsymbol{r}(t_f)\big). \tag{3.25}$$

The time t_f^* is thus the one that minimizes $J(t_f)$ in Eq. (3.25). Assuming that t_f^* lies inside a specified time window $[0, t_{\max}]$, this time can be computed approximately by evaluating $\boldsymbol{r}(t_f)$, $\boldsymbol{G}_{\mathrm{r}}(t_f)$ and $J(t_f)$ using Eqs. (3.21), (3.22), and (3.25) for $t_f = 0$ to $t_f = t_{\max}$, and selecting the t_f value for which $J(t_f)$ is minimum.

Finally, the values t_f^*, $\boldsymbol{r}(t_f^*)$, and $\boldsymbol{G}_{\mathrm{r}}(t_f^*)$ can be used to evaluate the optimal control function in Eq. (3.24). By applying such a control to the full nonlinear system of Eq. (3.1b) during t_f^* seconds, we will follow a trajectory ending in some state $\boldsymbol{y}'_{\text{rand}}$ close to $\boldsymbol{y}_{\text{rand}}$. This trajectory can be recovered on the $\mathcal{X}$ space by means of the ψ_c map and, if it lies in $\mathcal{X}_{\text{feas}}$, the corresponding branch can be added to the RRT.

Steering over multiple charts

If $\boldsymbol{x}_{\text{rand}}$ is not covered by the chart c of $\boldsymbol{x}_{near}$, we can iteratively apply the steering process as shown in Fig. 3.10(a). To this end, we compute $\boldsymbol{y}_{\text{rand}} = \varphi_c(\boldsymbol{x}_{\text{rand}})$ and drive the system from $\boldsymbol{y}_{near} = \varphi_c(\boldsymbol{x}_{near})$ towards $\boldsymbol{y}_{\text{rand}}$ on $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, projecting the intermediate states $\boldsymbol{y}$ to $\mathcal{X}$ via ψ_c. Eventually, we will reach some state $\boldsymbol{x}_k \in \mathcal{X}$ that is in the limit of the V_c set of the current chart (see the conditions in Section 3.4.2). At this point, we generate a chart at $\boldsymbol{x}_k$ and linearize the system again. We then use this linearization to recompute the optimal control function to go from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{\text{rand}}$. Such a “linearize and steer" process can be repeated as needed, until the system gets closely enough to $\boldsymbol{x}_{\text{rand}}$.

Although the previous procedure is often effective, it can also fail in some situations. As shown in Fig. 3.10(b), the initial steering on chart c might bring the system from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_k$ but, due to the position of $\boldsymbol{x}_{\text{rand}}$, a new control function computed at $\boldsymbol{x}_k$ would steer the system back to $\boldsymbol{x}_{near}$, leading to a back-and-forth cycle not converging to $\boldsymbol{x}_{\text{rand}}$. Such limit cycles can be detected however, because the time t_f^* will no longer decrease eventually. As shown in Fig. 3.10(c), moreover, the steering procedure can sometimes reach $\boldsymbol{y}_{\text{rand}}$, but we might find that $\psi_c(\boldsymbol{y}_{\text{rand}}) \neq \boldsymbol{x}_{\text{rand}}$ because, due to the curvature of $\mathcal{X}$, several states can project to the same point on a given tangent space. Even so, such situations do not prevent the connection of $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$, as the steering algorithm is to be used inside a higher-level RRT planner. The implementation of such a planner is next addressed.

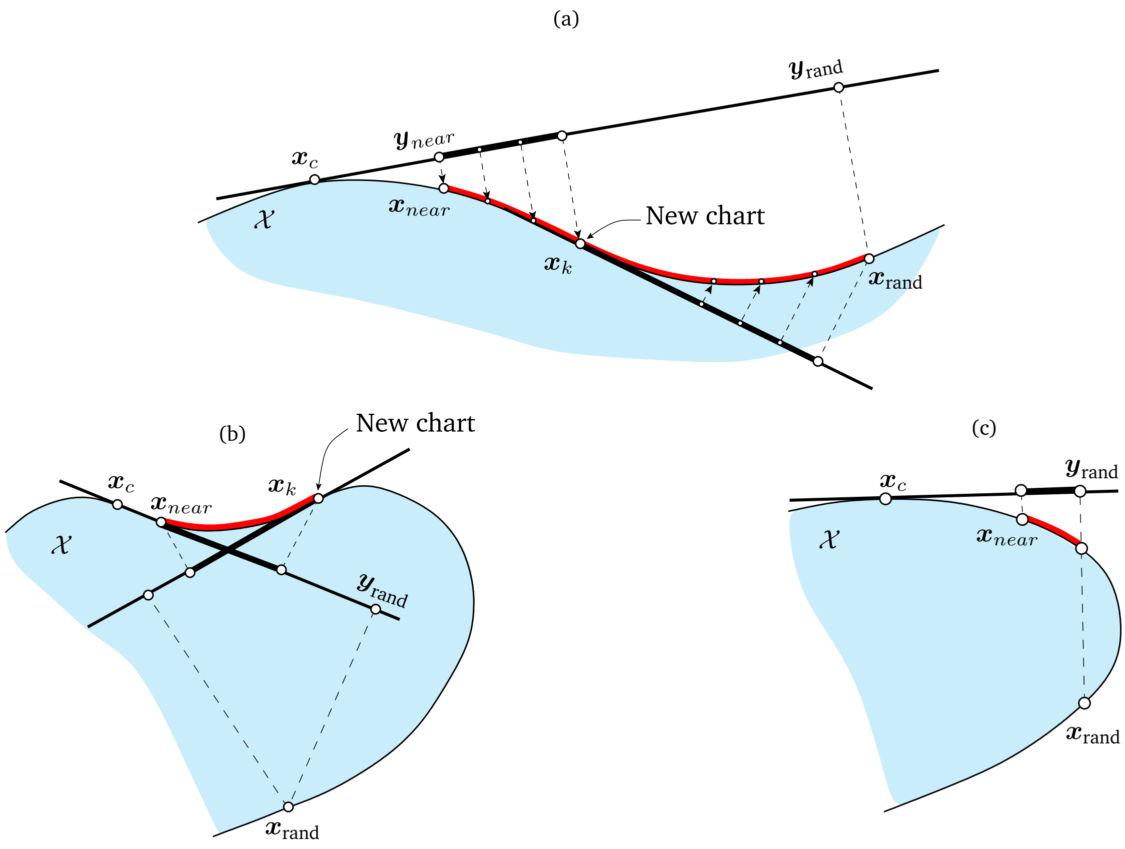


Figure 3.10: (a) Steering towards states not covered by the chart of $\boldsymbol{x}_{near}$. (b) Cyclic behavior of the steering method. (c) Convergence to $\boldsymbol{y}_{\text{rand}}$ but not to $\boldsymbol{x}_{\text{rand}}$.

3.6 Planner Implementation

Algorithm 3.1 gives the top-level pseudocode of the planner. At this level, the algorithm is almost identical to the RRT planner in [41]. The only difference is that, in our case, we construct an atlas A of $\mathcal{X}$ to support the lower-level sampling, simulation, and steering tasks. As in [41], the algorithm implements a bidirectional RRT where a tree T_s is rooted at $\boldsymbol{x}_s$ (line 2) and another tree T_g is rooted at $\boldsymbol{x}_g$ (line 3). Consequently, the atlas is initialized with one chart centred at $\boldsymbol{x}_s$ and another chart centered at $\boldsymbol{x}_g$ (line 1). Initially, a random state is sampled ($\boldsymbol{x}_{\text{rand}}$ in line 5), the nearest state in T_s is determined ($\boldsymbol{x}_{near}$ in line 6), and then T_s is extended with the aim of connecting $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{\text{rand}}$ using the CONNECT method (line 7). This method reaches a state $\boldsymbol{x}_{new}$ that, due to the presence of obstacles, or to the particular features of the steering method used, may be different from $\boldsymbol{x}_{\text{rand}}$. Then, the state in T_g that is nearest to $\boldsymbol{x}_{new}$ is determined ($\boldsymbol{x}'_{near}$ in line 8) and T_g is extended from $\boldsymbol{x}'_{near}$ with the aim of reaching $\boldsymbol{x}_{new}$ (line 9).

Algorithm 3.1: The top-level pseudocode of the planner

```
PLAN TRAJECTORY($\boldsymbol{x}_s, \boldsymbol{x}_g$)
input : The query states, $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$.
output: A trajectory connecting $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$.
1  $A \leftarrow$ INITATLAS($\boldsymbol{x}_s, \boldsymbol{x}_g$)
2  $T_s \leftarrow$ INITRRT($\boldsymbol{x}_s$)
3  $T_g \leftarrow$ INITRRT($\boldsymbol{x}_g$)
4  repeat
5      $\boldsymbol{x}_{\text{rand}} \leftarrow$ SAMPLE($A, T_s$)
6      $\boldsymbol{x}_{near} \leftarrow$ NEARESTSTATE($T_s, \boldsymbol{x}_{\text{rand}}$)
7      $\boldsymbol{x}_{new} \leftarrow$ CONNECT($A, T_s, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}$)
8      $\boldsymbol{x}'_{near} \leftarrow$ NEARESTSTATE($T_g, \boldsymbol{x}_{new}$)
9      $\boldsymbol{x}'_{new} \leftarrow$ CONNECT($A, T_g, \boldsymbol{x}'_{near}, \boldsymbol{x}_{new}$)
10     SWAP($T_s, T_g$)
11 until $\|\boldsymbol{x}_{new} - \boldsymbol{x}'_{new}\| < \beta$
12 RETURN(TRAJECTORY($T_s, \boldsymbol{x}_{new}, T_g, \boldsymbol{x}'_{new}$))
```

Algorithm 3.2: Generate a random state $\boldsymbol{x}_{\text{rand}}$.

```
SAMPLE($A, T$)
input : The atlas $A$ and the tree $T$ to be extended.
output: A guiding sample $\boldsymbol{x}_{\text{rand}}$.
1 repeat
2     $c \leftarrow$ RANDOMCHARTINDEX($A, T$)
3     $\boldsymbol{y}_{\text{rand}} \leftarrow$ RANDOMONBALL($\sigma$)
4 until $\boldsymbol{y}_{rand} \in P_c$
5 $\boldsymbol{x}_{\text{rand}} \leftarrow \boldsymbol{\psi}_c(\boldsymbol{y}_{\text{rand}})$
6 if $\boldsymbol{x}_{rand} =$ NULL then
7     $\boldsymbol{x}_{\text{rand}} \leftarrow \boldsymbol{x}_c + \boldsymbol{U}_c\, \boldsymbol{y}_{\text{rand}}$
8 RETURN($\boldsymbol{x}_{\text{rand}}$)
```

This extension generates a new state $\boldsymbol{x}'_{new}$. After this step, the trees are swapped (line 10) and, if the last connection was unsuccessful, i.e., if $\boldsymbol{x}_{new}$ and $\boldsymbol{x}'_{new}$ are not closer than a user-provided threshold (line 11), lines 5 to 10 are repeated again. If the connection was successful, a solution trajectory is reconstructed using the paths from $\boldsymbol{x}_{new}$ and $\boldsymbol{x}'_{new}$ to the roots of T_s and T_g (line 12). Due to the particularities of the two steering methods, the CONNECT method is given in Algorithm 3.3 or 3.4 for the randomized or LQR steering, respectively. Different metrics can be used to determine the distance between two states without affecting the overall structure of the planner. As in [41], we use Euclidean distance for simplicity.

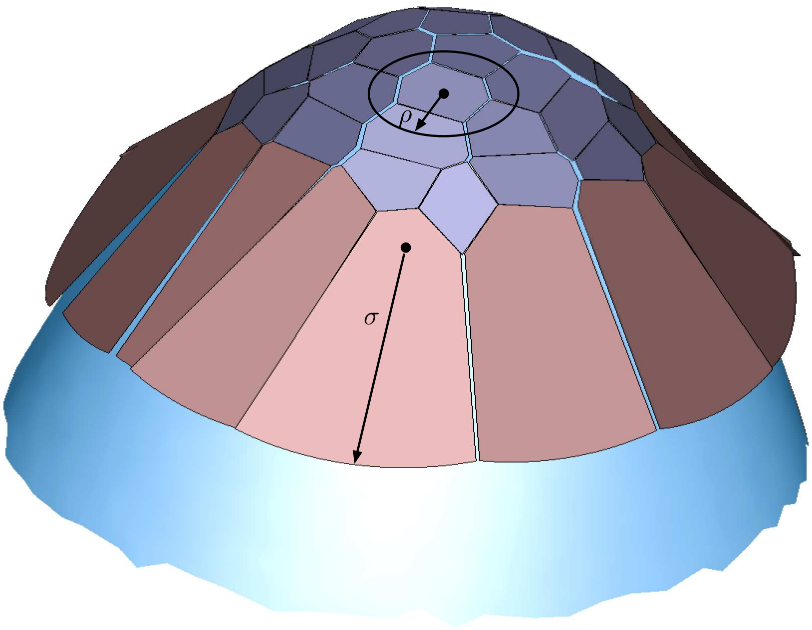


Figure 3.11: A partial atlas of a paraboloid, with its inner and border charts colored in blue and red, respectively. Black dots indicate chart centers.

3.6.1 Sampling

The SAMPLE method is described in Algorithm 3.2. Initially, one of the charts covering the tree T is selected at random with uniform distribution (line 2). A vector $\boldsymbol{y}_{\text{rand}}$ of parameters is then randomly sampled also with uniform distribution inside a ball of radius σ centred at the origin of $\mathbb{R}^{d_{\mathcal{X}}}$ (line 3). Chart selection and parameter sampling are repeated until $\boldsymbol{y}_{\text{rand}}$ falls inside the P_c set for the selected chart. This process generates a sample $\boldsymbol{y}_{\text{rand}}$ with uniform distribution over the union of the P_c sets covering T. Note here that the P_c set of a chart in the interior of the atlas is included in a ball of radius ρ, while the P_c set of a chart at the border of the atlas is included inside a ball of radius $\sigma > \rho$ (Fig. 3.11). If we fix ρ but increase σ the overall volume of the border charts increases, whereas that of the inner charts stays constant. Therefore, by increasing σ we can increase the exploration bias of the algorithm. This bias is analogous to the Voronoi bias in standard RRTs [69]. After generating a valid sample, the method then attempts to compute the point $\boldsymbol{x}_{\text{rand}} = \boldsymbol{\psi}_c(\boldsymbol{y}_{\text{rand}})$ (line 5) and returns this point if the Newton method implementing $\boldsymbol{\psi}_c$ is successful (line 8). Otherwise, it returns the ambient space point corresponding to $\boldsymbol{y}_{\text{rand}}$ (line 7). This point lies on $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$, instead of on $\mathcal{X}$, but it still provides a guiding direction to steer the tree towards unexplored regions of $\mathcal{X}$.

Algorithm 3.3: Try to connect $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{\text{rand}}$ with randomized steering.

```
ConnectRandom($A, T, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}$)
input : An atlas $A$, a tree $T$, the state $\boldsymbol{x}_{near}$ from which $T$ is to be extended towards $\boldsymbol{x}_{\text{rand}}$.
output: The new state $\boldsymbol{x}_{new}$.
1  $d_{ref} \leftarrow \|\boldsymbol{x}_{near} - \boldsymbol{x}_{\text{rand}}\|$
2  repeat
3      $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{near}$
4      $d_{new} \leftarrow \infty$
5      foreach $\boldsymbol{u} \in \mathcal{U}_s$ do
6          $(\boldsymbol{x}_{sim}, \boldsymbol{u}_{sim}) \leftarrow$ Simulate($A, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}, \boldsymbol{u}, \Delta t$)
7          if $\boldsymbol{x}_{sim} \in \mathcal{X}_{feas}$ then
8              $d \leftarrow \|\boldsymbol{x}_{sim} - \boldsymbol{x}_{\text{rand}}\|$
9              if $d < d_{new}$ then
10                 $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{sim}$
11                 $\boldsymbol{u}_{new} \leftarrow \boldsymbol{u}_{sim}$
12                 $d_{new} \leftarrow d$
13     if $\boldsymbol{x}_{new} \neq \boldsymbol{x}_{near}$ then
14         $T \leftarrow$ AddEdge($T, \boldsymbol{x}_{near}, \boldsymbol{u}_{new}, \boldsymbol{x}_{new}$)
15     if $d_{new} \leq d_{ref}$ then
16         $\boldsymbol{d}_{ref} \leftarrow \boldsymbol{d}_{new}$
17         $\boldsymbol{x}_{near} \leftarrow \boldsymbol{x}_{new}$
18 until $d_{new} > d_{ref}$
19 Return($\boldsymbol{x}_{new}$)
```

3.6.2 Tree Extension

Algorithm 3.3 attempts to connect a state $\boldsymbol{x}_{near}$ to a state $\boldsymbol{x}_{\text{rand}}$ using randomized steering. The procedure simulates for a fixed time Δt the motion of the system (line 6) for a set $\mathcal{U}_s$ of constant control policies, which can either be selected at random or be defined beforehand (line 5). The action that yields a new state $\boldsymbol{x}_{new}$ closer to $\boldsymbol{x}_{\text{rand}}$ is added to the RRT with an edge connecting it to $\boldsymbol{x}_{near}$ (line 14). The action $\boldsymbol{u}_{new}$ generating the transition from $\boldsymbol{x}_{near}$ to the new state $\boldsymbol{x}_{new}$ is also stored in the tree so that an action trajectory can be returned after planning. This process is repeated as long as there is progress towards $\boldsymbol{x}_{\text{rand}}$.

The analogous procedure using LQR steering is given in Algorithm 3.4. The algorithm implements a loop where, initially, the optimal control policy $\boldsymbol{u}^*(t)$ and time t_f^* to connect these two states are computed (line 4). The control is a function of time given by Eq. (3.24). If t_f^* is lower than the optimal time t_{fp}^* obtained in the previous iteration, the control is used to simulate the evolution of the system from $\boldsymbol{x}_{near}$ (line 7). The simulation produces a state $\boldsymbol{x}_{new}$ which, if

Algorithm 3.4: Try to connect $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{\text{rand}}$ with LQR steering.

```
ConnectLQR($A, T, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}$)
input  : An atlas $A$, a tree $T$, the state $\boldsymbol{x}_{near}$ from which $T$ is to be extended towards $\boldsymbol{x}_{\text{rand}}$.
output: The new state $\boldsymbol{x}_{new}$.
1  $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{near}$
2  $t^*_{fp} \leftarrow \infty$
3  repeat
4      $(\boldsymbol{u}^*, t^*_f) \leftarrow$ LQRControl($A, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}$)
5      if $t^*_f \leq t^*_{fp}$ then
6          $t^*_{fp} \leftarrow t^*_f$
7          $(\boldsymbol{x}_{new}, \boldsymbol{u}_{new}) \leftarrow$ Simulate($A, \boldsymbol{x}_{near}, \boldsymbol{x}_{\text{rand}}, \boldsymbol{u}^*(t), t^*_f$)
8          if $\boldsymbol{x}_{new} \in \mathcal{X}_{feas}$ and $\boldsymbol{x}_{new} \neq \boldsymbol{x}_{near}$ then
9              $T \leftarrow$ AddEdge($T, \boldsymbol{x}_{near}, \boldsymbol{u}_{new}, \boldsymbol{x}_{new}$)
10             $\boldsymbol{x}_{near} \leftarrow \boldsymbol{x}_{new}$
11 until $t^*_f > t^*_{fp}$ or $\|\boldsymbol{\varphi}_c(\boldsymbol{x}_{new}) - \boldsymbol{\varphi}_c(\boldsymbol{x}_{rand})\| \leq \delta$ or $\boldsymbol{x}_{new} \notin \mathcal{X}_{feas}$
12 $\boldsymbol{x}_{new} \leftarrow \boldsymbol{x}_{near}$
13 Return($\boldsymbol{x}_{new}$)
```

it is feasible and different from $\boldsymbol{x}_{near}$, it is added to the tree. This involves the creation of an edge between $\boldsymbol{x}_{near}$ and $\boldsymbol{x}_{new}$ (line 9), which stores the control sequence $\boldsymbol{u}_{new}$ executed in the simulation. The loop is repeated until t^*_f is larger than t^*_{fp} (line 11), or $\boldsymbol{x}_{\text{rand}}$ is reached with accuracy δ in parameter space, or the next state is unfeasible.

Algorithm 3.5 summarizes the procedure used to simulate a given control policy $\boldsymbol{u}(t)$ from a particular state $\boldsymbol{x}_k$. The simulation progresses while the new state is valid, the target state is not reached with accuracy δ in parameter space, and the integration time t is lower than t_{sim} (line 6). A state is not valid if is not in $\mathcal{X}_{\text{feas}}$ due to obstacles or constraint forces out of bounds (line 10). In that case, both the simulation and the connection between states are stopped. In the case of non-constant policies, when the simulation reaches a new chart, the simulation is stopped, but the connection continues after recomputing the optimal control, either on a newly created chart (line 15) or on the neighboring chart.

Only when using LQR steering, a state is also not valid if it is not in the validity area of the chart (line 17), or is not included in the current P_c set (line 24), i.e., it is parameterized by a neighboring chart. In these two cases the simulation is stopped, but the connection continues after recomputing the optimal control, either on a newly created chart (line 15) or on the neighboring chart, respectively.

Algorithm 3.5: Simulate an action.

```
SIMULATE($A, \boldsymbol{x}_k, \boldsymbol{x}_{\text{rand}}, \boldsymbol{u}(t), t_{sim}$)
input  : An atlas $A$, the state $\boldsymbol{x}_k$ from where the simulation starts, the state $\boldsymbol{x}_{\text{rand}}$ to be
         approached, the control policy $\boldsymbol{u}(t)$ to be applied, the simulation time $t_{sim}$.
output: The last state in the simulation and the executed control sequence.

1  $t \leftarrow 0$
2  $\boldsymbol{y}_k \leftarrow \varphi_c(\boldsymbol{x}_k)$
3  $\boldsymbol{u}_k \leftarrow \emptyset$
4  $c \leftarrow$ CHARTINDEX($\boldsymbol{x}_{near}$)
5  STOP $\leftarrow$ FALSE
6  while NOT STOP and $\|\varphi_c(\boldsymbol{x}_k) - \varphi_c(\boldsymbol{x}_{rand})\| > \delta$ and $|t| < t_{sim}$ do
7      $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, h) \leftarrow$ NEXTSTATE($\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{u}(t), \boldsymbol{F}, \boldsymbol{x}_c, \boldsymbol{U}_c, \delta$)
8      if $\boldsymbol{x}_{k+1} \notin \mathcal{X}_{feas}$ then
9          $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
10         STOP $\leftarrow$ TRUE
11     else
12         if $\|\boldsymbol{x}_{k+1} - (\boldsymbol{x}_c + \boldsymbol{U}_c\,\boldsymbol{y}_{k+1})\| > \epsilon$ or
13            $\|\boldsymbol{y}_{k+1} - \boldsymbol{y}_k\| / \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \cos(\alpha)$ or
14            $\|\boldsymbol{y}_{k+1}\| > \rho$ then
15             $c \leftarrow$ ADDCHARTTOATLAS($A, \boldsymbol{x}_k$)
16             if NOT CONSTANT($\boldsymbol{u}$) then
17                 STOP $\leftarrow$ TRUE
18         else
19             $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k+1}$
20             $\boldsymbol{u}_k \leftarrow \boldsymbol{u}_k \cup \{(\boldsymbol{u}(t), h)\}$
21             $t \leftarrow t + h$
22             if $\boldsymbol{y}_{k+1} \notin P_c$ then
23                 if NOT CONSTANT($\boldsymbol{u}$) then
24                     STOP $\leftarrow$ TRUE
25                 else
26                     $c \leftarrow$ NEIGHBORCHART($A, c, \boldsymbol{y}_{k+1}$)
27 RETURN($\boldsymbol{x}_k, \boldsymbol{u}_k$)
```

The key procedure in the simulation is the NEXTSTATE method (line 7), which provides the next state $\boldsymbol{x}_{k+1}$, given the current state $\boldsymbol{x}_k$ and the action $\boldsymbol{u}(t)$ at time t. The elements of $\boldsymbol{u}(t)$ are saturated to their bounds in Eq. (3.2) if such bounds are surpassed. Then, the simulation is implemented by integrating the dynamics in local coordinates as explained in Section 3.4.1. Any numerical integration method could be used to discretize Eq. (3.5), either explicit or implicit. We here apply the trapezoidal rule as it yields an implicit integrator whose computational cost (integration and projection to the manifold) is similar to the cost of using an explicit method of the same order [30]. Using this rule, Eq. (3.5) is discretized as

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \frac{h}{2}\boldsymbol{U}_c^\top\left(\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u})\right), \tag{3.26}$$

where h is the integration time step. The value $\boldsymbol{x}_{k+1}$ in Eq. (3.26) is unknown but, since the $\boldsymbol{\psi}_k$ map is defined implicitly by Eq. (3.4), it must fulfill

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}_{k+1}) = \boldsymbol{0}, \\ \boldsymbol{U}_c^\top(\boldsymbol{x}_{k+1} - \boldsymbol{x}_c) - \boldsymbol{y}_{k+1} = \boldsymbol{0}. \end{cases}$$

Now, substituting Eq. (3.26) into Eq. (3.27) we obtain

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}_{k+1}) = \boldsymbol{0}, \\ \boldsymbol{U}_c^\top\left(\boldsymbol{x}_{k+1} - \frac{h}{2}(\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{u}) + \boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{u})) - \boldsymbol{x}_c\right) - \boldsymbol{y}_k = \boldsymbol{0}, \end{cases}$$

where $\boldsymbol{x}_k$, $\boldsymbol{y}_k$, and $\boldsymbol{x}_c$ are known values, and $\boldsymbol{x}_{k+1}$ is the unknown to be determined. We could use a Newton method to solve this system, but the Broyden method is preferable as it avoids the computation of the Jacobian of the system at each step. Potra and Yen [30] gave an approximation of this Jacobian that allows finding $\boldsymbol{x}_{k+1}$ in only a few iterations. For backward integration, i.e., when extending the RRT with root at $\boldsymbol{x}_g$, the time step h in Eq. (3.28) must simply be negative.

3.6.3 Setting the Planner Parameters

The planner depends on eight parameters: the three parameters ϵ, α, and ρ controlling chart creation, the radius σ used for sampling, the tolerances δ and β measuring closeness between states and trees, respectively, and the LQR steering parameters $\boldsymbol{R}$ and t_{max}. All of them are positive reals except $\boldsymbol{R}$, which must be an $n_u \times n_u$ symmetric positive-definite matrix.

Parameters ϵ, α, and ρ appear, respectively, in Eqs. (3.6), (3.7), and (3.8). Parameter α bounds the angle between neighboring charts. This angle should be small, otherwise the V_c sets for neighboring charts might not overlap, impeding a smooth transition between the charts [65]. Such problematic areas can be detected and patched [9], but this process introduces inefficiencies. Thus, we suggest to keep this parameter below $\pi/6$. Parameter ϵ is only relevant if the distance between the manifold and the tangent space becomes large without a significant change in curvature, which rarely occurs. Since this distance is computed in ambient space, if on average we wish to tolerate an error of e in each dimension, we should set $\epsilon \simeq e\sqrt{n_x}$. In our test cases we have used $\epsilon = 0.05\,\sqrt{n_x}$. Finally, ρ must be set by taking into account the curvature of the manifold [65] and it must be smaller than σ to ensure the eventual creation of new charts. In practice, it only plays a relevant role on almost flat manifolds. Following [9], we suggest to set $\rho = d_{\mathcal{X}}/2$. With this value, charts are generally created before the numerical process implementing the inverse map ψ_k in Eq. (3.4) fails and before Eqs. (3.6) and (3.7) hold. In this way, the charting of the manifold tends to be more regular.

As explained in Section 3.6.1, the sampling radius σ used in line 3 of Algorithm 3.2 controls the exploration bias of the algorithm. The role of σ is analogous to that of the parameter used in standard RRTs to limit the sampling space (e.g., the boundaries of a 2D space where a mobile robot is set to move). A too large σ complicates the solution of problems with narrow corridors. Thus, we propose to set $\sigma = 2\,\rho$ since this a moderate value that still creates a strong push towards unexplored regions, specially in large-dimensional state spaces. If necessary, existing techniques to automatically tune this parameter [70] could be adapted to kinodynamic planning.

Parameter δ appears in line 11 of Algorithm 3.4 and in line 6 of Algorithm 3.5. An equivalent parameter is present in the standard RRT algorithm [41]. If two states are closer than δ, they are considered to be close enough so that the transition between them is not problematic. Thus, this parameter is used as an upper bound of the distance between consecutive states along an RRT branch. Therefore, the value of h in Eq. (3.28) is adjusted so that $\|\boldsymbol{\varphi}_c(\boldsymbol{x}_k) - \boldsymbol{\varphi}_c(\boldsymbol{x}_{k+1})\| < \delta$. Moreover, to correctly detect the transition between charts, δ must be significantly smaller than ρ. With these considerations in mind, we propose to set $\delta \simeq 0.02\,\rho$.

Parameter β appears in line 11 of Algorithm 3.1 and is the tolerated error in the connection between trees. This parameter is also used in the standard RRT algorithm. A small value may unnecessarily complicate some problems, specially if the steering algorithm is not very precise (like in randomized steering), and a large value may produce unfeasible solutions. We suggest to use $\beta = 0.1\sqrt{n_x}$, but this value has to be tuned according to the particularities of the obstacles in the environment.

Matrix $\boldsymbol{R}$ in Eq. (5.22) is used in the standard LQR to penalize the control effort employed and is typically initialized using the Bryson rule [71]. Finally, t_{max} fixes the time window over which $J(t_f)$ in Eq. (3.25) is to be minimized. Ideally, it should be slightly larger than t_f^*. A much larger value would result in a waste of computational resources and a too low value would produce sub-optimal controls.

3.7 Probabilistic Completeness

In its fully randomized version, i.e., when using randomized steering instead of LQR steering, the planner is probabilistically complete. A formal proof of this point would replicate the same arguments used in [72] with minor adaptations, so we only sketch the main points supporting the claim.

Assume that the action to execute is selected at random from $\mathcal{U}$, with a random time horizon. Then, in the part of $\mathcal{X}$ already covered by a partial atlas, we are in the same situation as the one considered in [72, Section IV]: $\mathcal{X}$ is a smooth manifold, we have a procedure to sample $\mathcal{X}$, Euclidean distance is used to determine nearest neighbors, and the system motion is governed by a differential equation depending on the state and the control inputs. The main relevant difference is that our sample distribution is uniform in tangent space, but not on $\mathcal{X}$. However, the difference between the uniform distribution in parameter space and the actual distribution on the manifold is bounded by parameter α [73]. Thus, the probability bounds given in [72] may need to be modified, but their proof would still hold. Thus, under the same mild conditions assumed in [72] (i.e., Lipschitz-continuity conditions), our planner with randomized steering is probabilistically complete in, at least, the part of the manifold already covered by the atlas. This implies that the planner will be probabilistically complete provided it is able to extend the atlas to cover $\mathcal{X}$ completely. Since new charts are generated when the RRT branches reach the border of the subset of $\mathcal{X}$ covered up to a given moment, the expansion of the atlas will stop when the atlas has no border, i.e., when it fully covers $\mathcal{X}$. The reasoning in [72] can also be used to provide a formal proof that the tree will eventually reach the border regions of the atlas just by defining goal areas in them. As described in [65], $\mathcal{X}$ will be correctly covered if ρ is small relative to the curvature of the manifold in each V_c set. Despite the chart coordination procedure described in Section 3.4.3 may leave uncovered areas of $\mathcal{X}$ of size $O(\alpha)$, such areas can be detected during tree extension, and can be eliminated by slightly enlarging the P_c sets of the charts around them, as described in [9].

In principle, the use of LQR steering instead of randomized steering can only result in better performance, as it should facilitate the connection between the balls used in [72, Theorem 2] to cover the solution trajectory: connecting them using LQR steering should be easier than doing so with randomized steering. However, a formal proof of this point would require to provide error bounds for the LQR steering controls analogous to those in [72, Lemma 3] for randomly-selected constant actions. As in [72], the obtention of such bounds remains as an open problem, so we only conjecture the planner to be probabilistically complete if LQR steering is used. Even so, note we could always retain the probabilistic completeness by using randomized steering once in a while, instead of using LQR steering exclusively.

3.8 Dealing with Forward Singularities

As mentioned in Sections 2.2 and 2.3.3, systems with closed kinematic chains can exhibit forward singularities, which are configurations in which the velocities of the actuators do not determine the full configuration velocity of the robot. Such singularities also have an impact on the system dynamics. In their proximity, the inverse dynamic problem yields very large or unbounded motor actions, which can cause controllability issues [17, 19, 74, 75]. Moreover, the robot becomes locally underactuated, being unable to traverse these configurations under arbitrary accelerations.

The planner in Section 3.6, however, is robust to forward singularities. The exploration process and the steering methods in Section 3.5 rely on the forward dynamics given by Eq. (3.1a), which is always well defined as $\boldsymbol{\Phi}_{\boldsymbol{q}}$ was assumed to be always full rank. Therefore, the trajectories obtained by the planner are kinematically and dynamically feasible even in such configurations, which are naturally crossed using feasible action from $\mathcal{U}$ always.

Once a trajectory has been planned, however, a control algorithm must be used to execute it on the real robot, and it is at this point that forward singularities may become an issue. Broadly speaking, controllers can be classified into those that are based on inverse dynamics, and those that are based on forward dynamics. The former controllers are interesting due to their global basins of attraction, but they cannot cope with forward singularities because the inverse dynamics problem degenerates at such configurations and therefore, they cannot stabilize trajectories obtained by the planner in principle. In contrast, the latter controllers have smaller basins of attraction, but they are robust to forward singularities and thus can stabilize any of the trajectories obtained by the planner (see Chapter 5 for details). Nonetheless, if we prefer to use controllers based on inverse dynamics due to their attractive convergence properties, we can still extend the formulation in Section 3.2 to avoid forward singularities during the planning stage.

The approach relies on a system of equations characterizing the singularity-free state space of the robot. The solution set of this system is a smooth manifold diffeomorphic to the classic state space, but with all forward singularities removed. We also adapt the dynamic equations of the robot to define an action-varying vector field on this manifold. This extended formulation substitutes Eqs. (3.1a) and (3.1b) and allows us to use the planner in Section 3.6 to find trajectories connecting two states, while avoiding forward singularities. We illustrate it using a five-bar robot that has to perform a highly dynamic task in Section 3.9.2.

3.8.1 Planning in the Singularity-free State Space

Recall from Section 2.2 that a forward singularity is a point $\boldsymbol{q}$ for which the Jacobian $\boldsymbol{\Phi}_{\boldsymbol{q}_r}$ is rank deficient. To exclude these singularities from $\mathcal{X}$, thus, we need to enforce $\det(\boldsymbol{\Phi}_{\boldsymbol{q}_r}) \neq 0$ if the system is nonredundant, or $\det\left(\boldsymbol{\Phi}_{\boldsymbol{q}_r}^{\top} \boldsymbol{\Phi}_{\boldsymbol{q}_r}\right) \neq 0$ if the system is redundantly actuated. However, the continuation method used by the planner requires to convert this condition into equality form. To this end, we introduce the constraint

$$b \cdot d(\boldsymbol{q}) = 1, \tag{3.29}$$

where b is a newly-defined auxiliary variable and $d(\boldsymbol{q}) = \det(\boldsymbol{\Phi}_{\boldsymbol{q}_r})$ or $d(\boldsymbol{q}) = \det\left(\boldsymbol{\Phi}_{\boldsymbol{q}_r}^{\top} \boldsymbol{\Phi}_{\boldsymbol{q}_r}\right)$ in the case of nonredundant and redundant systems, respectively. Note that Eq. (3.29) defines a hyperbola that never crosses the $d = 0$ axis of the (b, d) plane. Then, the singularity-free state space $\mathcal{X}_{\text{sfree}}$ will be the set of extended states $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}}, b)$ satisfying

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{\Phi}(\boldsymbol{q}) \\ \boldsymbol{\Phi}_{\boldsymbol{q}}\, \dot{\boldsymbol{q}} \\ b \cdot d(\boldsymbol{q}) - 1 \end{bmatrix} = \boldsymbol{0}, \tag{3.30}$$

which includes the singularity avoidance constraint in Eq. (3.29) and the position and velocity constraints in Eqs. (2.1) and (2.2).

We now need to extend the vector field defined by the dynamics in Eq. (3.1b) to include the auxiliary variable b. This can be done by taking the time derivative of Eq. (3.29) to yield

$$\dot{b} = -\frac{b}{d(\boldsymbol{q})} \cdot \frac{\partial d(\boldsymbol{q})}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}},$$

so the extended vector field becomes

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}) \\ -\frac{b}{d(\boldsymbol{q})} \cdot \frac{\partial d(\boldsymbol{q})}{\partial \boldsymbol{q}} \dot{\boldsymbol{q}} \end{bmatrix}. \tag{3.31}$$

Finally, we can simply substitute Eqs. (3.1a) and (3.1b) with Eqs. (3.30) and (3.31) to plan in the singularity-free state space $\mathcal{X}_{\text{sfree}}$.

The effect of the previous extensions is illustrated in Fig. 3.12. On the left figure we represent $\mathcal{X}$ as a plane whose forward singularity locus is a closed curve. Without using the singularity-avoidance constraints, the planner may find the red or green trajectories to go from $\boldsymbol{x}_s$ to $\boldsymbol{x}_g$. On the right figure, in contrast, the planner uses the singularity-avoidance formulation, so the singularity locus becomes a surface that extends infinitely along the b dimension (a cylinder)

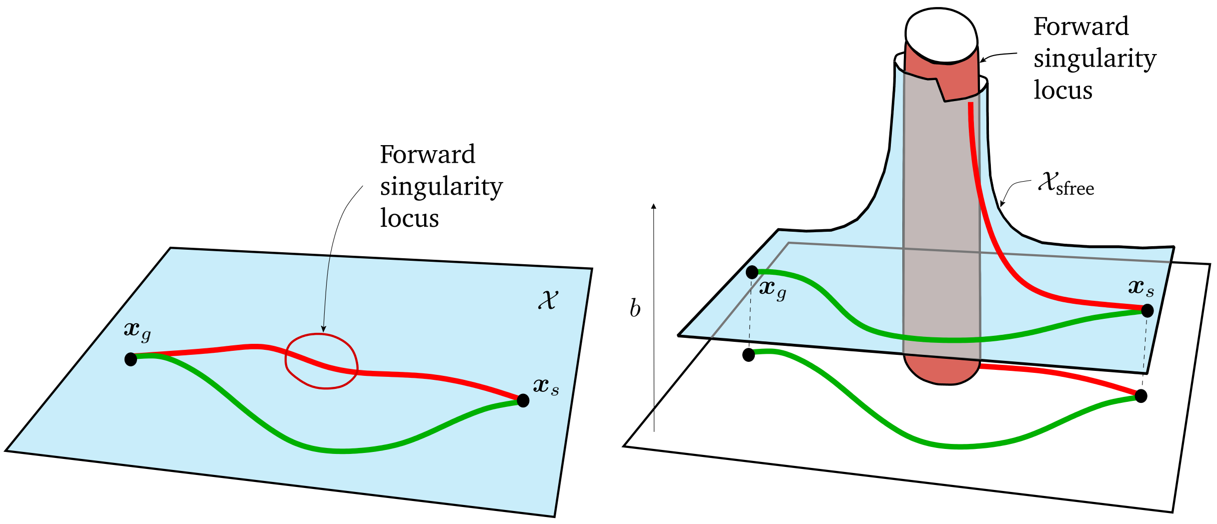


Figure 3.12: Singularity-avoidance planning. Left: The planning on the state space manifold $\mathcal{X}$ defined by Eq. (3.1a) can yield trajectories either crossing (red) or avoiding (green) forward singularities ($\mathcal{X}$ is drawn as a plane for simplicity). Right: If we use the auxiliary variable b and the singularity avoidance formulation in Eq. (3.30), the new singularity-free manifold $\mathcal{X}_{\text{sfree}}$ will go to infinity at the singularity. In this scenario, the planner will only find solutions avoiding forward singularities (green).

without intersecting with $\mathcal{X}$. In this situation the planner will only generate singularity-free motions like the green curve, as motions like the red curve do not lead to feasible connections with $\boldsymbol{x}_g$. The use of singularity avoidance constraints is illustrated in Section 3.9.2 with an example.

Note that the previous procedure can also be used to avoid C-space singularities if necessary. We just need to ensure $\boldsymbol{\Phi}_{\boldsymbol{q}}$ is full rank by defining $d(\boldsymbol{q})$ as $\det\left(\boldsymbol{\Phi}_{\boldsymbol{q}}\ \boldsymbol{\Phi}_{\boldsymbol{q}}^{\top}\right)$.

3.8.2 Smoothness of $\mathcal{X}_{\text{sfree}}$

Recall that, to be able to construct the charts required by the planner, $\mathcal{X}_{\text{sfree}}$ must be a smooth manifold. To see that this is indeed the case, observe that all functions defining $\boldsymbol{F}(\boldsymbol{x})$ are differentiable, so the Jacobian

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} = \boldsymbol{F}_{\boldsymbol{q},\dot{\boldsymbol{q}},b} = \left[\begin{array}{c|c|c} \boldsymbol{\Phi}_{\boldsymbol{q}} & \mathbf{0} & \mathbf{0} \\ \hline \dfrac{\partial \boldsymbol{\Phi}_{\boldsymbol{q}}}{\partial \boldsymbol{q}} \cdot \dot{\boldsymbol{q}} & \boldsymbol{\Phi}_{\boldsymbol{q}} & \mathbf{0} \\ \hline b \cdot \frac{\partial d}{\partial \boldsymbol{q}} & \mathbf{0} & d \end{array}\right]$$

is well defined at every $\boldsymbol{x} \in \mathcal{X}_{\text{sfree}}$. This Jacobian, moreover, is full rank for all $\boldsymbol{x} \in \mathcal{X}_{\text{sfree}}$ because each of the diagonal blocks contains a non-vanishing minor of maximal size, since $d \neq 0$ for all $\boldsymbol{x} \in \mathcal{X}_{\text{sfree}}$, and $\boldsymbol{\Phi}_{\boldsymbol{q}}$ is guaranteed to be full rank on $\mathcal{X}_{\text{sfree}}$. By the implicit function theorem, these facts imply that $\mathcal{X}_{\text{sfree}}$ is smooth.

3.9 Planning Examples

The planner has been implemented in C and it has been integrated into the CUIK suite [76]. We next analyze its performance in planning five tasks of different complexities (Figs. 3.13 and 3.14). For each task, the figures show the start and goal states to be connected. The first three tasks are to be solved with planar robots (Fig. 3.13), which are simple enough to illustrate key aspects of the planner, like the performance of the steering method, the traversal or avoidance of forward singularities, or the treatment of constraint forces. The fourth and fifth tasks show the planner performance in computing collision-free motions for spatial robots of considerable complexity (Fig. 3.14). The fifth task also shows how the LQR steering strategy clearly outperforms the randomized one when n_u is large.

In all tasks the robots are subject to gravity and viscous friction in all joints, and their action bounds $u_{i,\text{max}}$ in in Eq. (3.2) are small enough so as to impede direct trajectories from $\boldsymbol{x}_s$ to $\boldsymbol{x}_g$. This complicates the problems substantially, and forces the generation of swinging motions to reach the goal. The formulations used for the kinematic and dynamic constraints are those given in Chapter 2.

The planner parameters have been set according to the guidelines in Section 3.6.3. In particular we have used $\cos(\alpha) = 0.9$, $\beta = 0.1\,\sqrt{n_x}$, $\epsilon = 0.05\,\sqrt{n_x}$, $\rho = d_{\mathcal{X}}/2$, $\sigma = 2\,\rho$, and $\delta = 0.02\,\rho$. Matrix $\boldsymbol{R}$ in Eq. (3.14), in turn, has been chosen to be diagonal with $\boldsymbol{R}_{i,i} = 1/{u_{i,\text{max}}}^2$, and we use $t_{\text{max}} = 1.5$. The planner performance, however, does not depend on these parameters exclusively. The peculiarities of each problem, like the torque limits of the actuators, the system masses, or the presence of obstacles also have a strong influence. The complete set of geometric and dynamic parameters of all examples, as well as the planner implementation, are provided in http://www.iri.upc.edu/cuik.

Table 3.1 summarizes the problem dimensions and performance statistics for the different examples. The table provides the average performance over twenty runs using a MacBook Pro with an Intel i9 octa-core processor running at 2.93 GHz. The column “Success” gives the percentage of planner runs that were able to solve each problem in at most one hour. Statistics for both the randomized steering strategy and the LQR steering strategy of are given for com-

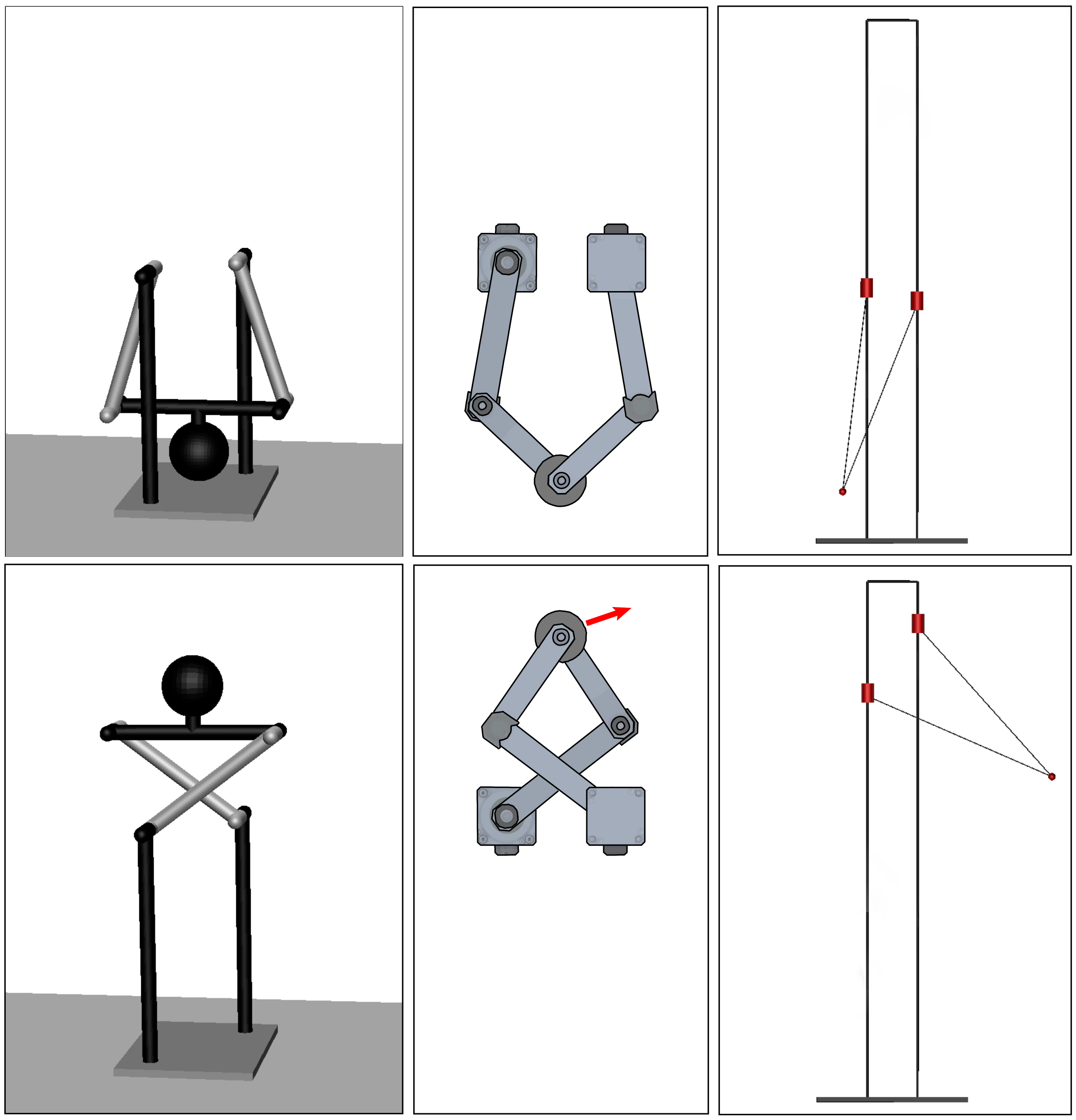


Figure 3.13: Example tasks solved with the planner. From left to right and columnwise: weight lifting with a four-bar robot, weight throwing with a five-bar robot, and a pick-and-place task with a cable-driven robot. The top and bottom rows show the start and goal states in each task. The robot is assumed to be at rest in such states, except in the goal state of the second task, in which the load has to be thrown at the shown velocity (in red).

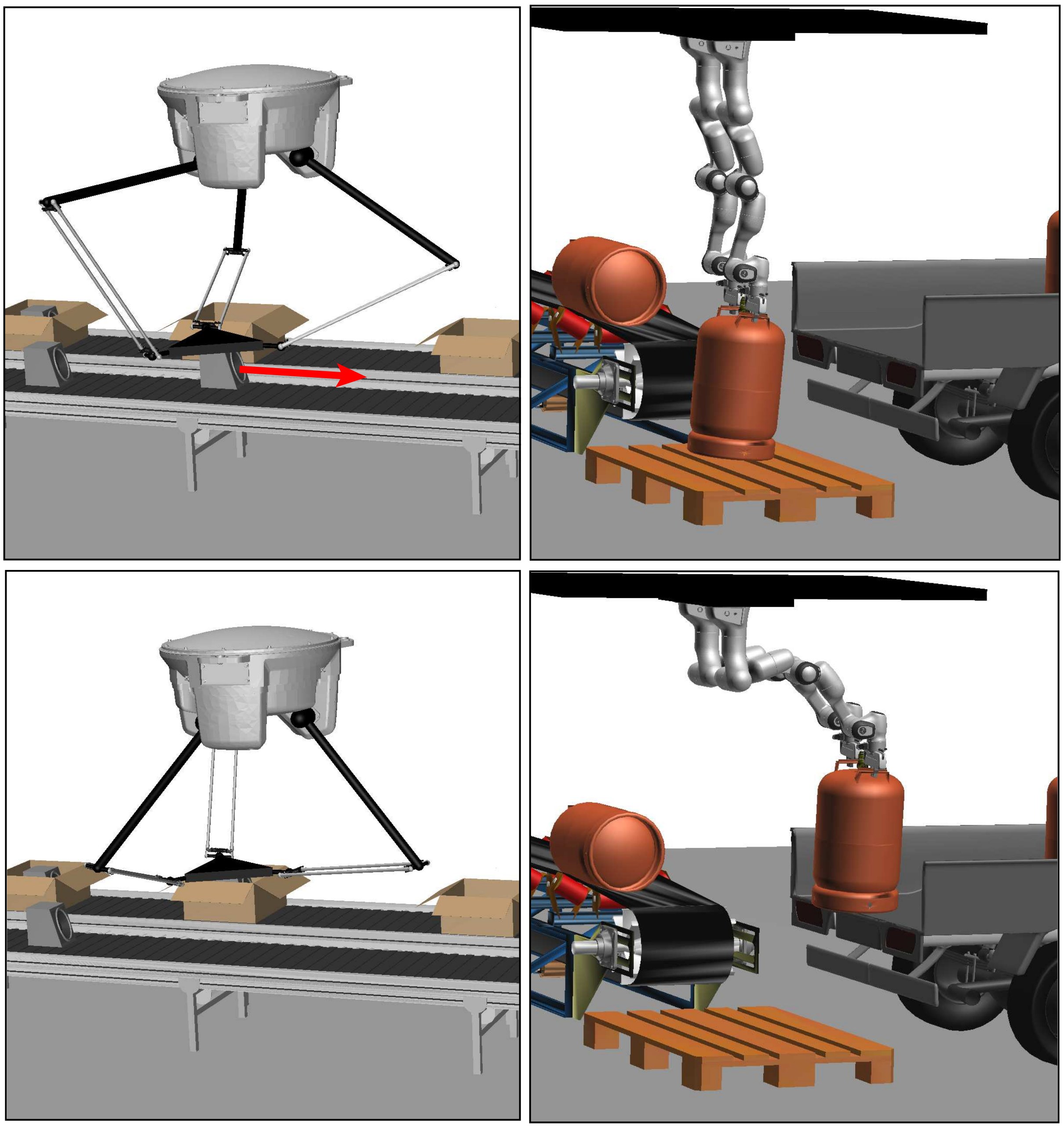


Figure 3.14: Example tasks solved with the planner on spatial robots. Left: conveyor switching with a Delta robot. Right: truck loading with two cooperative arms. The top and bottom rows show the start and goal states in each task. The robot is assumed to be at rest in such states, except in the start state of the left task, in which the Delta robot has to pick a loudspeaker that is moving to the right at the shown velocity (in red).

Example Task	Section	n_q	n_e	$d_\mathcal{X}$	n_u	Steering method	N_s	N_c	Time [s]	Success
Weight lifting with a four-bar robot	3.9.1	4	3	2	1	Random	582	111	0.85	100%
						LQR	180	63	0.61	100%
Weight throwing with a five-bar robot (crossing singularities)	3.9.2	5	3	4	2	Random	13119	9933	130.15	100%
						LQR	10113	3763	49.23	100%
Weight throwing with a five-bar robot (avoiding singularities)	3.9.2	5	3	4	2	Random	28555	19618	475.60	100%
						LQR	9104	3076	72.13	100%
Pick-and-place with a cable-driven robot	3.9.3	4	2	4	2	Random	102163	5129	108.60	100%
						LQR	21331	1048	158.32	100%
Conveyor switching with a Delta robot	3.9.4	15	12	6	3	Random	15162	912	308.30	100%
						LQR	3910	297	50.95	100%
Truck loading with cooperative arms	3.9.5	10	6	8	10	Random	9455	1813	1967.27	40%
						LQR	10882	1296	125.47	100%

Table 3.1: Problem dimensions and performance statistics for the different tasks to be planned. For each task we provide the number of generalized coordinates in $\boldsymbol{q}$ (n_q), the number of loop-closure constraints (n_e), the dimension of the state space ($d_\mathcal{X}$), and the dimension of the action space (n_u). For the two steering methods we also provide the average over twenty runs of the number of samples (N_s) and charts (N_c), the planning time, and the success rate.

parison. The randomized strategy employs $2\,n_u$ random actions from $\mathcal{U}$ to obtain $\mathcal{U}_s$, which are applied during $\Delta t = 0.1$ seconds. As seen in the table, the LQR strategy is more efficient than the randomized strategy in almost all cases, as it requires a smaller number of samples and charts, and less time, to find a solution. In particular, the success rate of the randomized strategy is only 40% in the example of Section 3.9.5. Instead, the LQR steering is always successful. Only in the example of Section 3.9.3 the randomized steering strategy outperforms the LQR steering one, since the formulation of the latter does not account for the unilateral constraints of the problem and, hence, many tree extensions yield infeasible solutions. Further details on the examples are next provided.

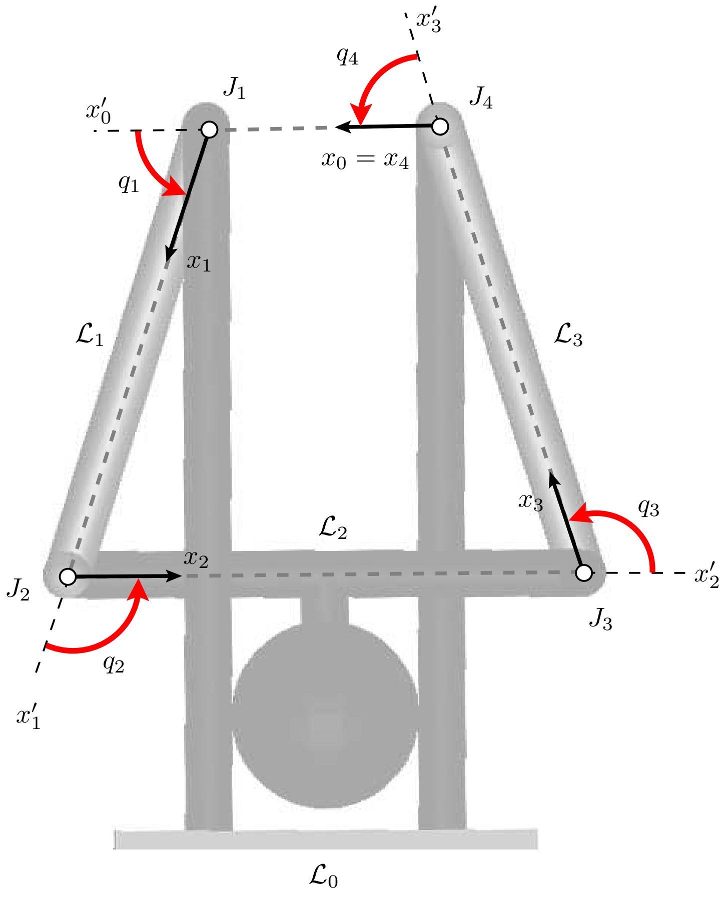


Figure 3.15: Geometry of the four-bar mechanism. For each coordinate system, only the x axis is depicted.

3.9.1 Weight Lifting with a Four-bar Robot

In this example, we have to plan the trajectory of a weight lifting task to be done with a four-bar robot (Fig. 3.13, left). The robot involves four links cyclically connected with revolute joints (Fig. 3.15). Following Section 2.4, we label the links as $\mathcal{L}_0, \ldots, \mathcal{L}_3$, and the joints as $J_1, \ldots, J_4$. Only joint J_1 is actuated. The relative angle with the following link is denoted by q_i, and the robot configuration is given by $\boldsymbol{q} = (q_1, q_2, q_3, q_4)$.

To formulate Eq. (2.1), we attach two coordinate systems to each link $\mathcal{L}_i$ centered at joints J_i and J_{i+1} respectively. The former system is called the link i coordinates and has the x_i axis aligned with the link. The overall transformation through the kinematic loop is

$$ {}^{0}\boldsymbol{X}_4(\boldsymbol{q}) = \prod_{i=1}^{4} \underbrace{\boldsymbol{X}_{\mathcal{L}_{i-1}} \cdot \boldsymbol{X}_z(q_i)}_{{}^{i-1}\boldsymbol{X}_i}, $$

where

$$\boldsymbol{X}_{\mathcal{L}_{i-1}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -L_{i-1} & 0 & 1 & 0 \\ 0 & L_{i-1} & 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\boldsymbol{X}_z(q_i) = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 & 0 & 0 & 0 \\ \sin(q_i) & \cos(q_i) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(q_i) & -\sin(q_i) & 0 \\ 0 & 0 & 0 & \sin(q_i) & \cos(q_i) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

are the transforms in Eq. (2.17), L_i is the distance between the two revolute joints of link $\mathcal{L}_i$, and

$${}^0\boldsymbol{X}_4(\boldsymbol{q}) = \begin{bmatrix} {}^0\boldsymbol{R}_4 & \boldsymbol{0} \\ \boldsymbol{S}({}^0\vec{\boldsymbol{p}}_4)\, {}^0\boldsymbol{R}_4 & {}^0\boldsymbol{R}_4 \end{bmatrix}.$$

To enforce the closure of the loop, we simply set ${}^0\boldsymbol{R}_4 = \boldsymbol{I}_3$ and ${}^0\vec{\boldsymbol{p}}_4 = \boldsymbol{0}$. Since the mechanism is planar, only 3 scalar equations are independent. Thus, to form Eq. (2.1), it suffices to select the scalar equations that correspond to the components $(2,1)$ of ${}^0\boldsymbol{R}_4$ and the first and second components of ${}^0\vec{\boldsymbol{p}}_4$. Then, Eq. (2.1) results in

$$\begin{bmatrix} L_0 + L_1\,\mathsf{c}(\bar{q}_1) + L_2\,\mathsf{c}(\bar{q}_2) + L_3\,\mathsf{c}(\bar{q}_3) \\ L_1\,\mathsf{s}(\bar{q}_1) + L_2\,\mathsf{s}(\bar{q}_2) + L_3\,\mathsf{s}(\bar{q}_3) \\ s(\bar{q}_4) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad (3.32)$$

where $\mathsf{s}(\cdot)$ and $\mathsf{c}(\cdot)$ denote the sine and cosine of their argument, respectively, and $\bar{q}_i = \sum_{j=1}^{i} q_j$ gives the angle of link $\mathcal{L}_i$ relative to ground.

Equation (2.2) could be obtained by taking the time derivative of Eq. (3.32), giving

$$\boldsymbol{\Phi}_{\boldsymbol{q}}\, \dot{\boldsymbol{q}} = \boldsymbol{0},$$

where

$$\boldsymbol{\Phi_q} = \begin{bmatrix} -L_1\,\mathrm{s}(\bar{q}_1) - L_2\,\mathrm{s}(\bar{q}_2) - L_3\,\mathrm{s}(\bar{q}_3) & -L_2\,\mathrm{s}(\bar{q}_2) - L_3\,\mathrm{s}(\bar{q}_3) & -L_3\,\mathrm{s}(\bar{q}_3) & 0 \\ L_1\,\mathrm{c}(\bar{q}_1) + L_2\,\mathrm{c}(\bar{q}_2) + L_3\,\mathrm{c}(\bar{q}_3) & L_2\,\mathrm{c}(\bar{q}_2) + L_3\,\mathrm{c}(\bar{q}_3) & L_3\,\mathrm{c}(\bar{q}_3) & 0 \\ \mathrm{c}(\bar{q}_4) & \mathrm{c}(\bar{q}_4) & \mathrm{c}(\bar{q}_4) & \mathrm{c}(\bar{q}_4) \end{bmatrix}. \tag{3.33}$$

Alternatively, recall from Section 2.4 that Eq. (3.33) reduces to $\boldsymbol{J}\,\dot{\boldsymbol{q}} = \boldsymbol{0}$ when $\boldsymbol{\Phi}(\boldsymbol{q}) = \boldsymbol{0}$, where $\boldsymbol{J}$ is the screw Jacobian of the four-bar loop. This formulation of the velocity constraint is often preferable, as $\boldsymbol{J}$ can be efficiently computed using the algorithms in Section 2.4. This Jacobian has the form

$$\boldsymbol{J} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ -a_1 & -a_2 & -a_3 & -a_4 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

where (a_i, b_i) are the (x, y) coordinates of joint J_i in link 0 coordinates [37]. Using the fact that $(a_1, b_1) = (L_0, 0)$, these coordinates can be written as

$$\begin{aligned} a_i &= a_{i-1} + L_{i-1}\,\cos(\bar{q}_{i-1}), \\ b_i &= b_{i-1} + L_{i-1}\,\sin(\bar{q}_{i-1}). \end{aligned}$$

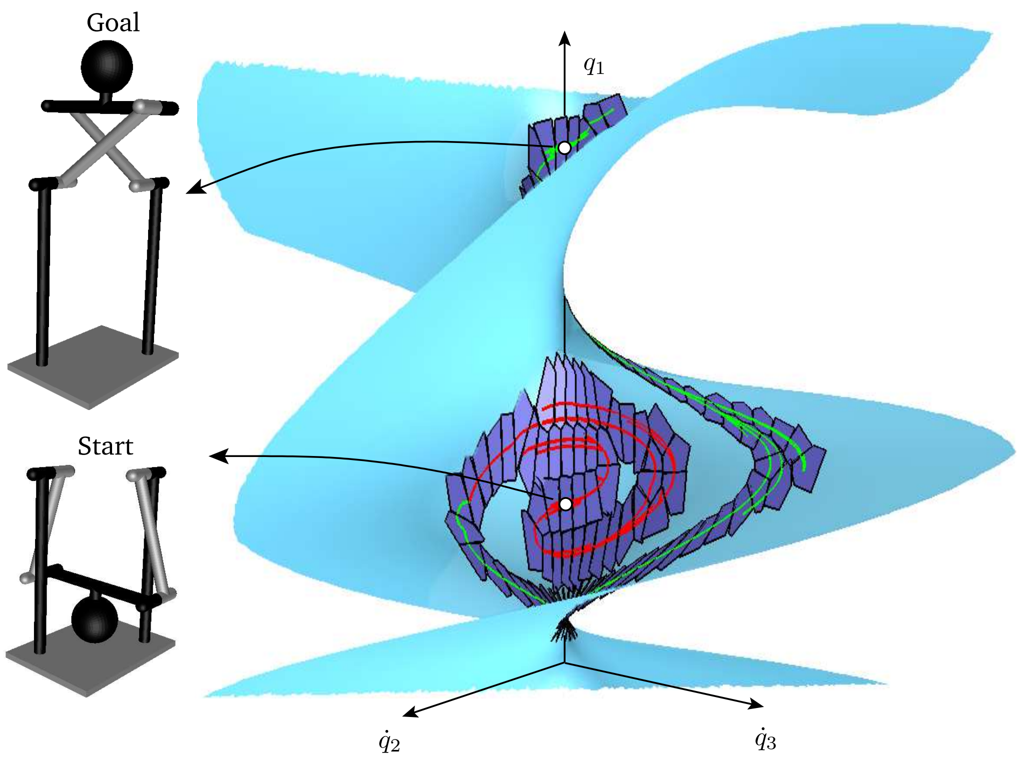
Then, the screw Jacobian becomes

$$\boldsymbol{J} = \begin{bmatrix} 0 & L_1\,\mathrm{s}(\bar{q}_1) & L_1\,\mathrm{s}(\bar{q}_1) + L_2\,\mathrm{s}(\bar{q}_2) & 0 \\ -L_0 & -L_0 - L_1\,\mathrm{c}(\bar{q}_1) & -L_0 - L_1\,\mathrm{c}(\bar{q}_1) - L_2\,\mathrm{c}(\bar{q}_2) & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

which is the same as Eq. (3.33) if Eq. (3.32) is satisfied.

Under the previous formulation we have $n_q = 4$ and $n_e = 3$, so in this case $\mathcal{X}$ is of dimension $d_{\mathcal{X}} = 2\,d_{\mathcal{C}} = 2\,(n_q - n_e) = 2$. Finally, to set up the dynamics in Eq. (3.1b), we use the methods in Section 2.4.

Figure 3.16, top, shows the shape of $\mathcal{X}$ when projected to the space defined by q_1, $\dot{q}_2$, and $\dot{q}_3$, with the start and goal states indicated. To design a trajectory connecting $\boldsymbol{x}_s$ with $\boldsymbol{x}_g$, the planner constructs the partial atlas that is shown in the figure. Since the motor torque at J_1 is limited, quasi static trajectories near the straight line from $\boldsymbol{x}_s$ to $\boldsymbol{x}_g$ are impossible, and the robot is deemed to perform pendulum-like motions to reach the goal. This translates into the spirally tree trajectories that we observe in the figure. The trajectory returned by the planner can be seen in Fig. 3.16, bottom.



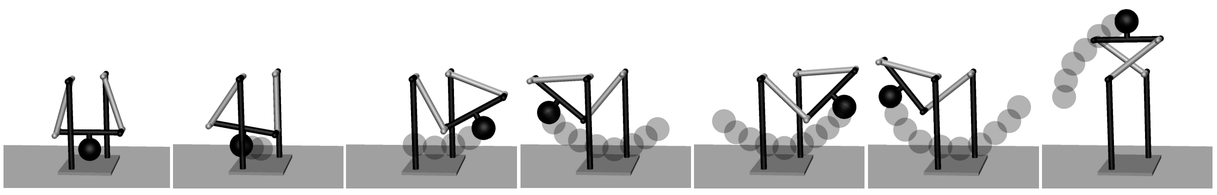


Figure 3.16: Top: A partial atlas (in dark blue) of the state space $\mathcal{X}$ (in light blue) used to plan the lifting of a weight with the four-bar robot. The red and green trees are rooted at the start and goal states respectively, and they are grown towards each other in parallel with the atlas. Each polygon in dark blue corresponds to the P_c set of a given chart. Note that the validity of each P_c set is larger in practice, but it is reduced here for illustrative purposes. Bottom: snapshots of the solution trajectory found by the planner. In each snapshot, the trail depicts earlier positions of the load. See youtu.be/6STb_FjgDJg for an animated version of this figure.

This example can also be used to illustrate the performance of the LQR steering strategy. Fig. 3.17-left, shows an example in which this strategy successfully finds a trajectory connecting $\boldsymbol{x}_{near}$ with $\boldsymbol{x}_{\text{rand}}$, with t_f^* always decreasing. In contrast, Fig. 3.17-right shows another example in which the process tends to a limit cycle like the one in Fig. 3.10(b), and is never able to reach $\boldsymbol{x}_{\text{rand}}$. The steering method in Algorithm 3.4 would stop after a few iterations because a point is reached in which t_f^* no longer decreases.

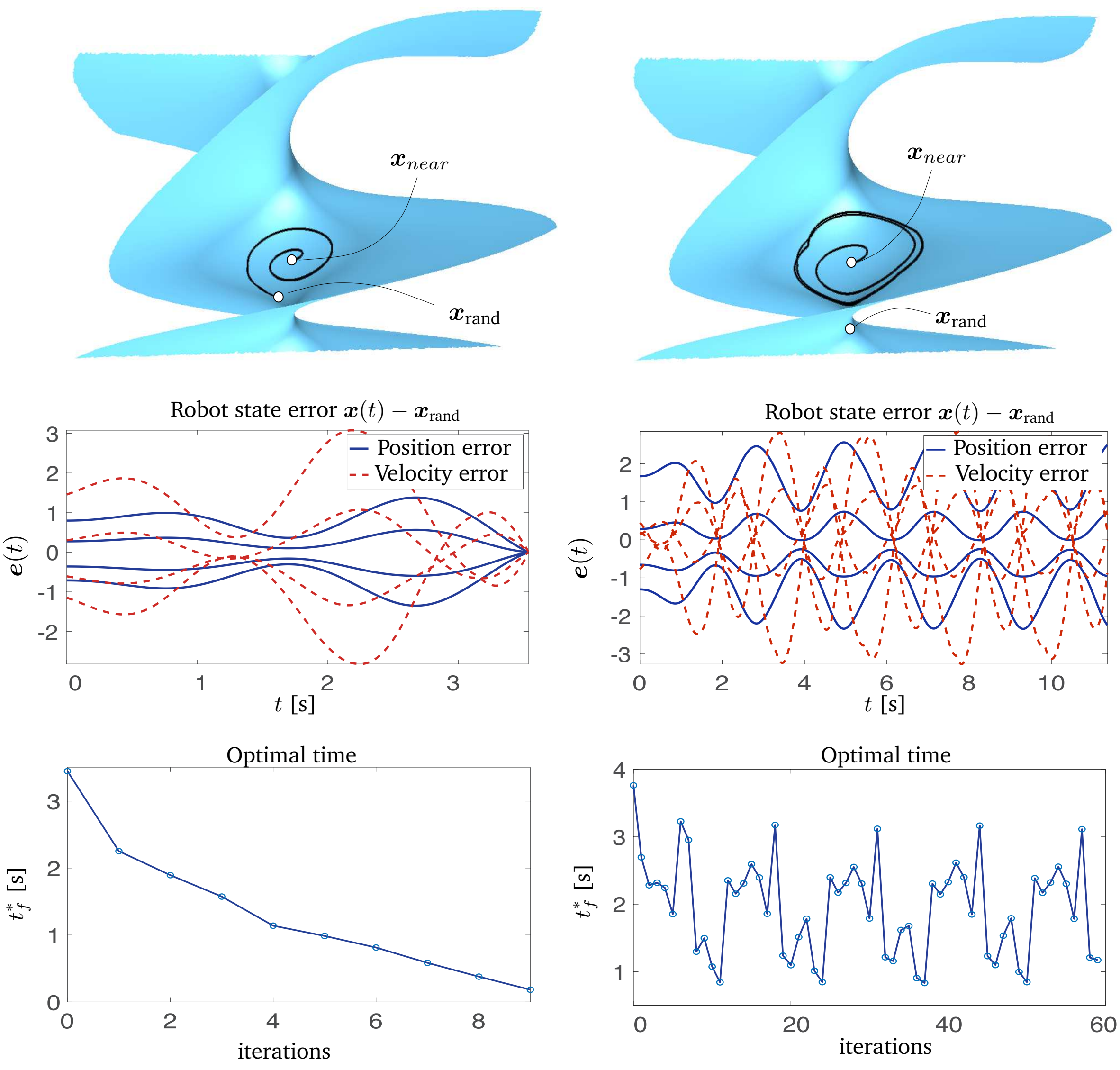
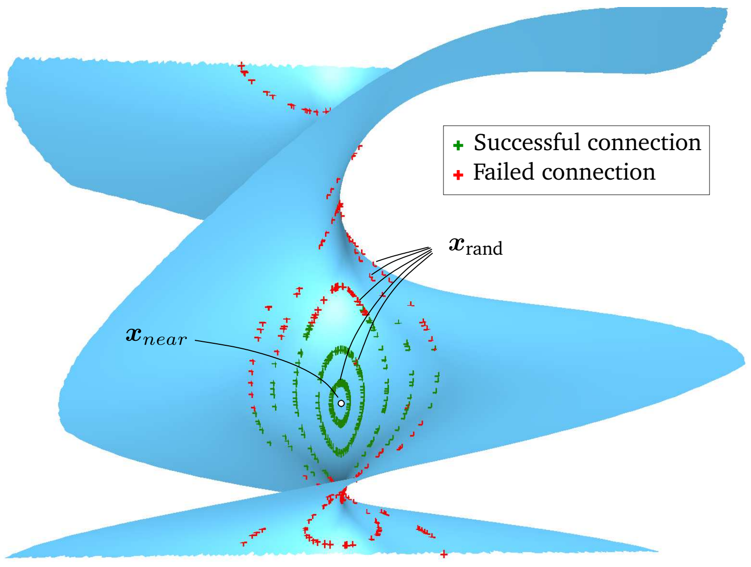


Figure 3.17: Steering the four-bar robot from x_{near} to x_{rand}. Left: The LQR strategy allows the planner to connect x_{near} and x_{rand}. Right: The strategy enters a limit cycle and is never able to reach x_{rand}. The right plot shows that t_f^* no longer decreases after six iterations, so it would be aborted at this point.

In Fig. 3.18 we also show the performance of the LQR strategy for states x_{rand} that are progressively further away from x_{near}. We have generated 5 batches of 100 random samples, where the samples in each batch are at tangent space distances of 0.4, 1, 2, 3, and 4 from x_{near}. As a reference, the distance from x_s to x_g is 3.7 in this example. The states x_{rand} that could be connected to x_{near} are shown in green, while those that could not are shown in red. As expected for a local planner, the closer x_{rand} from x_{near}, the higher the probability of success of the steering process.



Distance	0.4	1	2	3	4
Success rate	100%	99%	56%	26%	10%

Figure 3.18: Success rate of the LQR steering strategy for states $\boldsymbol{x}_{\text{rand}}$ that are increasingly far from $\boldsymbol{x}_{near}$.

3.9.2 Weight Throwing with a Five-bar Robot

In this example, the planner has to compute a trajectory to throw a load with a five-bar robot, assuming the start and goal states indicated in Fig. 3.13 (center). Initially, the load is at rest in a bottom position, and the robot has to perform back-and-forth motions to bring it to the upright position with the required velocity.

The planner will be set to compute the trajectory in two situations. In one of them it will be allowed to cross forward singularities, so these trajectories will have to be stabilized with an LQR controller later on (Chapter 5). In the other, the planner will have to avoid the singularities, so a computed-torque controller will suffice for the purpose of trajectory tracking.

The geometry of the robot is shown in Fig 3.19. The mechanism involves one kinematic loop with five links $\mathcal{L}_0, \ldots, \mathcal{L}_4$ pairwise connected by revolute joints. A load is mounted at point Q in the figure (the axis of the third joint, which plays the role of the end-effector). In the figure, q_i denotes the relative angle at the i-th joint, which connects $\mathcal{L}_i$ with $\mathcal{L}_{i-1}$. The robot configuration is then given by $\boldsymbol{q} = (q_1, q_2, q_3, q_4, q_5)$. Joints 1 and 5 are actuated, while the rest of joints are passive. Note from Fig 3.19 (right) that the links and the load move in different planes, so collisions need not be taken into account in this robot.

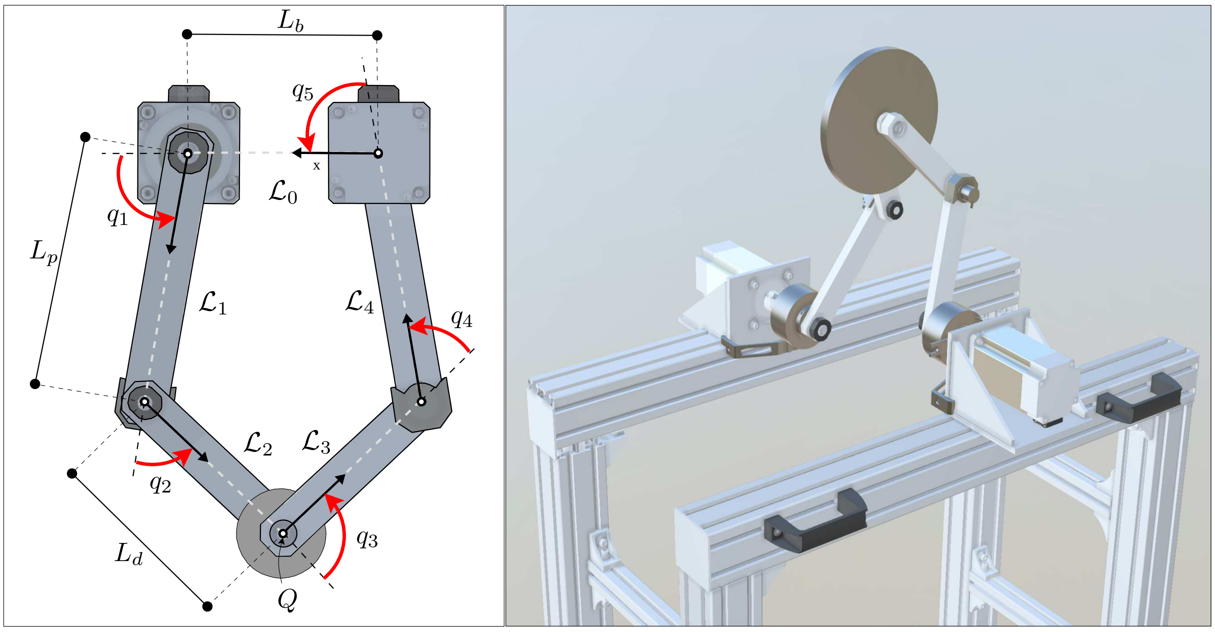


Figure 3.19: Geometry of a five-bar parallel robot. All joints are of revolute type. The joints of q_1 and q_5 are actuated, and the remaining joints are passive. The motors are fixed to the ground, which acts as a fifth bar.

The formulation of the kinematic constraints is analogous to the one in the previous example, with the difference that the robot now has one additional link and joint. As a result, $n_q = 5$, $n_e = 3$, and $\mathcal{X}$ is four-dimensional in this case. The loop closure constraint in Eq. (2.1) is thus given by

$$\begin{bmatrix} L_p\ \mathrm{c}(q_1) + L_d\ \mathrm{c}(q_1+q_2) + L_d\ \mathrm{c}(q_1+q_2+q_3) + L_p\ \mathrm{c}(q_1+q_2+q_3+q_4) + L_b \\ L_p\ \mathrm{s}(q_1) + L_d\ \mathrm{s}(q_1+q_2) + L_d\ \mathrm{s}(q_1+q_2+q_3) + L_p\ \mathrm{s}(q_1+q_2+q_3+q_4) \\ s(q_1+q_2+q_3+q_4+q_5) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad (3.36)$$

where L_b is the base distance, L_p is the length of the proximal links $\mathcal{L}_1$ and $\mathcal{L}_4$, and L_d is the length of the distal links $\mathcal{L}_2$ and $\mathcal{L}_3$. By taking into account that the actuated and remaining coordinates are $\boldsymbol{q}_u = [q_1, q_5]^\top$ and $\boldsymbol{q}_r = [q_2, q_3, q_4]^\top$ respectively, we can determine the forward singularity condition by taking the partial derivative of Eq. (3.36) with respect to $\boldsymbol{q}_r$ to obtain

$$\boldsymbol{\Phi}_{\boldsymbol{q}_r} = \begin{bmatrix} -L_d\ \mathrm{s}(\bar{q}_2) - L_d\ \mathrm{s}(\bar{q}_3) - L_p\ \mathrm{s}(\bar{q}_4) & -L_d\ \mathrm{s}(\bar{q}_3) - L_p\ \mathrm{s}(\bar{q}_4) & -L_p\ \mathrm{s}(\bar{q}_4) \\ L_d\ \mathrm{c}(\bar{q}_2) + L_d\ \mathrm{c}(\bar{q}_3) + L_p\ \mathrm{c}(\bar{q}_4) & L_d\ \mathrm{c}(\bar{q}_3) + L_p\ \mathrm{c}(\bar{q}_4) & L_p\ \mathrm{c}(\bar{q}_4) \\ \mathrm{c}(\bar{q}_5) & \mathrm{c}(\bar{q}_5) & \mathrm{c}(\bar{q}_5) \end{bmatrix}.$$

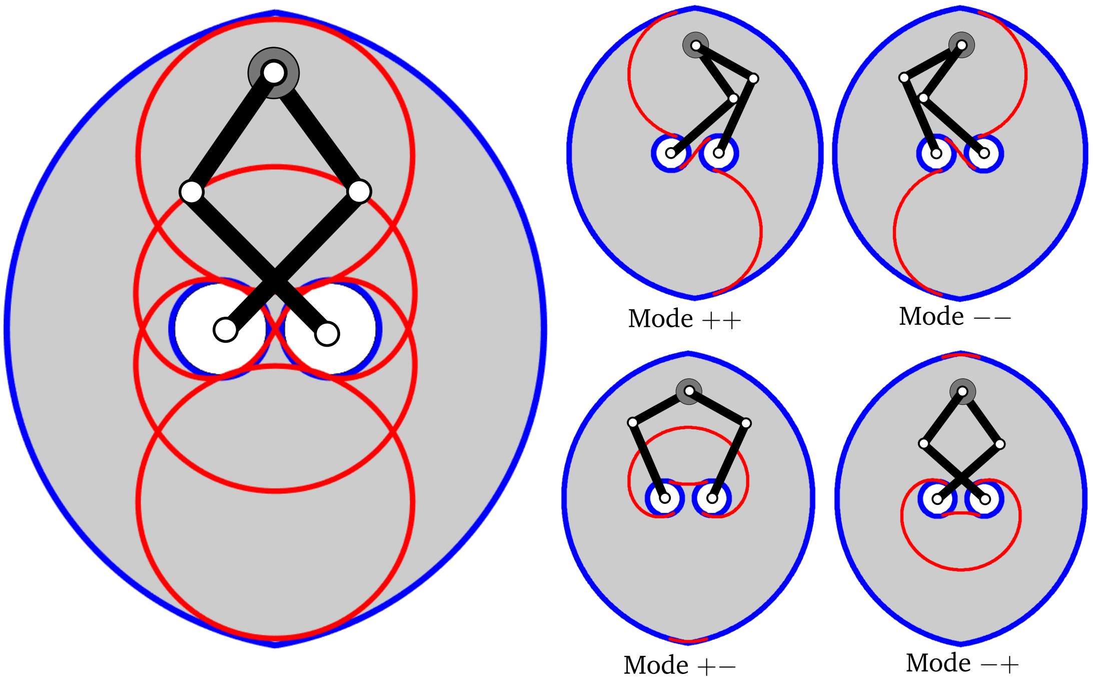


Figure 3.20: Left: The workspace (in gray) and forward singularities (in red) of the five-bar robot. Right: the same singularities, classified according to the different working modes. For each mode, we depict a robot configuration that corresponds to the solution of the inverse kinematics for the shown position of Q under the assumed mode.

When $\mathbf{\Phi}_{\boldsymbol{q}_r}$ is rank deficient, i.e. when $\det(\mathbf{\Phi}_{\boldsymbol{q}_r}) = 0$, the robot is at a forward singularity. As described in [77], this condition is fulfilled when q_3 is either 0 or π, i.e., when the distal links get aligned. The positions of Q for which such an event occurs are depicted in red in Fig. 3.20, left. From this figure it appears that the workspace of Q (in gray) is severely limited by the presence of forward singularities, but note that each position of Q can be attained by up to four inverse kinematic solutions of the robot. Each solution corresponds to one working mode identified by the signs of q_4 and q_2 in the range $[-\pi, \pi]$. If we separate the singularities according to these signs, larger singularity-free regions arise (Fig. 3.20, right).

Figure 3.21 shows two trajectories obtained by the planner. In both cases, the robot oscillates to gain momentum before throwing the load. While the upper trajectory is obtained by planning in the state space $\mathcal{X}$, the lower one is obtained by planning in the singularity-free state space $\mathcal{X}_{\text{sfree}}$, which uses the formulation given in Section 3.8. Note how the upper trajectory easily crosses forward singularities while also changing the working mode. This occurs in the second, third and fourth snapshots. In contrast, in the lower trajectory there are also many working mode changes during the move but the robot never crosses the forward singularity lo-

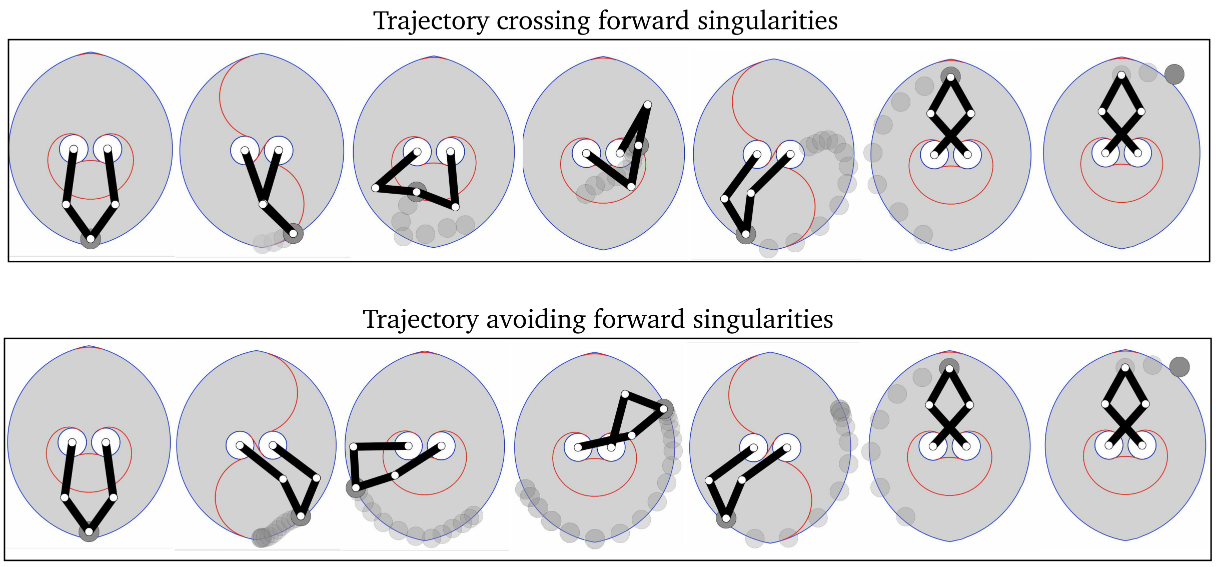

Trajectory crossing forward singularities
Trajectory avoiding forward singularities

Figure 3.21: Snapshots of a solution trajectory for the five-bar robot crossing (top) and avoiding (bottom) forward singularities. The second, third and fourth top snapshots depict singularity crossings. See youtu.be/QsiMc6bm21s for an animated version of this figure.

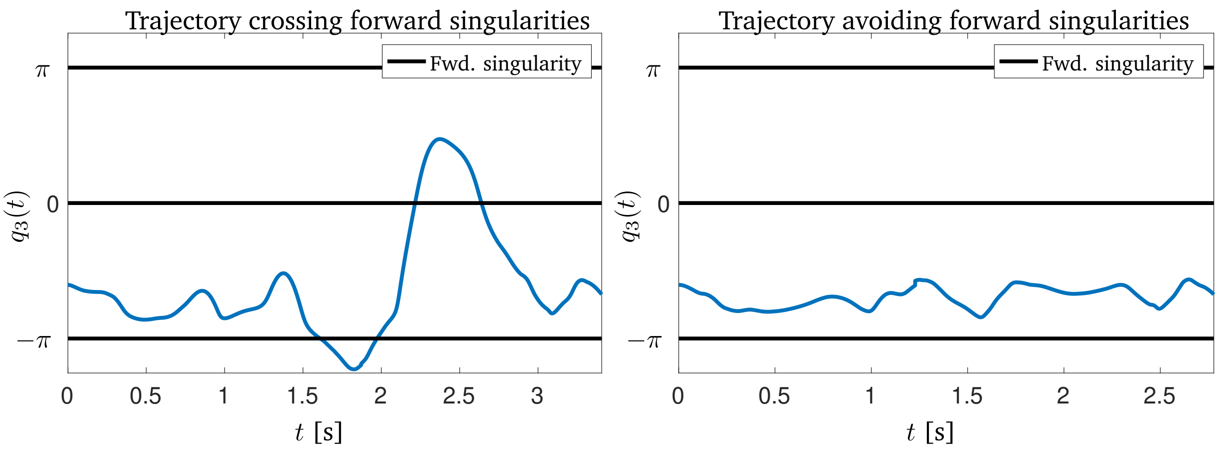


Figure 3.22: Trajectory of $q_3(t)$ for the five-bar robot crossing (left) and avoiding (right) forward singularities. Note that there is a singularity crossing when q_3 is 0 or $\pm\pi$.

cus. The crossing and avoidance of forward singularities can also be seen in the trajectory plot of Fig. 3.22, which shows the trajectories of $q_3(t)$ corresponding to Fig. 3.21. Note there are several singularity crossings when q_3 is either 0 or $\pm\pi$ in Fig. 3.22, left, while this never occurs in the singularity-free trajectory in Fig. 3.22, right.

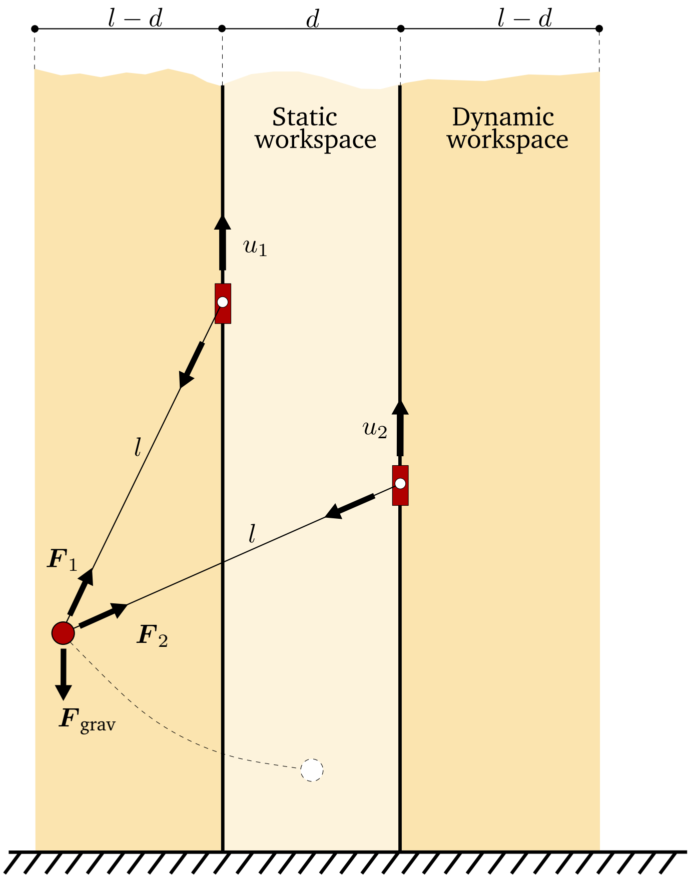


Figure 3.23: A 2-DOF cable-driven robot. This robot is subject to the gravitational force $\boldsymbol{F}_{\text{grav}}$ and the actuation forces u_1 and u_2. During operation, the cable tensions $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ must remain positive and below admissible upper bounds. Traditionally, these robots are moved in their static workspace (of width d in the figure), but our planner allows to extend their motions to the dynamic workspace (of width $2l-d$).

3.9.3 Pick-and-place Operations with a Cable-driven Robot

Consider the robot in Fig. 3.23 now, which consists of a moving load suspended from two cables, each connected to a vertical slider actuated by a DC motor. While in our previous examples we did not limit the constraint forces, in this robot we have to keep the cable tensions $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ positive to maintain the load under control. Also, the tensions must be below certain limits to avoid breaking the cables. Thus, this example serves to illustrate the ability of the planner to keep constraint forces within bounds. For each state visited by the RRT, these forces are computed using the methods in Section 2.4.5, and the state is judged as unfeasible if they surpass the allowed limits. If that happens, the expansion of the RRT has to continue in other directions.

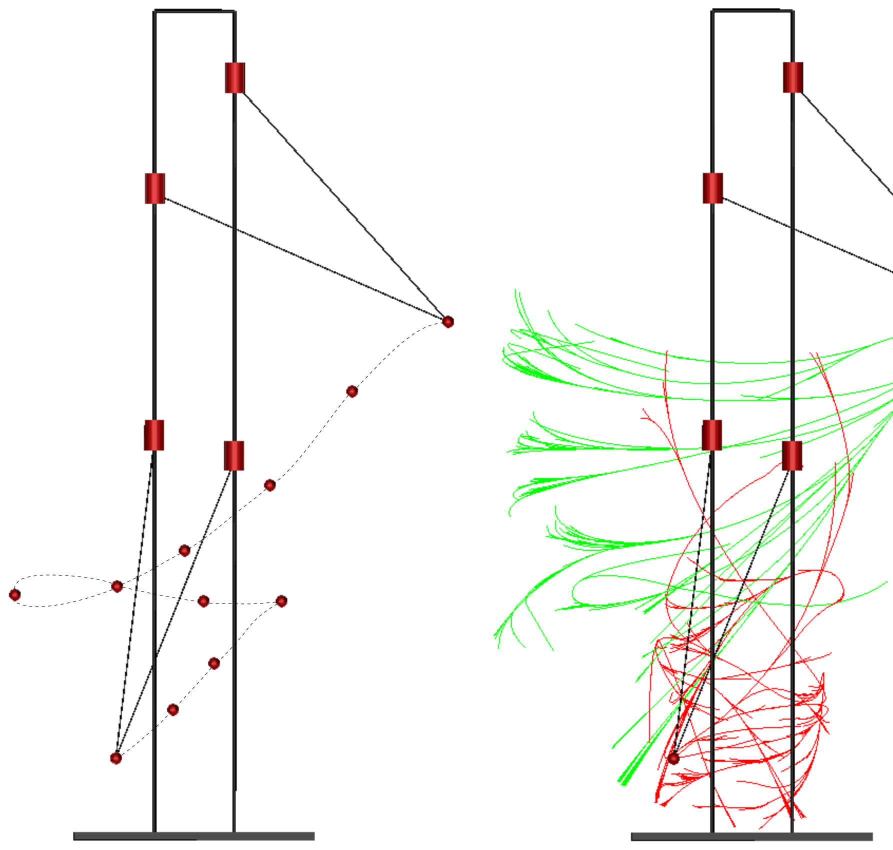


Figure 3.24: Left: A dynamic trajectory computed by the planner. Right: The RRT generated to obtain the trajectory. See youtu.be/DlxlQ3TSQ-I for an animated version of this figure.

The example also shows the potential of the planner in cable-driven robotics. Traditionally, cable-suspended robots like this one are used as robotic cranes, operating them in quasi-static conditions in which gravity is the sole source of tension [78, 79]. While this simplifies the planning and control of the motions, it also confines them to the static workspace (the region between bars in Fig. 3.23). More recently, however, inertia has also been proposed as another source of tension [80], which allows more complex motions in the dynamic workspace [81]. This region is usually much larger, as it is the set of points that can be attained when load accelerations are possible (the total area in Fig. 3.23). Using inertia and pendulum-like motions, tasks outside the footprint of the robot can then be planned [82, 83]. While such trajectories are often designed in an ad-hoc manner, our planner can generate them in a large variety of mechanisms.

The particular task to be planned consists in picking a load that is at rest in a bottom position of the workspace, to later deliver it in a higher position also at rest. The two positions, shown in Fig. 3.13 (right), are clearly outside the static workspace. We also assume that the cables and the sliders move on different planes, so their collisions need not be checked during the planning.

To model this particular robot, we view each bar-cable connection as a prismatic and a revolute joint in series. The robot configuration is then defined by the two slider displacements and the angles of their revolute joints, which results in $n_q = 4$ coordinates. Then, the formulation of Eq. (2.1) simply consists in forcing the coincidence of the cables at the endpoints of the load. This imposes two loop-closure constraints, so $n_e = 2$ in this case. Therefore, $d_{\mathcal{X}} = 2(n_q - n_e) = 4$, and, since the two sliders are actuated, $n_u = 2$.

The planner obtains the trajectory shown in Fig. 3.24 (left). From this figure and its video we see that the load is forced to oscillate in order to reach the goal. The planner has created the RRT shown in Fig. 3.24 (right) to solve the problem, consisting of two trees rooted at the start and goal states (in red and green respectively). While the robot has a limited footprint, including dynamics has increased its usable workspace substantially.

3.9.4 Conveyor Switching with a Delta robot

So far, all robots have been single-loop mechanisms operating in obstacle-free environments. To exemplify the planner in a multi-loop mechanism surrounded by obstacles, we next consider a Delta robot that has to perform a conveyor (Fig. 3.14, left). The system is formed by a fixed base connected to a moving platform by means of three legs. Each leg is an R-R-Pa-R chain, where R and Pa refer to a revolute and a parallelogram joint respectively (Fig. 3.25, left).

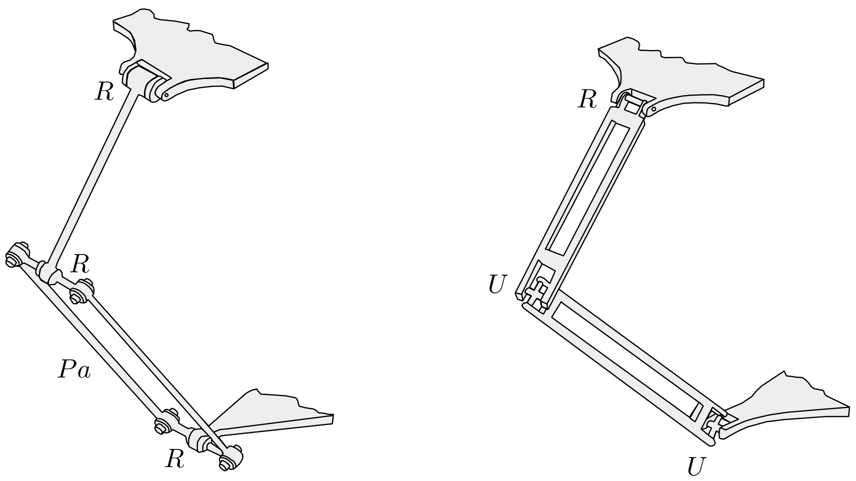


Figure 3.25: A leg of a Delta robot (left) and its equivalent R-U-U chain (right).

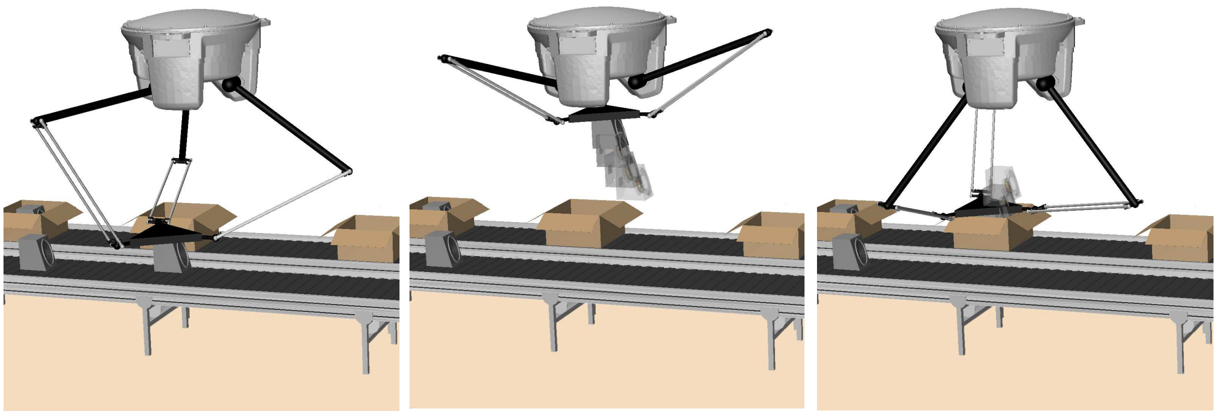


Figure 3.26: Snapshots of a solution trajectory for the conveyor switching task. An animated version of this figure can be seen in youtu.be/gU0WBW2MlHI.

The Pa joint is a planar four-bar mechanism whose opposite sides are of equal length. While it seems that such a leg should be modeled with seven joint angles, we use the fact that the leg is kinematically equivalent to an R-U-U chain (Fig. 3.25, right), where U refers to a universal joint. Since a U joint is equivalent to two R joints with orthogonal axes, we conclude that only five angles are needed to define a leg configuration. Our $\boldsymbol{q}$ vector for the Delta robot will thus involve $n_q = 3 \cdot 5 = 15$ angles in total. Only the revolute joints at the fixed base of the robot are actuated, meaning that $n_u = 3$ in this case.

To formulate the kinematic constraints, note that every spatial kinematic loop of the robot defines a six-dimensional loop-closure constraint (Chapter 2), which gives a total of three loops. However, only two of these loops are actually independent, so $n_e = 2 \cdot 6 = 12$ in this system. This means that $d_{\mathcal{X}} = 2\,(n_q - n_e) = 2\,(15 - 12) = 6$. As in all Delta robots, the robot dimensions are such that the moving platform can only translate in its workspace.

The task to be planned consists in picking a loudspeaker from a conveyor belt moving at a certain speed, to later place it inside a static box on a second belt. Obstacles play a major role in this example, as the planner has to avoid the collisions of the robot with the conveyor belts, the boxes, and the supporting structure, while respecting the joint limits. In fact, around 70% of branch extensions are stopped due to collisions in this example. The resulting trajectory can be seen in Fig. 3.26. Given the velocity of the moving belt, the planner is forced to reduce the initial kinetic energy of the load before it can place it inside the destination box. The trajectory follows an ascending path that converts the initial momentum into potential energy, to later move the load back to the box at the goal location.

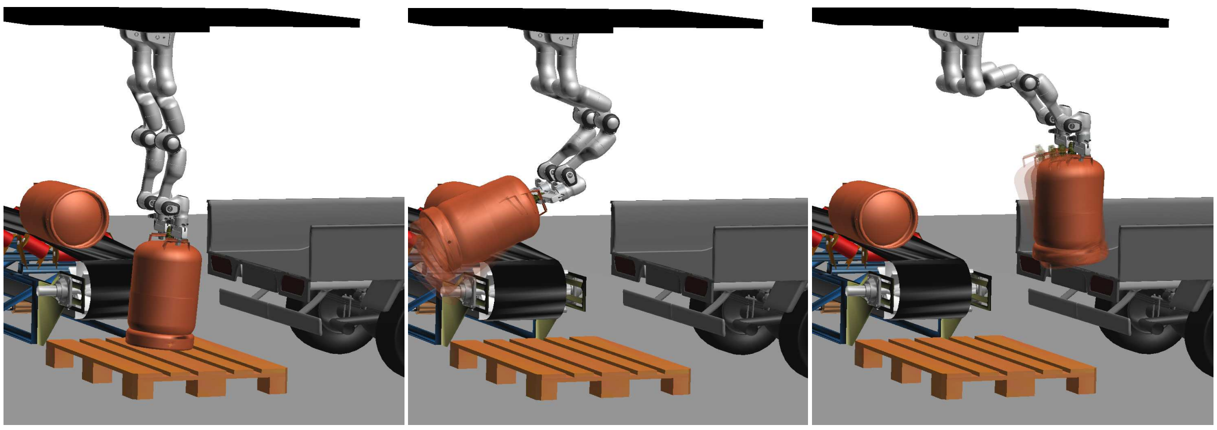


Figure 3.27: Snapshots of a solution trajectory for the truck loading task. See youtu.be/1Jm3HBM0koI for an animated version of this figure.

3.9.5 Truck Loading with Cooperative Arms

The last task to be planned involves two 7-DOF Franka Emika arms moving a gas bottle cooperatively (Fig. 3.14, right). The task consists in lifting the bottle onto a truck while avoiding the collisions with the surrounding obstacles (a conveyor belt, the ground, and the truck). The first and last joints in each arm are held fixed during the task, and the goal is to compute control functions for the remaining joints, which are all actuated. The weight of the bottle is twice the added payload of the two arms, so in this example the planner allows the system to move much beyond its static capabilities.

In this problem, we have $n_q = 10$ and $n_e = 6$, so $d_{\mathcal{C}} = n_q - n_e = 4$, and $d_{\mathcal{X}} = 2d_{\mathcal{C}} = 8$. Since all joints are actuated we have $n_u = 10$ and there are no forward singularities. Clearly, the resulting system is redundantly actuated, as $n_u > d_C$.

The example also illustrates that the randomized steering strategy performs poorly when n_u is large as in redundant manipulators. In this case, $n_u = 10$, which is notably higher than in the previous examples. Note that the number of random actions needed to properly represent $\mathcal{U}$ should be proportional to its volume, so it should grow exponentially with n_u in principle. To alleviate the curse of dimensionality, however, [41] proposes to simulate only $2\, n_u$ random actions for each branch extension. Our implementation adopts this criterion but, like [41], it then shows a poor exploration capacity when n_u is large, resulting in the excessive planning times reported for the truck loading task (Table 3.1). We have also tried to simulate 2^{n_u} random actions, instead of just $2n_u$, but then the gain in exploration capacity does not outweigh the large computational cost of simulating the actions. In contrast, the LQR strategy only computes one control per branch extension, so an increase in n_u does not affect the planning time dramatically (Table 3.1, last column). Using this strategy, the planner obtained trajectories like the one shown in Fig. 3.27, in which we see that, in order to gain momentum, the bottle is moved backwards before lifting it onto the truck.

4

Trajectory Optimization

The planner in Chapter 3 obtains trajectories $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ that respect the kinodynamic constraints, but that are not optimal in any specific sense in principle. Their motions may be too jerky, or require excessive time or control effort in comparison to those of other feasible trajectories. To improve them, we next provide optimization tools that locally modify $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ until they minimize a predetermined cost function. For example, if the goal state must be reached as rapidly as possible, we can minimize time, and bang-bang controls arise as a result. Alternatively, we can minimize the $\boldsymbol{u}(t)$ or $\dot{\boldsymbol{u}}(t)$ values to obtain smoother, easier-to-track controls. Weighted combinations of these or other cost functions can also be treated with the tools we provide. We begin with a short review of how our work fits in the context of earlier developments (Sections 4.1), to then state the problem to be solved (Section 4.2) and provide background techniques for the rest of the chapter (Section 4.3). Following a direct approach, we tackle the problem by transcribing it into one of constrained minimization, which we then solve using state-of-the-art methods of nonlinear programming. We use collocation for the transcription, but we show how standard collocation schemes generate trajectories that substantially drift away from $\mathcal{X}$ in closed-chain robots (Section 4.4). To overcome this problem we then develop two new methods that do not generate drift along the obtained trajectories (Section 4.5). We also mention a few points that must be considered when implementing the methods, and compare their performance on optimizing the weight throwing task of the previous chapter (Sections 4.6 and 4.7).

4.1 Related Work

In essence, all trajectory optimization methods solve an instance of the variational problem of optimal control. Two strategies are mainly followed [84, 85]. Indirect approaches initially derive the Pontryagin conditions of optimality and then solve the resulting boundary-value problem numerically. Direct approaches, in contrast, discretize the optimal control problem at the outset, and then tackle the discrete problem with nonlinear programming methods. While indirect approaches tend to be more accurate on approximating the optimal trajectory, they also require good initial guesses of the solution, which are difficult to provide in general. Direct approaches, in contrast, may yield slightly suboptimal trajectories, but show larger regions of convergence, which makes them preferable to solve problems in robotics. In these approaches, the dynamic constraints can be discretized using shooting methods, which use explicit integrators to estimate the evolution of the system, or collocation methods, which avoid costly integration rules via spline approximations. Collocation methods are relatively fast and effectively solve a wide variety of problems, which justifies the growing interest they arouse [86–90] and why they constitute, in particular, the main tool to be used in this chapter.

When optimizing a trajectory, a main concern is to guarantee the consistency of the motor actions with the trajectory states, so the actions can closely reproduce such states when executed in the real robot. This calls for the use of realistic robot models, but also for an accurate satisfaction of the kinematic and dynamic constraints in such models along the entire trajectory. While direct collocation methods are good at ensuring dynamic accuracy, little work has been devoted to also guarantee their kinematic accuracy on closed-chain robotic systems. In such systems, the trajectories must stay on the $\mathcal{X}$ manifold we defined in Chapter 2, otherwise they would be unrealistic and difficult to track with a controller [87, 89]. Existing strategies to avoid drifting away from $\mathcal{X}$ fall into two categories. In a first group of methods, which we call basic methods, the kinematic constraints are simply not enforced, but fine trajectory discretizations or high-order integrators can be used to keep manifold drift to a minimum [88, 90–94]. This approach is simple, but it increases the computational cost of solving the problem, and one can find situations in which excessive drift accumulates despite the precautions [95]. A second group of methods implements some sort of trajectory stabilization [87, 95–99]. The most popular technique is Baumgarte stabilization [100], which modifies the dynamic vector field of the system to make it convergent to the manifold. The method is easy to implement, and keeps the trajectory near the manifold, but it is also problematic in some ways: it adds artificial compliance and energy dissipation to the system, its stabilizing parameters may be difficult to tune, and trajectory instabilities may arise as a result [63].

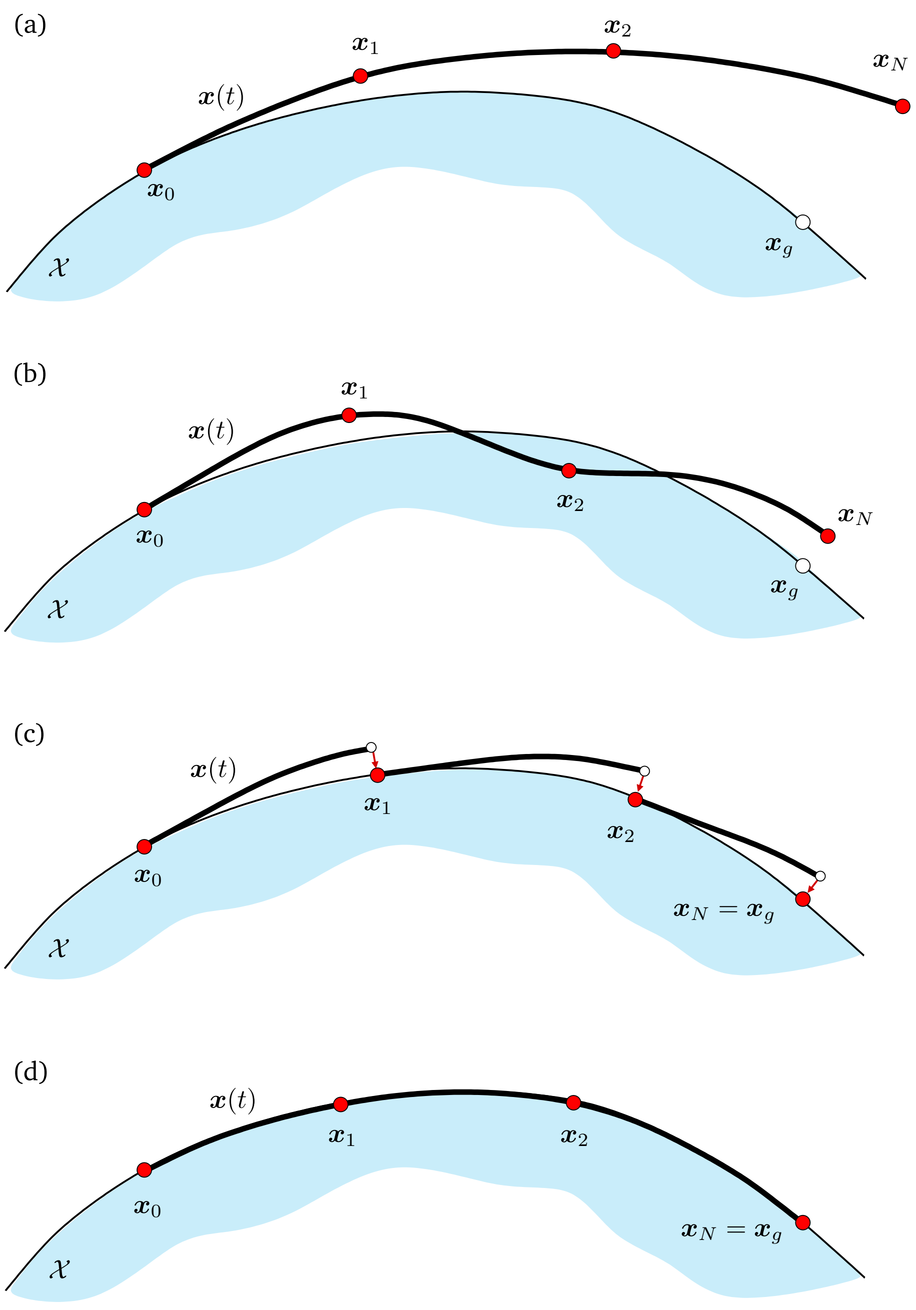


Figure 4.1: Qualitative form of the trajectories $\boldsymbol{x}(t)$ obtained by existing collocation methods (a and b) and those we propose in this chapter (c and d). From top to bottom: output of the "basic", "Baumgarte", "projection", and "local coordinates" methods. The blue surface $\mathcal{X}$ is the state space manifold on which $\boldsymbol{x}(t)$ should evolve. The start and goal states are $\boldsymbol{x}_0$ and $\boldsymbol{x}_g$. The red dots indicate the knot states $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N$ used to discretize $\boldsymbol{x}(t)$. In the basic method, the knot states, and so $\boldsymbol{x}(t)$, may increasingly drift away from $\mathcal{X}$. Both the knot states and $\boldsymbol{x}(t)$ are kept near $\mathcal{X}$ in the Baumgarte method, but at the cost of modifying the system dynamics. In contrast, the methods we propose fully eliminate drift at the knot states (c) or along the entire trajectory (d) without modifying the system dynamics.

Our goal in this chapter is to review the foregoing methods and to propose new ones to overcome their limitations. By using projections and local charts of the manifold, we present two collocation methods that keep the discrete trajectory exactly on the manifold without adding artificial terms to the system dynamics. The two methods are referred to as the “projection” and “local coordinates” methods (Fig. 4.1). While the former is easier to implement, the latter achieves full drift elimination even for the continuous-time trajectory. Both methods leverage results and techniques from geometric integration on manifolds [25, 26].

4.2 The Trajectory Optimization Problem

Recall from Section 2.3.2 that the time evolution of closed-chain robot is determined by the system of differential-algebraic equations

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, & (4.1a) \\ \dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), & (4.1b) \end{cases}$$

where Eq. (4.1a) defines the state space $\mathcal{X}$ of the robot, and Eq. (4.1b) defines the robot dynamics on $\mathcal{X}$. Besides being constrained by Eqs. (4.1a) and (4.1b), $\boldsymbol{x}$ and $\boldsymbol{u}$ are also subject to a system of path constraints with the form

$$\boldsymbol{0} \leq \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}), \tag{4.2}$$

where $\boldsymbol{h}$ is a differentiable function used to encode, in a single constraint, the joint, force, or velocity limits, the obstacle-avoidance constraints, or any other constraints imposing state or action bounds. Thus, Eq. (4.2) can be used to define the sets $\mathcal{X}_{\text{feas}}$ and $\mathcal{U}$ we considered in Chapter 3.

With these definitions, our trajectory optimization problem can be formally posed as follows. Given a running cost function $L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$, and two states of $\mathcal{X}$, $\boldsymbol{x}_s$ and $\boldsymbol{x}_g$, find the state and action trajectories $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$, and a time t_f, that

$$\begin{aligned}
\underset{\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_f}{\text{minimize}} \quad & \int_0^{t_f} L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)\, \mathrm{d}t & (4.3a) \\
\text{subject to} \quad & \boldsymbol{x}(0) = \boldsymbol{x}_s, & (4.3b) \\
& \boldsymbol{x}(t_f) = \boldsymbol{x}_g, & (4.3c) \\
& t_f \geq 0, & (4.3d) \\
& \boldsymbol{0} \leq \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \qquad t \in [0, t_f], & (4.3e) \\
& \dot{\boldsymbol{x}}(t) = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [0, t_f]. & (4.3f)
\end{aligned}$$

It is important to realize that Eq. (4.1a) is not added to this continuous-time formulation, as it is already accounted for implicitly by Eq. (4.3f) when $\boldsymbol{x}_s \in \mathcal{X}$. Note also that, while the cost function in (4.3a) is of Lagrange form, cost functions in the more general Bolza form could also be considered, as they can always be re-expressed as in Eq. (4.3a) [101].

4.3 Transcription Techniques

4.3.1 Problem Discretization

To solve Problem (4.3) we will transcribe it into one of constrained minimization. This entails approximating all functionals in Eqs. (4.3a)-(4.3f) by functions of discrete states and actions. To achieve so, we discretize the time horizon $[0, t_f]$ into $N+1$ time instants

$$t_0, \dots, t_k, \dots, t_N,$$

where $t_0 = 0$ and $t_N = t_f$, and represent $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ by the $N+1$ states

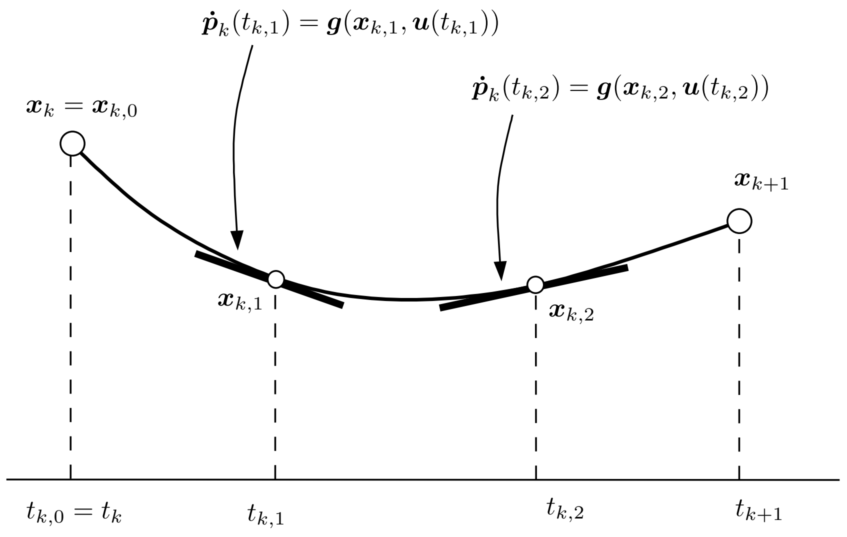
$$\boldsymbol{x}_0, \dots, \boldsymbol{x}_k, \dots, \boldsymbol{x}_N$$

and actions

$$\boldsymbol{u}_0, \dots, \boldsymbol{u}_k, \dots, \boldsymbol{u}_N$$

at those instants. The values $t_0, \dots, t_N$ are known as the knot points, and we assume them to be uniformly spaced for simplicity, so $\Delta t = t_{k+1} - t_k$ takes the same value for $k = 0, \dots, N-1$. If Δt is constant, the time horizon t_f is fixed, but if Δt is a decision variable of the problem, the time horizon will (also) be variable [102].

The transcriptions of Eqs. (4.3a) and (4.3e) are relatively straightforward and less relevant for the purpose of this chapter. They can be done, for example, by approximating the integral in Eq. (4.3a) by means of some quadrature rule, and by enforcing $\mathbf{0} \leq \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ for $k = 0, \dots, N$. Essentially, the different methods differ only on how Eq. (4.3f) is transcribed, so we devote the rest of this section to recall background techniques to this end.



$\dot{\boldsymbol{p}}_k(t_{k,1}) = \boldsymbol{g}(\boldsymbol{x}_{k,1}, \boldsymbol{u}(t_{k,1}))$
$\dot{\boldsymbol{p}}_k(t_{k,2}) = \boldsymbol{g}(\boldsymbol{x}_{k,2}, \boldsymbol{u}(t_{k,2}))$
$\boldsymbol{x}_k = \boldsymbol{x}_{k,0}$
$\boldsymbol{x}_{k+1}$
$\boldsymbol{x}_{k,1}$
$\boldsymbol{x}_{k,2}$
$t_{k,0} = t_k$
$t_{k,1}$
$t_{k,2}$
t_{k+1}

Figure 4.2: Collocation scheme for $d = 2$ in the interval $[t_k, t_{k+1}]$. The derivative of the polynomial must match the dynamics at the collocation times.

4.3.2 Transcription of Differential Constraints

To approximate Eq. (4.3f), the first step is to define $\boldsymbol{u}(t)$ in terms of $\boldsymbol{u}_0, \ldots, \boldsymbol{u}_N$. While many choices are possible here, we use a first-order hold filter due to its good balance between simplicity and accuracy of representation. For all $t \in [t_k, t_{k+1}]$ it will thus be

$$\boldsymbol{u}(t) = \boldsymbol{u}_k + \frac{t - t_k}{\Delta t} \cdot (\boldsymbol{u}_{k+1} - \boldsymbol{u}_k). \tag{4.4}$$

The second step is to determine the state $\boldsymbol{x}_{k+1}$ that would be reached from $\boldsymbol{x}_k$ under the actions in Eq. (4.4). To this end we can use shooting or collocation methods, but in this work we opt for the latter because they tend to be numerically preferable, and they allow larger regions of convergence for the optimizer.

Transcription via collocation works as follows. The form of $\boldsymbol{x}(t) \in [t_k, t_{k+1}]$ is not known a priori, but we assume it to be approximated by a polynomial $\boldsymbol{p}_k(t)$ of degree d taking the value $\boldsymbol{x}_k$ for $t = t_k$. Without loss of generality, we regard this polynomial as the one that interpolates $d + 1$ states

$$\boldsymbol{x}_{k,0}, \ldots, \boldsymbol{x}_{k,d}$$

corresponding to $d + 1$ time instants

$$t_{k,0}, \ldots, t_{k,d}$$

in the interval $[t_k, t_{k+1}]$ (Fig. 4.2), where $t_{k,0} = t_k$, and

$$t_k \leq t_{k,1} < \cdots < t_{k,d} \leq t_{k+1}.$$

Using Lagrange's interpolation formula [103] we thus can write

$$\boldsymbol{p}_k(t) = \boldsymbol{x}_{k,0} \cdot \ell_0(t - t_k) + \cdots + \boldsymbol{x}_{k,d} \cdot \ell_d(t - t_k), \tag{4.5}$$

where $\ell_0(t-t_k), \ldots, \ell_d(t-t_k)$ is the basis of Lagrange polynomials of degree d defined in $[0, \Delta t]$, which only depend on $t_{k,0}, \ldots, t_{k,d}$. Recall here that

$$\ell_j(t_{k,i} - t_k) = \begin{cases} 1, & \text{if } j = i \\ 0, & \text{otherwise} \end{cases}$$

so $\boldsymbol{p}_k(t_{k,i}) = \boldsymbol{x}_{k,i}$ [103]. Since we wish $\boldsymbol{p}_k(t_k) = \boldsymbol{x}_k$, it must be $\boldsymbol{x}_{k,0} = \boldsymbol{x}_k$ in Eq. (4.5), and we shall assume so hereafter. The remaining parameters $\boldsymbol{x}_{k,1}, \ldots, \boldsymbol{x}_{k,d}$ are unknown, but they can be determined by forcing $\boldsymbol{p}_k(t)$ to satisfy $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ for all $t = t_{k,1}, \ldots, t_{k,d}$, which yields the d collocation constraints

$$\dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}) \quad \text{for} \quad i = 1, \ldots, d, \tag{4.6}$$

where $\boldsymbol{u}_{k,i} = \boldsymbol{u}(t_{k,i})$. The values $t_{k,1}, \ldots, t_{k,d}$ are called the collocation points and they are fixed beforehand. The state $\boldsymbol{x}_{k+1}$ is then given by the continuation constraint

$$\boldsymbol{x}_{k+1} = \boldsymbol{p}_k(t_{k+1}). \tag{4.7}$$

In each interval $[t_k, t_{k+1}]$, Eq. (4.3f) is thus transcribed into Eqs. (4.6) and (4.7), with $\boldsymbol{p}_k(t)$ being defined by Eq. (4.5). In the end, once the transcribed problem is solved, the trajectory $\boldsymbol{x}(t)$ will be the continuous spline defined by $\boldsymbol{p}_0(t), \ldots, \boldsymbol{p}_{N-1}(t)$.

It is worth adding that the left-hand side of Eq. (4.6) is easy to formulate, as $\dot{\boldsymbol{p}}_k(t_{k,i})$ can always be written in the form

$$\dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{D}(t_{k,0}, \ldots, t_{k,d}) \cdot \begin{bmatrix} \boldsymbol{x}_{k,0} \\ \vdots \\ \boldsymbol{x}_{k,d} \end{bmatrix},$$

where $\boldsymbol{D}$ is a constant differentiation matrix that solely depends on $t_{k,0}, \ldots, t_{k,d}$ [86, 103].

The accuracy of the transcription depends on the particular choice of collocation points $t_{k,1}, \dots, t_{k,d}$ and the order d of $\boldsymbol{p}_k(t)$. These can be selected according to multiple schemes, like those leading to the trapezoidal or Hermite-Simpson rules, or any of the orthogonal collocation methods [86]. In this work, we choose the Gauss-Legendre orthogonal collocation scheme due to its low integration errors [26]. The order of the resulting integration is $O(\Delta t^{2d})$, which is the maximum possible for d collocation points. Also, the method is symmetric, A- and B-stable, and symplectic [104], which results in accurate transcriptions.

4.4 Conventional Collocation Schemes

We next review the two conventional ways of transcribing Problem (4.3). The two methods are referred to as the basic and Baumgarte methods (Sections 4.4.1 and 4.4.2 respectively).

4.4.1 Basic Collocation

A naive way of transcribing Problem (4.3) consists in directly applying the methods in the previous section to each one of its equations. This results in the optimization problem

$$\begin{aligned}
&\underset{\boldsymbol{w}}{\text{minimize}} && C(\boldsymbol{w}) && \text{(4.8a)}\\
&\text{subject to} && \boldsymbol{x}_0 = \boldsymbol{x}_s, && \text{(4.8b)}\\
& && \boldsymbol{x}_N = \boldsymbol{x}_g, && \text{(4.8c)}\\
& && \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), \quad k = 0, \dots, N-1, \quad i = 1, \dots, d, && \text{(4.8d)}\\
& && \boldsymbol{p}_k(t_{k+1}) = \boldsymbol{x}_{k+1}, \quad k = 0, \dots, N-1, && \text{(4.8e)}\\
& && \boldsymbol{0} \leq \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, \dots, N, && \text{(4.8f)}
\end{aligned}$$

where $C(\boldsymbol{w})$ is the discrete version of the integral cost in Eq. (4.3a), and $\boldsymbol{w}$ encompasses all the decision variables intervening: the actions $\boldsymbol{u}_0, \dots, \boldsymbol{u}_N$, the states $\boldsymbol{x}_0, \dots, \boldsymbol{x}_N$, and all collocation states $\boldsymbol{x}_{k,i}$ for $k = 0, \dots, N-1$, and $i = 1, \dots, d$.

While the previous transcription is straightforward, it also makes the problem difficult to solve. Notice that Eq. (4.8c) contains $n_x - d_{\mathcal{X}}$ redundant constraints (as $\boldsymbol{x}_N$ is implicitly restricted to $\mathcal{X}$ by Eqs. (4.8d) and (4.8e)), so the formulation violates the linear independence constraint qualification (LICQ) required by the Karush-Kuhn-Tucker conditions of optimality [105]. A way around this problem consists in replacing Eq. (4.8c) by

$$\boldsymbol{U}_g^\top (\boldsymbol{x}_N - \boldsymbol{x}_g) = \boldsymbol{0}, \tag{4.9}$$

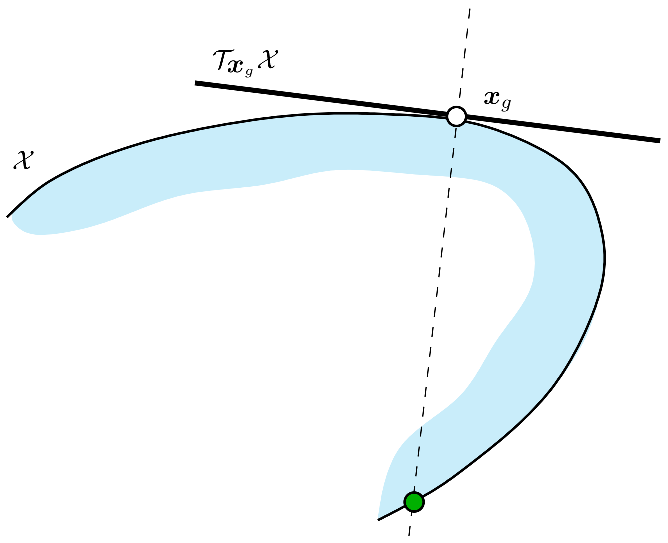


Figure 4.3: Equation (4.9) constrains $\boldsymbol{x}_N$ to lie in the normal space of $\mathcal{X}$ at $\boldsymbol{x}_g$ (the dashed ray in the figure). Since this space can intercept $\mathcal{X}$ at points different from $\boldsymbol{x}_g$ (for example the green point), $\boldsymbol{x}_N$ may converge to undesired states in the basic collocation method.

where $\boldsymbol{U}_g$ is an $n_x \times d_{\mathcal{X}}$ matrix whose columns provide an orthogonal basis of $\mathcal{T}_{\boldsymbol{x}_g}\mathcal{X}$ (Fig. 4.3). An advantage of Eq. (4.9) is that it locally constrains $\boldsymbol{x}_N$ to $\boldsymbol{x}_g$ by adding $d_{\mathcal{X}}$ independent constraints only. However, since Eq. (4.9) also holds for points in $\mathcal{X}$ different from $\boldsymbol{x}_g$, we also include a small penalty proportional to

$$\|\boldsymbol{x}_N - \boldsymbol{x}_g\|^2$$

in $C(\boldsymbol{w})$ to favor the convergence of $\boldsymbol{x}_N$ to $\boldsymbol{x}_g$. By applying these modifications to Problem (4.8) we obtain a new optimization problem with the following form:

$$\begin{aligned}
&\underset{\boldsymbol{w}}{\text{minimize}} && C(\boldsymbol{w}) && && \text{(4.10a)}\\
&\text{subject to} && \boldsymbol{x}_0 = \boldsymbol{x}_s, && && \text{(4.10b)}\\
& && \boldsymbol{U}_g^\top(\boldsymbol{x}_N - \boldsymbol{x}_g) = \boldsymbol{0}, && && \text{(4.10c)}\\
& && \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), && k = 0,\dots,N-1, \quad i = 1,\dots,d, && \text{(4.10d)}\\
& && \boldsymbol{p}_k(t_{k+1}) = \boldsymbol{x}_{k+1}, && k = 0,\dots,N-1, && \text{(4.10e)}\\
& && \boldsymbol{0} \le \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), && k = 0,\dots,N. && \text{(4.10f)}
\end{aligned}$$

The process that consists in transcribing Problem (4.3) into Eqs. (4.10) will be called the basic collocation method hereafter.

While the basic method is simple, it presents an important limitation. Note that the states $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N$ tend to drift off from $\mathcal{X}$ due to the discretization errors inherent to Eqs. (4.10d) and (4.10e), which leads to unrealistic trajectories that can be difficult to control, or terminate far from $\boldsymbol{x}_g$ (see Section 4.7 for an example). Unfortunately, this problem cannot be solved by just adding

$$\boldsymbol{F}(\boldsymbol{x}_k) = \mathbf{0}, \quad k = 1, \ldots, N \tag{4.11}$$

to the transcribed formulation, as Eqs. (4.10d) and (4.10e) already determine $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ once $\boldsymbol{x}_0$ and a control sequence is fixed. Thus, Eqs. (4.11) would again introduce redundant constraints violating the LICQ condition. Despite the problem of drift, however, the simplicity of the basic method makes it a good tool to provide, e.g., initial guesses for more elaborate methods.

4.4.2 Collocation with Baumgarte Stabilization

An alternative to the basic method is to resort to Baumgarte stabilization. This technique consists in altering the system dynamics in Eq. (4.1b) by adding artificial forces that damp the trajectory deviations from $\mathcal{X}$. To this end, recall from Section 2.3.2 that in order to obtain Eq. (4.1b), we combined the acceleration constraint

$$\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} - \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathbf{0}, \tag{4.12}$$

with the dynamic equation

$$\boldsymbol{M}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^{\top} \boldsymbol{\lambda} = \boldsymbol{\tau}_{\text{FD}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}). \tag{4.13}$$

The Baumgarte method basically consists on modifying Eq. (4.12) by adding stabilizing terms for the residuals of the position and velocity constraints. The modified equation then takes the form

$$\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \ddot{\boldsymbol{q}} - \boldsymbol{\xi}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \alpha'\, \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})\, \dot{\boldsymbol{q}} + \beta'\, \boldsymbol{\Phi}(\boldsymbol{q}) = \mathbf{0}, \tag{4.14}$$

where α' and β' are constant parameters that have to be tuned to ensure a stable trajectory. The modified acceleration constraint in Eq. (4.14) is combined with Eq. (4.13) to obtain a stabilized dynamics equation

$$\dot{\boldsymbol{x}} = \boldsymbol{g}_{\text{stab}}(\boldsymbol{x}, \boldsymbol{u}). \tag{4.15}$$

Using Baumgarte stabilization, thus, the transcription in Problem (4.10) is improved by replacing Eq. (4.10d) by

$$\dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}_{\text{stab}}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}) \quad \text{for} \quad i = 1, \dots, d. \tag{4.16}$$

The resulting problem then takes the form

$$\begin{aligned}
&\underset{\boldsymbol{w}}{\text{minimize}} && C(\boldsymbol{w}) && && (4.17\text{a})\\
&\text{subject to} && \boldsymbol{x}_0 = \boldsymbol{x}_s, && && (4.17\text{b})\\
& && \boldsymbol{U}_g^\top(\boldsymbol{x}_N - \boldsymbol{x}_g) = \boldsymbol{0}, && && (4.17\text{c})\\
& && \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}_{\text{stab}}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), && k = 0, \dots, N-1, \quad i = 1, \dots, d, && (4.17\text{d})\\
& && \boldsymbol{p}_k(t_{k+1}) = \boldsymbol{x}_{k+1}, && k = 0, \dots, N-1, && (4.17\text{e})\\
& && \boldsymbol{0} \le \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), && k = 0, \dots, N. && (4.17\text{f})
\end{aligned}$$

The simplicity of this technique has made it a common approach to mitigate the problem of drift, but the method also has several drawbacks. First of all, the tuning of α' and β' in Eq. (4.14) is not trivial, as the dynamics from the stabilizing terms should be faster than the drift dynamics produced by the transcribed equations, otherwise the trajectory might depart from $\mathcal{X}$ in an unstable manner. So far, however, only heuristic rules are available to choose α' and β' [63]. Note also that Eq. (4.15) has a more complex structure than the one in Eq. (4.1b), which complicates the computation of gradients and Hessians needed by the optimizer. The approach also alters the energy of the system artificially. In fact, the use of Eq. (4.16) does not guarantee that the original collocation constraint in Eq. (4.6) will be fulfilled, so some dynamic error is to be expected at the collocation points (see Section 4.7 for an example).

4.5 New Collocation Schemes

We next propose two methods to solve the problem of drift. In both methods, Problem (4.8) is modified by replacing the collocation and continuation constraints in Eqs. (4.8d) and (4.8e) with alternative transcriptions of the dynamics in Eq. (4.3f). While the first method guarantees that $\boldsymbol{x}_0, \dots, \boldsymbol{x}_N$ will lie on $\mathcal{X}$ exactly (Section 4.5.1), the second achieves full drift elimination over the entire trajectory $\boldsymbol{x}(t)$ (Section 4.5.2). These two methods are called the projection and local coordinates methods respectively.

4.5.1 The Projection Method

This method cancels the drift at each knot point by projecting the end state of each Lagrange interpolation orthogonally to $\mathcal{X}$. The method only requires modifying Problem (4.8) as follows. We first add N additional states

$$\boldsymbol{x}'_1, ..., \boldsymbol{x}'_N,$$

and N auxiliary vectors

$$\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N,$$

to the decision variables in $\boldsymbol{w}$. We then replace the continuation constraint in Eq. (4.8e) by

$$\boldsymbol{x}'_{k+1} = \boldsymbol{p}_k(t_{k+1}) \qquad k = 0, \ldots, N-1,$$

so the end state of the Lagrange interpolation is now $\boldsymbol{x}'_{k+1}$, and introduce the constraints

$$\begin{aligned} \boldsymbol{x}_{k+1} = \boldsymbol{x}'_{k+1} + \boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_{k+1})^\top \boldsymbol{\mu}_k \qquad & k = 0, \ldots, N-1, \\ \boldsymbol{F}(\boldsymbol{x}_{k+1}) = \boldsymbol{0} \qquad & k = 0, \ldots, N-1, \end{aligned}$$

to ensure that $\boldsymbol{x}_{k+1}$ is the orthogonal projection of $\boldsymbol{x}'_{k+1}$ on $\mathcal{X}$ (Fig. 4.4). Note that the rows of $\boldsymbol{F}_{\boldsymbol{x}}$ provide a basis of the normal space to $\mathcal{X}$ at $\boldsymbol{x}_{k+1}$, so $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_{k+1})^\top \boldsymbol{\mu}_k$ will be a normal vector of $\mathcal{X}$ at $\boldsymbol{x}_{k+1}$. Thus, although $\boldsymbol{x}'_1, ..., \boldsymbol{x}'_N$ may deviate from $\mathcal{X}$, the drift will not accumulate because the joint effect of Eqs. (4.18) will fully remove it after every time step. The modified Problem (4.8) then takes the form

$$\begin{aligned}
&\underset{\boldsymbol{w}}{\text{minimize}} && C(\boldsymbol{w}) && && \text{(4.19a)}\\
&\text{subject to} && \boldsymbol{x}_0 = \boldsymbol{x}_s, && && \text{(4.19b)}\\
& && \boldsymbol{x}_N = \boldsymbol{x}_g, && && \text{(4.19c)}\\
& && \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), && k = 0, \ldots, N-1, \quad i = 1, \ldots, d, && \text{(4.19d)}\\
& && \boldsymbol{p}_k(t_{k+1}) = \boldsymbol{x}'_{k+1}, && k = 0, \ldots, N-1, && \text{(4.19e)}\\
& && \boldsymbol{x}_{k+1} = \boldsymbol{x}'_{k+1} + \boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_{k+1})^\top \boldsymbol{\mu}_k, && k = 0, \ldots, N-1, && \text{(4.19f)}\\
& && \boldsymbol{F}(\boldsymbol{x}_{k+1}) = \boldsymbol{0}, && k = 0, \ldots, N-1, && \text{(4.19g)}\\
& && \boldsymbol{0} \le \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), && k = 0, \ldots, N. && \text{(4.19h)}
\end{aligned}$$

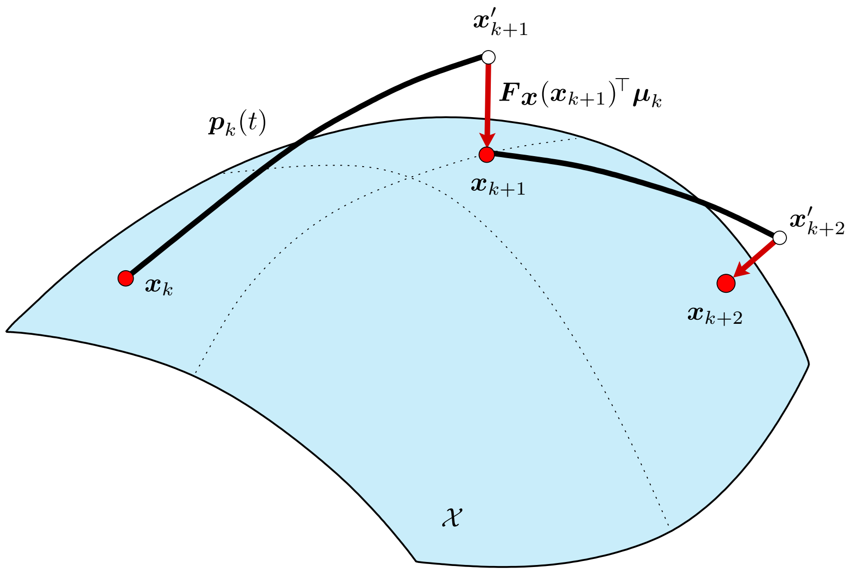


Figure 4.4: The projection method. The end state $\boldsymbol{x}'_{k+1}$ of each polynomial $\boldsymbol{p}_k(t)$ is projected orthogonally to $\mathcal{X}$ to remove the accumulated drift.

The main advantage of the projection method over the basic and Baumgarte methods is that it keeps $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ exactly on $\mathcal{X}$ while satisfying $\boldsymbol{x}_N = \boldsymbol{x}_g$ precisely. In contrast to the Baumgarte method, moreover, this is achieved without modifying the system dynamics artificially. The weak point is that some drift from $\mathcal{X}$ will persist despite the projections, as $\boldsymbol{x}(t)$ will have a shape similar to the one in Fig. 4.1(c), with discontinuities at the knot points. If a driftless continuous trajectory is required, however, we can always resort to the following method.

4.5.2 The Local Coordinates Method

The idea of this method is to transcribe the dynamics in Eq. (4.3f) using integration in local coordinates. This tool was already used in the previous chapter to keep the trajectories $\boldsymbol{x}(t)$ on $\mathcal{X}$, and we now show how the same concept can be applied to trajectory optimization. To make the presentation self-contained, a few concepts from Chapter 3 will be recalled where needed. We first derive the collocation constraints using generic maps and then we rewrite the constraints using tangent-space versions of such maps.

Collocation in Local Coordinates

Let $\boldsymbol{y} = \varphi_k(\boldsymbol{x})$ be a local chart of $\mathcal{X}$ defined around some point $\boldsymbol{x}_k \in \mathcal{X}$, so

$$\varphi_k : V_k \to P_k,$$

where V_k and P_k are open neighborhoods in $\mathcal{X}$ and $\mathbb{R}^{d_\mathcal{X}}$ respectively, including $\boldsymbol{x}_k$ and $\varphi_k(\boldsymbol{x}_k)$. Without loss of generality, we will assume that $\varphi_k(\boldsymbol{x}_k) = \mathbf{0}$, so φ_k maps $\boldsymbol{x}_k$ to the origin of $\mathbb{R}^{d_\mathcal{X}}$. Also let

$$\psi_k : P_k \to V_k$$

be the inverse map of φ_k, which provides a local parametrization of $V_k \subset \mathcal{X}$.

Using φ_k and ψ_k we can use integration in local coordinates to compute the state $\boldsymbol{x}_{k+1}$ that would be reached from $\boldsymbol{x}_k$ while ensuring $\boldsymbol{F}(\boldsymbol{x}_{k+1}) = \mathbf{0}$. To this end, we first take the time derivative of $\boldsymbol{y} = \varphi_k(\boldsymbol{x})$ to get

$$\dot{\boldsymbol{y}} = \frac{\partial \varphi_k(\boldsymbol{x})}{\partial \boldsymbol{x}} \cdot \dot{\boldsymbol{x}},$$

and using $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ we obtain

$$\dot{\boldsymbol{y}} = \frac{\partial \varphi_k(\boldsymbol{x})}{\partial \boldsymbol{x}} \cdot \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}). \tag{4.20}$$

The substitution of $\boldsymbol{x} = \psi_k(\boldsymbol{y})$ in Eq. (4.20) then yields

$$\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}_k(\boldsymbol{y}, \boldsymbol{u}), \tag{4.21}$$

which provides the dynamic equation in $\boldsymbol{y}$ coordinates exclusively. We can use this equation as follows to compute $\boldsymbol{x}_{k+1}$ [Fig. 4.5(a)]: we first map $\boldsymbol{x}_k$ to the origin of $\mathbb{R}^{d_\mathcal{X}}$ using φ_k; we then integrate Eq. (4.21) to find the state $\boldsymbol{y}(t_{k+1}) = \boldsymbol{y}_{k+1}$ that would be reached from $\boldsymbol{y}(t_k) = \mathbf{0}$ under the actions $\boldsymbol{u}(t)$; and finally we project $\boldsymbol{y}_{k+1}$ back to $\mathcal{X}$ via

$$\boldsymbol{x}_{k+1} = \psi_k(\boldsymbol{y}_{k+1}). \tag{4.22}$$

Using this process, it is clear that $\boldsymbol{F}(\boldsymbol{x}_{k+1}) = \mathbf{0}$ by construction.

To integrate Eq. (4.21) using collocation, we assume that the trajectory $\boldsymbol{y}(t)$ is well approximated by a polynomial that starts at the origin of $\mathbb{R}^{d_\mathcal{X}}$ and interpolates d unknown states

$$\boldsymbol{y}_{k,1}, \ldots, \boldsymbol{y}_{k,d}$$

corresponding to d collocation times $t_{k,1}, \ldots, t_{k,d}$ from $[t_k, t_{k+1}]$ (see Fig. 4.5(b)). This polynomial can be written as

$$\boldsymbol{p}_k(t) = \boldsymbol{y}_{k,1} \cdot \ell_1(t - t_k) + \cdots + \boldsymbol{y}_{k,d} \cdot \ell_d(t - t_k),$$

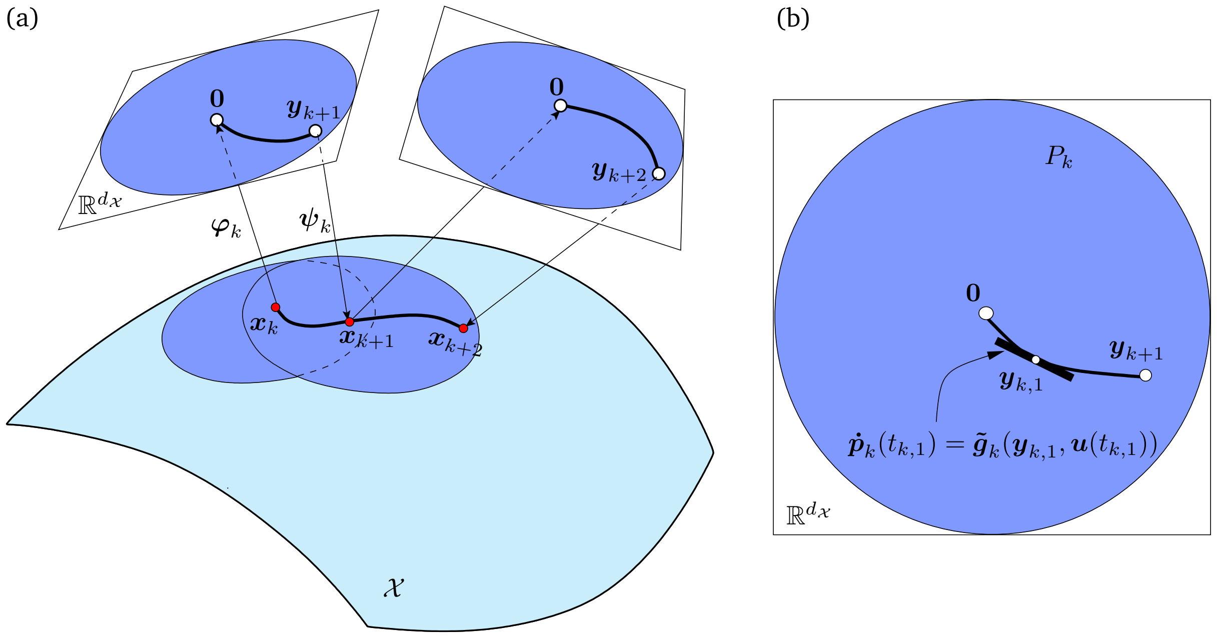


Figure 4.5: The local coordinates method. (a) To obtain $\boldsymbol{x}_{k+1}$, we first map $\boldsymbol{x}_k$ to $\boldsymbol{\varphi}_k(\boldsymbol{x}_k) = \mathbf{0} \in \mathbb{R}^{d_\mathcal{X}}$, we then integrate Eq. (4.21) to find $\boldsymbol{y}_{k+1}$, and we finally project $\boldsymbol{y}_{k+1}$ to $\mathcal{X}$ using $\boldsymbol{\psi}_k$. (b) The collocation constraint is now enforced in $\mathbb{R}^{d_\mathcal{X}}$. In the figure we assume $d = 1$, so $\dot{\boldsymbol{p}}_k(t_{k,1})$ must match $\tilde{\boldsymbol{g}}_k(\boldsymbol{y}_{k,1}, \boldsymbol{u}(t_{k,1}))$.

which satisfies $\boldsymbol{p}_k(t_k) = \mathbf{0}$ as required since $\ell_1(t - t_k), \ldots, \ell_d(t - t_k)$ are all zero for $t = t_k$. To determine $\boldsymbol{y}_{k,1}, \ldots, \boldsymbol{y}_{k,d}$, we only have to impose the d collocation constraints

$$\dot{\boldsymbol{p}}_k(t_{k,i}) = \tilde{\boldsymbol{g}}_k(\boldsymbol{y}_{k,i}, \boldsymbol{u}_{k,i}), \quad \text{for} \quad i = 1, \ldots, d.$$

The value of $\boldsymbol{y}_{k+1}$ will then be given by

$$\boldsymbol{y}_{k+1} = \boldsymbol{p}_k(t_{k+1}),$$

so Eq. (4.22) can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{\psi}_k(\boldsymbol{p}_k(t_{k+1})).$$

With the earlier procedure, thus, we can achieve a drift-free transcription of Problem (4.3) by replacing the collocation and continuation constraints in Eqs. (4.8d) and (4.8e) by

$$\begin{aligned} \dot{\boldsymbol{p}}_k(t_{k,i}) &= \tilde{\boldsymbol{g}}_k(\boldsymbol{y}_{k,i}, \boldsymbol{u}_{k,i}), \quad k = 0, \ldots, N-1, \quad i = 1, \ldots, d, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{\psi}_k(\boldsymbol{p}_k(t_{k+1})), \quad k = 0, \ldots, N-1. \end{aligned}$$

The resulting problem then takes the form

$$
\begin{aligned}
&\underset{\boldsymbol{w}}{\text{minimize}} && C(\boldsymbol{w}) && && (4.24a)\\
&\text{subject to} && \boldsymbol{x}_0 = \boldsymbol{x}_s, && && (4.24b)\\
& && \boldsymbol{x}_N = \boldsymbol{x}_g, && && (4.24c)\\
& && \dot{\boldsymbol{p}}_k(t_{k,i}) = \tilde{\boldsymbol{g}}_k(\boldsymbol{y}_{k,i}, \boldsymbol{u}_{k,i}), && k = 0, \dots, N-1, \quad i = 1, \dots, d, && (4.24d)\\
& && \boldsymbol{x}_{k+1} = \boldsymbol{\psi}_k(\boldsymbol{p}_k(t_{k+1})), && k = 0, \dots, N-1, && (4.24e)\\
& && \boldsymbol{0} \le \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), && k = 0, \dots, N. && (4.24f)
\end{aligned}
$$

Once the transcribed problem is solved we can even use $\boldsymbol{p}_1(t), \dots, \boldsymbol{p}_N(t)$ to construct a continuous spline for $\boldsymbol{x}(t)$ lying in $\mathcal{X}$ for all t, something that the earlier methods were unable to obtain. This spline will be given by

$$\boldsymbol{\psi}_0(\boldsymbol{p}_0(t)), \dots, \boldsymbol{\psi}_{N-1}(\boldsymbol{p}_{N-1}(t)).$$

Finally note that, since we have integrated the dynamics in Eq. (4.3f) using the independent $\boldsymbol{y}$ coordinates, the goal constraint in Eq. (4.24c) is not problematic in this method.

Collocation in Tangent Space Coordinates

As we know, in some robots it will be possible to define the maps $\boldsymbol{\psi}_k$ and $\boldsymbol{\varphi}_k$ using closed-form expressions. However, in order to achieve a general trajectory optimizer, and following the same criterion as in Chapter 3, we next define these maps using tangent space coordinates. When using these coordinates, recall that the map $\boldsymbol{y} = \boldsymbol{\varphi}_k(\boldsymbol{x})$ is obtained by projecting $\boldsymbol{x}$ orthogonally onto $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X} = \mathbb{R}^{d_\mathcal{X}}$ and takes the form

$$\boldsymbol{y} = \boldsymbol{U}_k(\boldsymbol{x}_k)^\top \, (\boldsymbol{x} - \boldsymbol{x}_k), \tag{4.25}$$

where $\boldsymbol{U}_k(\boldsymbol{x}_k)$ is an $n_x \times d_\mathcal{X}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$. Also recall that the inverse map $\boldsymbol{x} = \boldsymbol{\psi}_k(\boldsymbol{y})$ is implicitly determined by the system of nonlinear equations

$$\begin{cases} \boldsymbol{y} - \boldsymbol{U}_k(\boldsymbol{x}_k)^\top (\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{0}, \\ \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, \end{cases}$$

which we will write as

$$\boldsymbol{H}_k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0} \tag{4.27}$$

for convenience. The time derivative of Eq. (4.25) now provides the particular form of Eq. (4.20),

$$\dot{\boldsymbol{y}} = \boldsymbol{U}_k(\boldsymbol{x}_k)^\top \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}),$$

where $\boldsymbol{U}_k(\boldsymbol{x}_k)^\top$ corresponds to $\partial \boldsymbol{\varphi}_k(\boldsymbol{x})/\partial \boldsymbol{x}$.

Since the ψ_k map is only defined implicitly by Eq. (4.27), we cannot obtain the dynamics in Eq. (4.21) in $\boldsymbol{y}$ coordinates explicitly. However, using Eq. (4.27), we can still impose the collocation constraint in Eq. (4.24d) via

$$\begin{aligned} \boldsymbol{H}_k(\boldsymbol{x}_{k,i}, \boldsymbol{y}_{k,i}) &= \boldsymbol{0}, \quad k = 0, \dots, N-1, \quad i = 1, \dots, d, \\ \dot{\boldsymbol{p}}_k(t_{k,i}) &= \boldsymbol{U}_k(\boldsymbol{x}_k)^\top \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), \quad k = 0, \dots, N-1, \quad i = 1, \dots, d, \end{aligned}$$

and the continuation constraint in Eq. (4.24e) via

$$\boldsymbol{H}_k(\boldsymbol{x}_{k+1}, \boldsymbol{p}_k(t_{k+1})) = \boldsymbol{0}, \quad k = 0, \dots, N-1,$$

where $\boldsymbol{y}_{k,1}, \dots, \boldsymbol{y}_{k,d}$ must be added to the decision variables $\boldsymbol{w}$ of the problem. The resulting problem finally takes the form

$$\begin{aligned}
\underset{\boldsymbol{w}}{\text{minimize}} \quad & C(\boldsymbol{w}) && && \text{(4.29a)} \\
\text{subject to} \quad & \boldsymbol{x}_0 = \boldsymbol{x}_s, && && \text{(4.29b)} \\
& \boldsymbol{x}_N = \boldsymbol{x}_g, && && \text{(4.29c)} \\
& \boldsymbol{H}_k(\boldsymbol{x}_{k,i}, \boldsymbol{y}_{k,i}) = \boldsymbol{0}, && k = 0, \dots, N-1, \quad i = 1, \dots, d, && \text{(4.29d)} \\
& \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{U}_k(\boldsymbol{x}_k)^\top \boldsymbol{g}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k,i}), && k = 0, \dots, N-1, \quad i = 1, \dots, d, && \text{(4.29e)} \\
& \boldsymbol{H}_k(\boldsymbol{x}_{k+1}, \boldsymbol{p}_k(t_{k+1})) = \boldsymbol{0}, && k = 0, \dots, N-1, && \text{(4.29f)} \\
& \boldsymbol{0} \le \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k), && k = 0, \dots, N. && \text{(4.29g)}
\end{aligned}$$

4.6 Implementation Details

To compare the conventional techniques in Section 4.4 with the new ones proposed in Section 4.5, we have implemented them using the symbolic tool for nonlinear optimization and algorithmic differentiation CasADi [67]. CasADi provides the necessary means to formulate the problems, and to compute the gradients and Hessians of the transcribed equations using automatic differentiation. These derivatives are necessary to solve the nonlinear optimization problems that result, a task for which we rely on the interior-point solver IPOPT [106] in conjunction with the linear solver MA-27 [107]. We next discuss important details that must be taken into account when implementing and solving the transcribed problems, regardless of the software platform employed.

4.6.1 Explicit versus Implicit Dynamics

In all transcriptions so far, the collocation constraints have been formulated using the explicit form of the system dynamics in Eq. (4.1b). The derivation of Eq. (4.1b), however, requires inverting an extended mass matrix, which often complicates the expressions of the gradients and Hessians needed by the optimizer. Unless the mass matrix is simple enough, it is preferable to write the collocation constraints using the implicit form of the dynamics given by Eqs. (4.12) and (4.13). This can be done as follows.

Let us write Eqs. (4.12) and (4.13) compactly as

$$\boldsymbol{\mathcal{D}}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{\lambda}) = \boldsymbol{0}.$$

Then, in the case of the basic and projection methods, the collocation constraint in Eq. (4.10d) and (4.19d) respectively, can be replaced by

$$\begin{cases} \dot{\boldsymbol{p}}_k(t_{k,i}) = \dot{\boldsymbol{x}}_{k,i}, \\ \boldsymbol{\mathcal{D}}(\boldsymbol{x}_{k,i}, \dot{\boldsymbol{x}}_{k,i}, \boldsymbol{u}_{k,i}, \boldsymbol{\lambda}_{k,i}) = \boldsymbol{0}, \end{cases}$$

where $\dot{\boldsymbol{x}}_{k,i}$ and $\boldsymbol{\lambda}_{k,i}$ are decision variables now, so they will be part of $\boldsymbol{w}$.

Similarly, in the Baumgarte method, the collocation constraint in Eq. (4.17d) would be replaced by

$$\begin{cases} \dot{\boldsymbol{p}}_k(t_{k,i}) = \dot{\boldsymbol{x}}_{k,i}, \\ \boldsymbol{\mathcal{D}}_{\text{stab}}(\boldsymbol{x}_{k,i}, \dot{\boldsymbol{x}}_{k,i}, \boldsymbol{u}_{k,i}, \boldsymbol{\lambda}_{k,i}) = \boldsymbol{0}, \end{cases}$$

where $\boldsymbol{\mathcal{D}}_{\text{stab}}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{\lambda}) = \boldsymbol{0}$ includes Eq. (4.13) and the stabilized acceleration constraint in Eq. (4.14).

Finally, in the local coordinates method the collocation constraint in Eqs. (4.24d) and (4.29e) would be substituted by

$$\begin{cases} \dot{\boldsymbol{p}}_k(t_{k,i}) = \boldsymbol{U}_k(\boldsymbol{x}_k)^\top \, \dot{\boldsymbol{x}}_{k,i}, \\ \boldsymbol{\mathcal{D}}(\boldsymbol{x}_{k,i}, \dot{\boldsymbol{x}}_{k,i}, \boldsymbol{u}_{k,i}, \boldsymbol{\lambda}_{k,i}) = \boldsymbol{0}. \end{cases}$$

Clearly, the use of the implicit form of the dynamics adds more variables and equations to the transcribed problems, but the resulting expressions are simpler and numerically more stable. The optimization problem is larger but also sparser, which often improves the convergence.

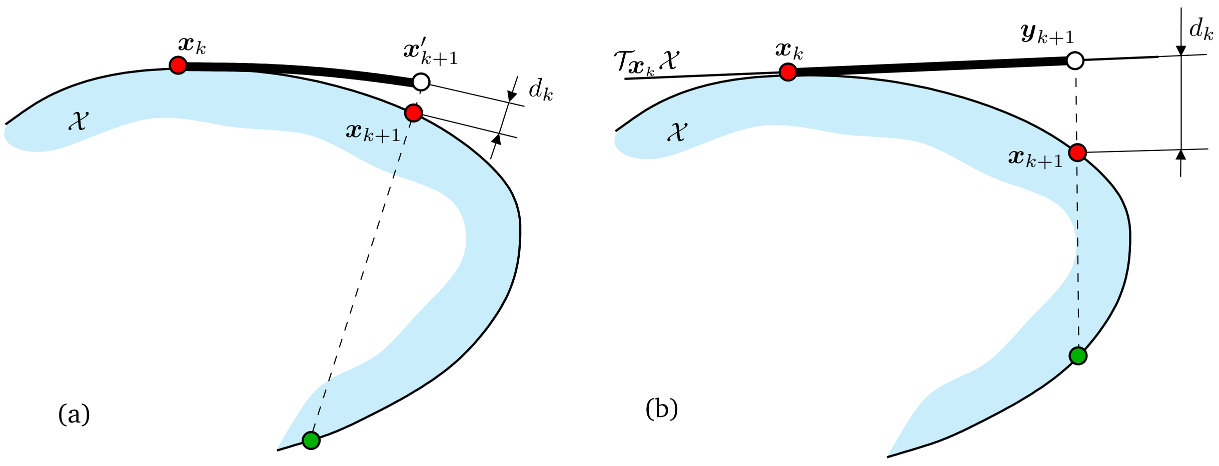


Figure 4.6: Projection rays in the projection (a) and local coordinates (b) methods. Whereas in (a) the ray is orthogonal to $\mathcal{X}$, in (b) it is orthogonal to $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$. In both cases there may be more than one solution to the projection step to $\mathcal{X}$ (red and green points). The transcriptions should ensure the optimizer selects the red point for $\boldsymbol{x}_{k+1}$, i.e., the one that is closest to the projection point (in white). See the text for details.

4.6.2 Ensuring Proper Projections

For both the projection and local coordinates methods, we may find situations in which the projection ray intercepts the $\mathcal{X}$ manifold in multiple points (Fig. 4.6). In these situations, we need to ensure that the position of $\boldsymbol{x}_{k+1}$ chosen by the optimizer (red point) is the one that is closest to the projection point (white point), otherwise the trajectory would suddenly jump to farther regions in $\mathcal{X}$ (green point). To this end, we define the distance d_k from the projection point to $\boldsymbol{x}_{k+1}$ as

$$d_k = \|\boldsymbol{x}'_{k+1} - \boldsymbol{x}_{k+1}\|$$

in the projection method, and as

$$d_k = \|(\boldsymbol{x}_k + \boldsymbol{U}_k(\boldsymbol{x}_k)\ \boldsymbol{y}_{k+1}) - \boldsymbol{x}_{k+1}\|$$

in the local coordinates method. Then, we add a small penalty term proportional to

$$\sum_{k=1}^{N} d_k^2,$$

in the cost function $C(\boldsymbol{w})$ in Eqs. (4.19a) and (4.29a). In this way, the optimizer will select the closest point to the projection point in the ray.

4.6.3 Computing a Basis of the Tangent Space

To compute a basis of $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$, i.e., of the null space of $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)$, we can use the QR decomposition. Using this decomposition, the $n_e \times n_x$ matrix $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)$ can be expressed as

$$\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)^\top = \boldsymbol{Q}_k\,\boldsymbol{R}_k, \tag{4.33}$$

where $\boldsymbol{Q}_k$ is an $n_x \times n_x$ orthonormal matrix, so $\boldsymbol{Q}_k^\top \boldsymbol{Q}_k = \boldsymbol{I}_{n_x}$, and $\boldsymbol{R}_k$ is an $n_x \times n_e$ upper triangular matrix. From Eq. (4.33) we have that

$$\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)\,\boldsymbol{Q}_k = \boldsymbol{R}_k^\top,$$

which can be written in block form as

$$\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)\,[\boldsymbol{V}_k\,\boldsymbol{U}_k] = [\boldsymbol{L}_k\,\boldsymbol{0}],$$

where $\boldsymbol{V}_k$ includes the first n_e columns of $\boldsymbol{Q}_k$, $\boldsymbol{U}_k$ includes the remaining $d_{\mathcal{X}}$ columns, and $\boldsymbol{L}_k$ is an $n_e \times n_e$ lower triangular matrix [31, 32]. Since $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)\,\boldsymbol{U}_k = \boldsymbol{0}$ and $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_{d_{\mathcal{X}}}$, $\boldsymbol{U}_k$ provides the desired orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$.

Note that the basis $\boldsymbol{U}_k$ is not unique and typical implementations of the QR decomposition apply column reordering with the aim of improving the numerical stability of the procedure. Therefore, applying the process just described on nearby points may produce significantly different bases, so the procedure does not guarantee the continuity and smoothness of the outputs. This is inconvenient for the optimization process, which requires the derivatives of $\boldsymbol{U}_k$ with respect to $\boldsymbol{x}_k$. Therefore, once $\boldsymbol{U}_k$ is computed for a given point $\boldsymbol{x}_k$, i.e., after the initialization of the optimization process, it is more convenient to use Gram-Schmidt orthonormalization to update the basis $\boldsymbol{U}_l$ for any point $\boldsymbol{x}_l$ close enough to the previous estimation of $\boldsymbol{x}_k$. In this process, the columns $\boldsymbol{u}_l^j$ of $\boldsymbol{U}_l$ for $j = \{1, \ldots, d_{\mathcal{X}}\}$ are computed in sequence from the columns $\boldsymbol{u}_k^j$ of $\boldsymbol{U}_k$ in two steps. We first compute

$$\boldsymbol{w}_l^j = \boldsymbol{u}_k^j - \boldsymbol{E}_l\,\boldsymbol{E}_l^\top\,\boldsymbol{u}_k^j,$$

with $\boldsymbol{E}_l = [\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_l)^\top\,\boldsymbol{u}_l^1 \ldots \boldsymbol{u}_l^{j-1}]$, and then obtain

$$\boldsymbol{u}_l^j = \frac{\boldsymbol{w}_l^j}{\|\boldsymbol{w}_l^j\|}.$$

The first step projects $\boldsymbol{u}_k^j$ to the null space of $\boldsymbol{E}_l$, i.e., of $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_l)$ and the vectors of $\boldsymbol{U}_l$ already computed. The second step just normalizes the resulting vector. This process is well-defined as long as none of the $\boldsymbol{w}_l^j$ vectors is zero, i.e., provided $\boldsymbol{U}_k$ and $\boldsymbol{U}_l$ are relatively similar.

4.6.4 Accuracy Metrics

To evaluate the quality of a trajectory it is essential to define proper accuracy metrics. These metrics allow us to compare the four transcription methods described in this chapter. We here use two errors functions defined in [86, 102] to quantify how well $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ satisfy the kinematic and dynamic constraints in Eqs. (4.1a) and (4.1b). The logic is that if these two equations are accurately fulfilled (both at the knot points and in between them) then the spline for $\boldsymbol{x}(t)$ will provide an accurate representation of the system motion under $\boldsymbol{u}(t)$. Therefore, the lower the errors, the lower the control effort needed to stabilize the trajectories a posteriori.

Specifically, we define the kinematic error as

$$e_K(t) = \|\boldsymbol{F}(\boldsymbol{x}(t))\|,$$

and the dynamic error as

$$e_D(t) = \|\dot{\boldsymbol{x}}(t) - \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t))\|.$$

The averages of these two errors over $[0, t_f]$,

$$E_K = \frac{1}{t_f}\int_0^{t_f} e_K(t)\,\mathrm{d}t,$$
$$E_D = \frac{1}{t_f}\int_0^{t_f} e_D(t)\,\mathrm{d}t,$$

provide global quantities summarizing the two types of errors over the whole trajectories $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$.

4.6.5 Including Obstacles

In our optimizations, we always use the trajectories from Chapter 3 as an initial guess. Since such trajectories are collision-free, a simple strategy can be applied to also obtain optimized collision-free trajectories. It consists in solving the optimization problem as if no obstacles were present, but checking at each iteration whether the current trajectory is free from collisions. If a collision is found in one iteration, we revert the trajectory to its previous form and split it at the discrete state that converged to a collision. Then, the split segments are optimized separately and the overall process is iterated until no reduction in cost is achieved. An alternative strategy is to use the shortcut method in [108], which optimizes randomly-selected segments of the trajectory while also neglecting the obstacles. If a collision is found when improving a segment, the method reverts it to its earlier version and attempts to optimize another segment, repeating the process until no improvement is possible. Both approaches are simple-enough, as they do not require the formulation of obstacle-avoidance constraints in the optimization problem. Other strategies directly formulate collision-avoidance in Problem 4.3, either as penalty terms in the cost function or as path constraints in Eq. 4.3e [15, 109]. However, both approaches have their shortcomings: while the former may converge to a collision trajectory, the latter can be hard to implement, as it requires the formulation of collision constraints using differentiable functions, which are difficult to obtain in general.

4.7 Performance Tests

We next compare the performance of the previous collocation methods in solving a trajectory optimization problem in the five-bars robot shown in Fig 3.19. The task is to lift a load from a bottom position in which the robot is at rest, to an upright position. To complicate the task, the robot is set to move on a vertical plane, so it must overcome gravity, and we limit the motor torques to a small range $[-u_{\max}, u_{\max}]$ that impedes direct trajectories to the goal. We initialize the optimization with the trajectories from the sampling-based planner of Chapter 3. These trajectories will often be jerky, and far from optimal, but usually they will be good enough to allow the convergence of the optimizer to a locally-optimal solution. All optimization problems have been solved using a MacBook Pro with an Intel i9 6-core processor running at 2.9 GHz. Except where noted, the final time t_f will be fixed beforehand.

Fig. 4.7 shows weight-lifting trajectories obtained by the four methods using $N = 137$, $\Delta t = 0.035$ [s], and the running cost $L(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \boldsymbol{u}(t)^\top \boldsymbol{u}(t)$ for Eq. (4.3a). As we see, in both trajectories the torque limits force the generation of swinging motions to reach the goal. The trajectory from the basic method shows that the drift inevitably accumulates, so the mechanism disassembles and the goal state cannot be reached (see the mismatch between the expected and obtained positions of the right motor joint in several snapshots, in red and white respectively). The trajectory from the Baumgarte method reveals that, although the drift is being stabilized (see how the 6-th snapshot presents the largest mismatch in the right joint position, and how it decreases in the last snapshots), the mechanism is still slightly disassembled at the goal. In contrast, the trajectories from the projection and local coordinates methods are able to reach the goal precisely.

The kinematic error corresponding to the previous trajectories is shown in Fig 4.8. Note that a large value of $e_K(t)$ implies that Eq. (4.1a) is violated, which results in unrealistic trajectories that are difficult to control later on. Thus, a method resulting in low values of $e_K(t)$ is preferable. From Fig. 4.8 we see that each method performs as expected. In the basic method, $e_K(t)$ accumulates over time, which in the trajectory from the basic method of Fig 4.7 corresponds to the progressive disassembly of the right motor joint. In the Baumgarte method, we

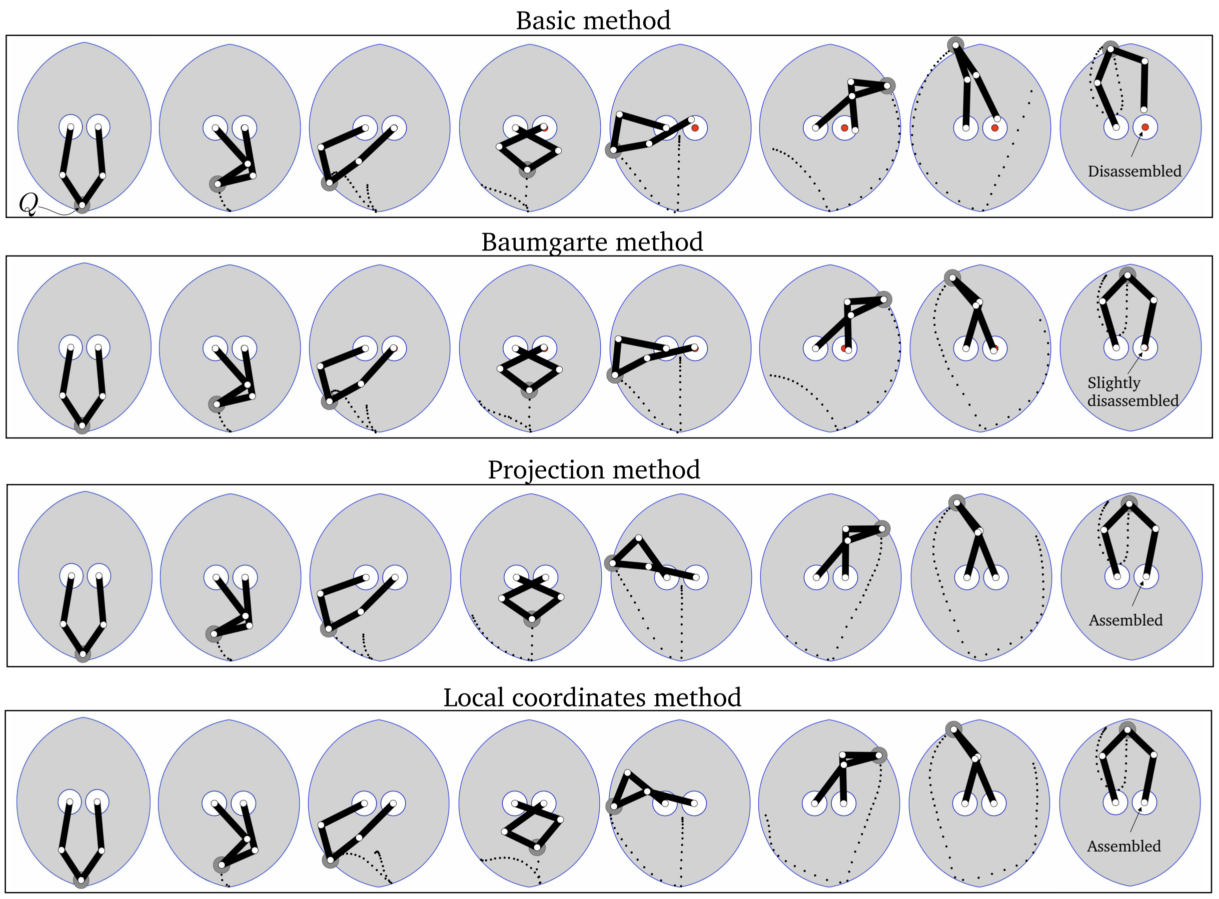


Figure 4.7: Trajectories obtained by the four methods in the weight lifting task. The grey area in each snapshot is the workspace of the robot, i.e., the set of positions that point Q can attain. The two base joints are actuated. The trajectory from the basic method shows that, due the accumulation of drift, the mechanism disassembles at the right motor joint (see the mismatch between the expected and obtained positions of this joint in the several snapshots, in red and white respectively) so Q leaves its workspace at times and the goal state cannot be reached. The trajectory from the Baumgarte method shows that, despite the stabilization of drift, the mechanism slightly disassembles at some instants, even at the goal. In the projection and local coordinates methods, in contrast, the goal state is reached exactly (compare the last snapshots in each row). See youtu.be/BYGUlpkHFXs for an animated version of this figure.

see in Fig. 4.8 that the stabilizing terms prevent the accumulation of drift, but many knot points deviate substantially from the manifold. In the projection method, the growth of $e_K(t)$ is also prevented, but now all of the knot points lie on $\mathcal{X}$ by construction. Finally, $e_K(t) = 0$ for all t

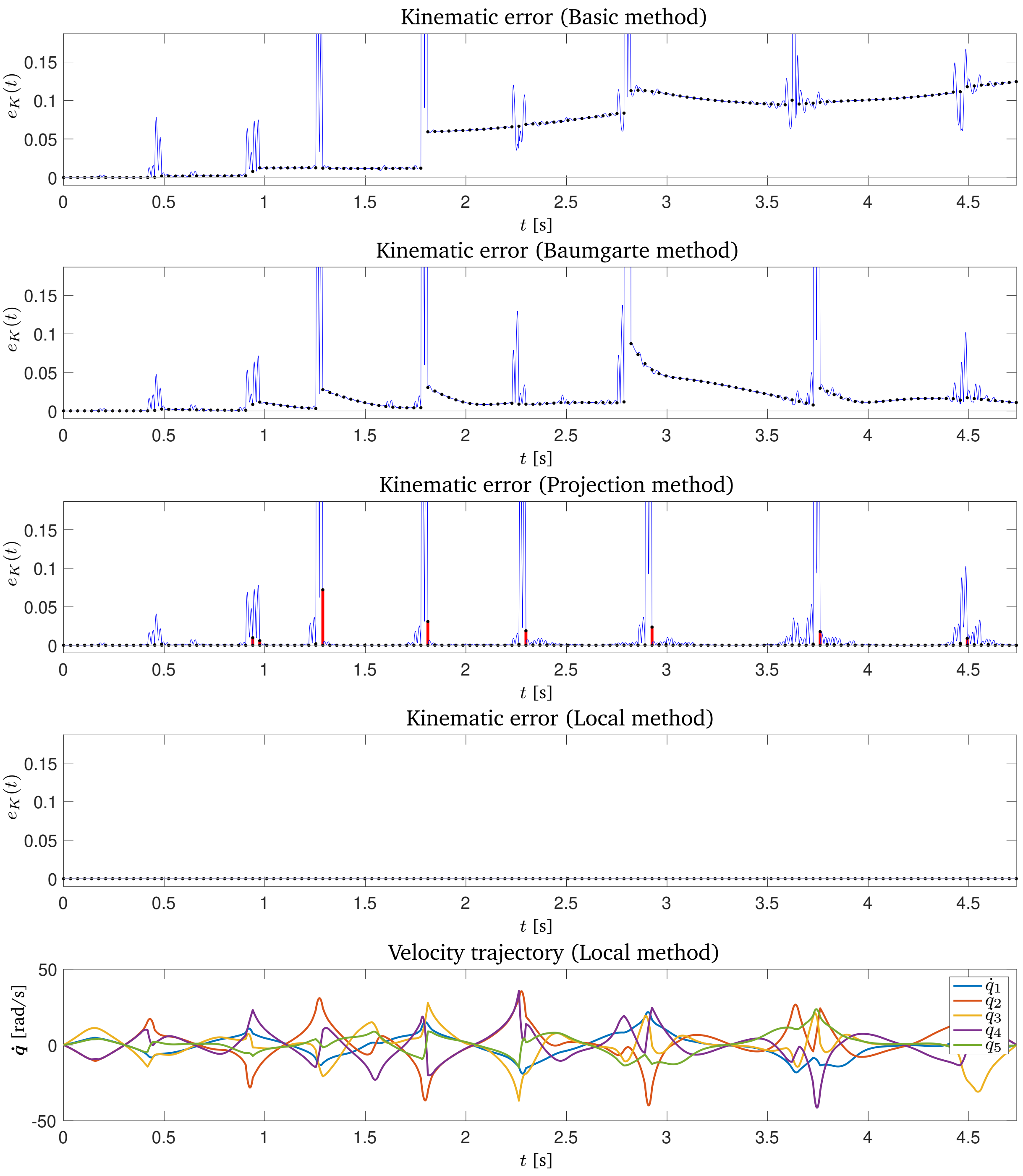


Figure 4.8: Top four plots: kinematic error $e_K(t)$ for the weight-lifting trajectories obtained with the basic, Baumgarte, projection, and local coordinates methods, respectively (shown in blue and obtained with $d = 2$). The black dots indicate the values of $e_K(t)$ at the knot points. In the third plot, the red vertical segments correspond to the projections from $\boldsymbol{x}'_k$ to $\boldsymbol{x}_k$ shown in Fig. 4.4. Bottom plot: The values taken by $\dot{q}_1, \dots, \dot{q}_5$ along the trajectory. It can be seen that, except in the local coordinates method, $e_K(t)$ tends to be larger when the robot velocities are high.

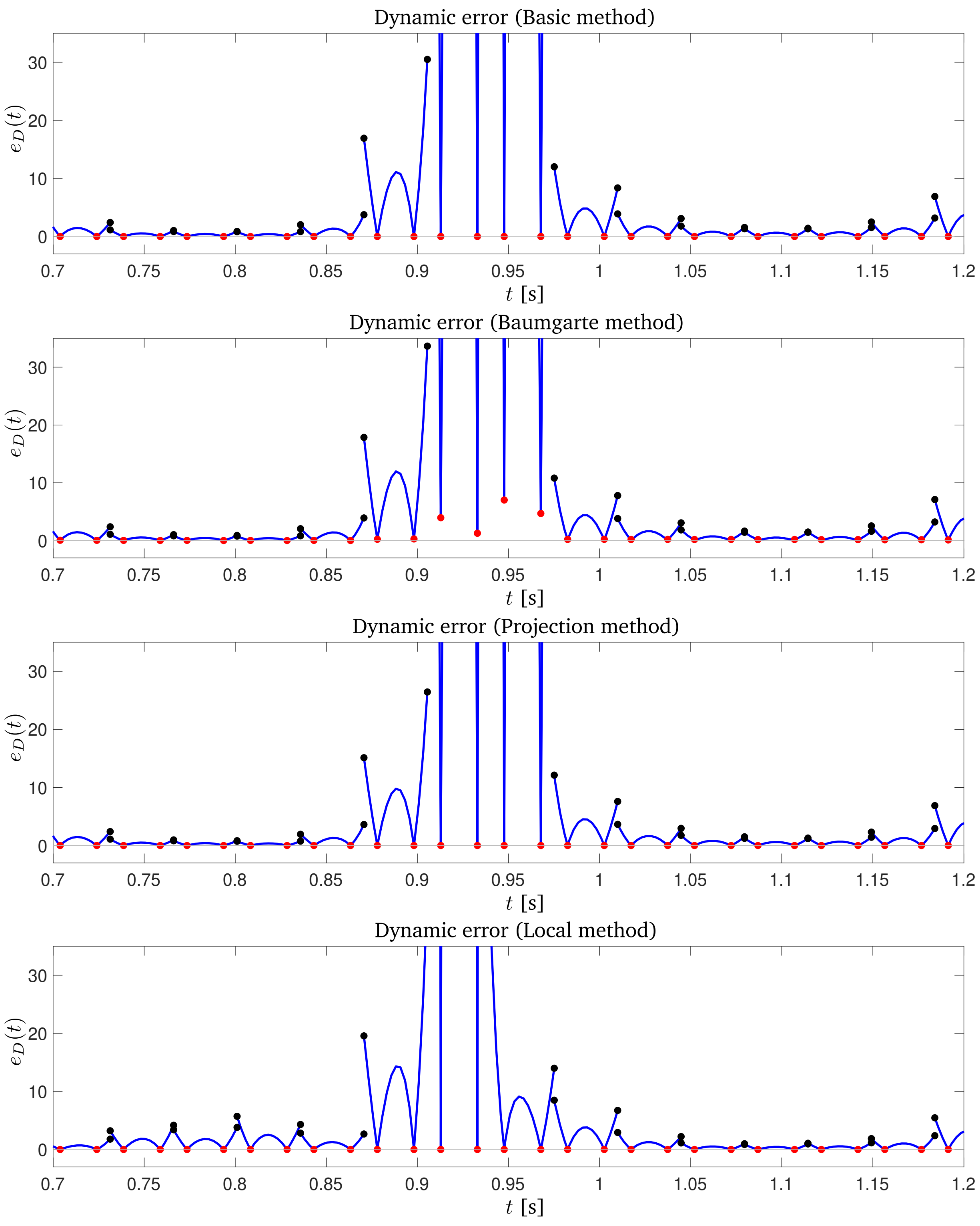


Figure 4.9: Dynamic error $e_D(t)$ for the weight-lifting task with $d = 2$. We only depict $e_D(t)$ for the time span $[1, 1.5]$ [s] to better appreciate this error at the knot points (in black) and at the collocation points (in red). The four methods yield similar errors over the whole time horizon.

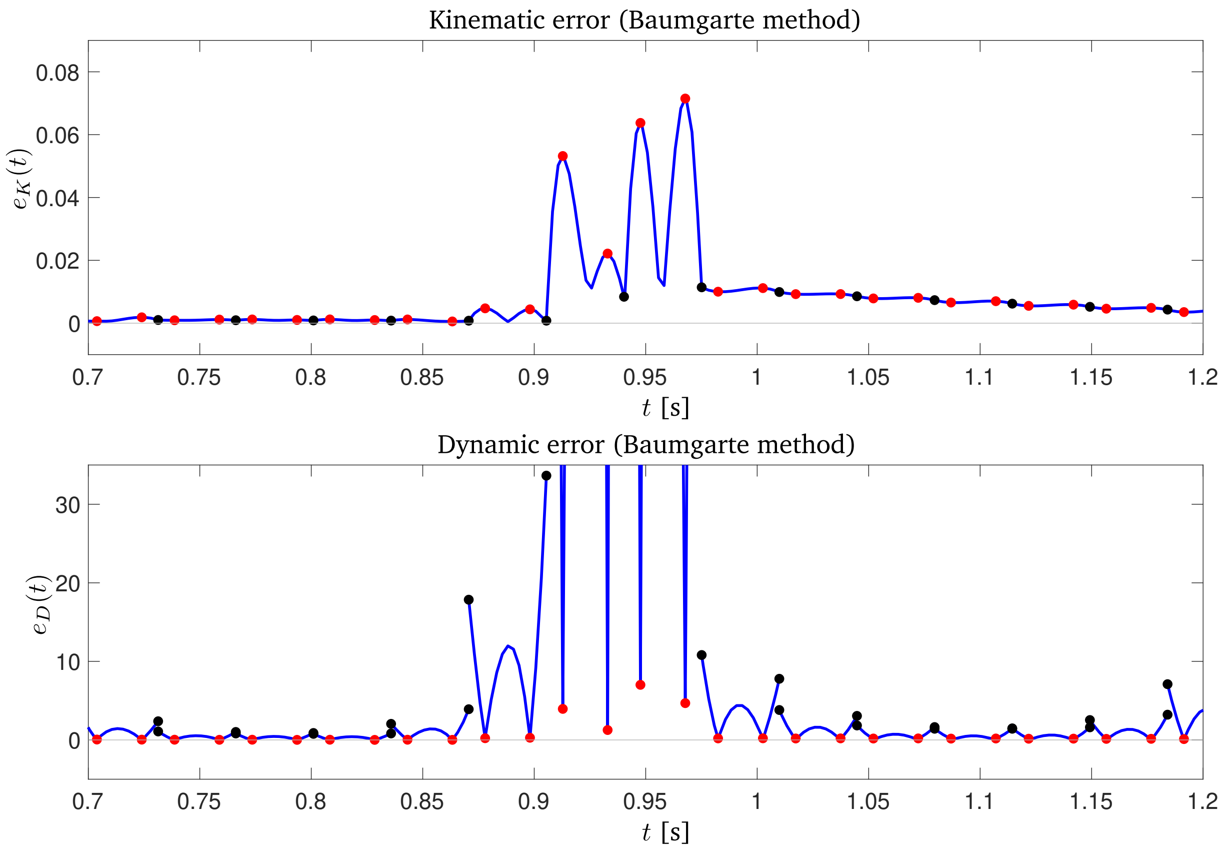


Figure 4.10: In the Baumgarte method, the presence of kinematic errors (top) introduces non-null values for the stabilizing terms in Eq. (4.14). Thus the real dynamics is violated at the collocation points (bottom). The black and red dots depict the error at the knot and collocation points, respectively.

in the local coordinates method, so this is the most accurate of all methods. It is worth noting that the error peaks arising in the basic, Baumgarte, and projection methods coincide with trajectory times in which the values of $\dot{\boldsymbol{q}}$ are high (compare the error peaks with the bottom plot in Fig. 4.8), a phenomenon to which the local coordinates method is immune.

Fig. 4.9 shows that the reduction in kinematic error of the projection and local coordinates methods does not result in increments of the dynamic error $e_D(t)$. This error gives an idea of how well the computed trajectories respect the robot dynamics, i.e. the vector field defined by Eq. (4.1b). As we see, the methods do not differ significantly on this regard, as they all resort to the same scheme of numerical integration (Gauss-Legendre collocation with $d = 2$ and a same time step Δt). In principle, since the solution trajectories satisfy the collocation constraints, the dynamic error should be zero at the collocation points (shown in red). This is the case for all methods except for the Baumgarte method. The reason is that this method

Method	d	E_K	E_D	C^*	t_{opt} [s]
Naive	2	$69.06 \cdot 10^{-3}$	75.31	1.04	4.28
	3	$15.00 \cdot 10^{-3}$	47.03	1.18	9.63
	4	$9.18 \cdot 10^{-3}$	42.61	1.19	18.74
Baumgarte	2	$29.70 \cdot 10^{-3}$	88.64	0.98	4.12
	3	$7.69 \cdot 10^{-3}$	43.93	1.20	11.25
	4	$5.04 \cdot 10^{-3}$	35.81	1.38	19.60
Projection	2	$15.19 \cdot 10^{-3}$	75.19	0.54	4.63
	3	$7.35 \cdot 10^{-3}$	51.69	1.07	9.11
	4	$3.73 \cdot 10^{-3}$	34.30	1.33	12.42
Local	2	$4.66 \cdot 10^{-9}$	48.04	1.61	22.18
	3	$5.93 \cdot 10^{-9}$	36.07	1.70	29.89
	4	$5.74 \cdot 10^{-9}$	21.07	1.81	37.54

Table 4.1: Weight lifting: Performance statistics of the different methods when d increases.

forces the satisfaction of the modified dynamics in Eq. (4.15) rather than the actual dynamics of Eq. (4.1b). In segments where the kinematic errors are high, the stabilizing terms are not zero at the collocation points, which yields a non-vanishing value of $e_D(t)$ at such points. This phenomenon is better illustrated in Fig. 4.10.

Table 4.1 provides global performance measures for the four methods when using polynomials of increasing degree d. For each value of d we provide the average kinematic and dynamic errors E_K and E_D given by Eqs. (4.34) and (4.35), the cost C^* of the computed trajectory, and the CPU time t_{opt} used by the optimizer. As expected, both E_K and E_D decrease when increasing d in each method. For a same value of d, the projection method has a value of E_K that is slightly smaller to that of the basic and Baumgarte methods. In contrast, the local coordinates method has a negligible E_K value, which only depends on the tolerance used by the optimizer when solving Eq. (4.27). For a same d, E_D is comparable in all methods.

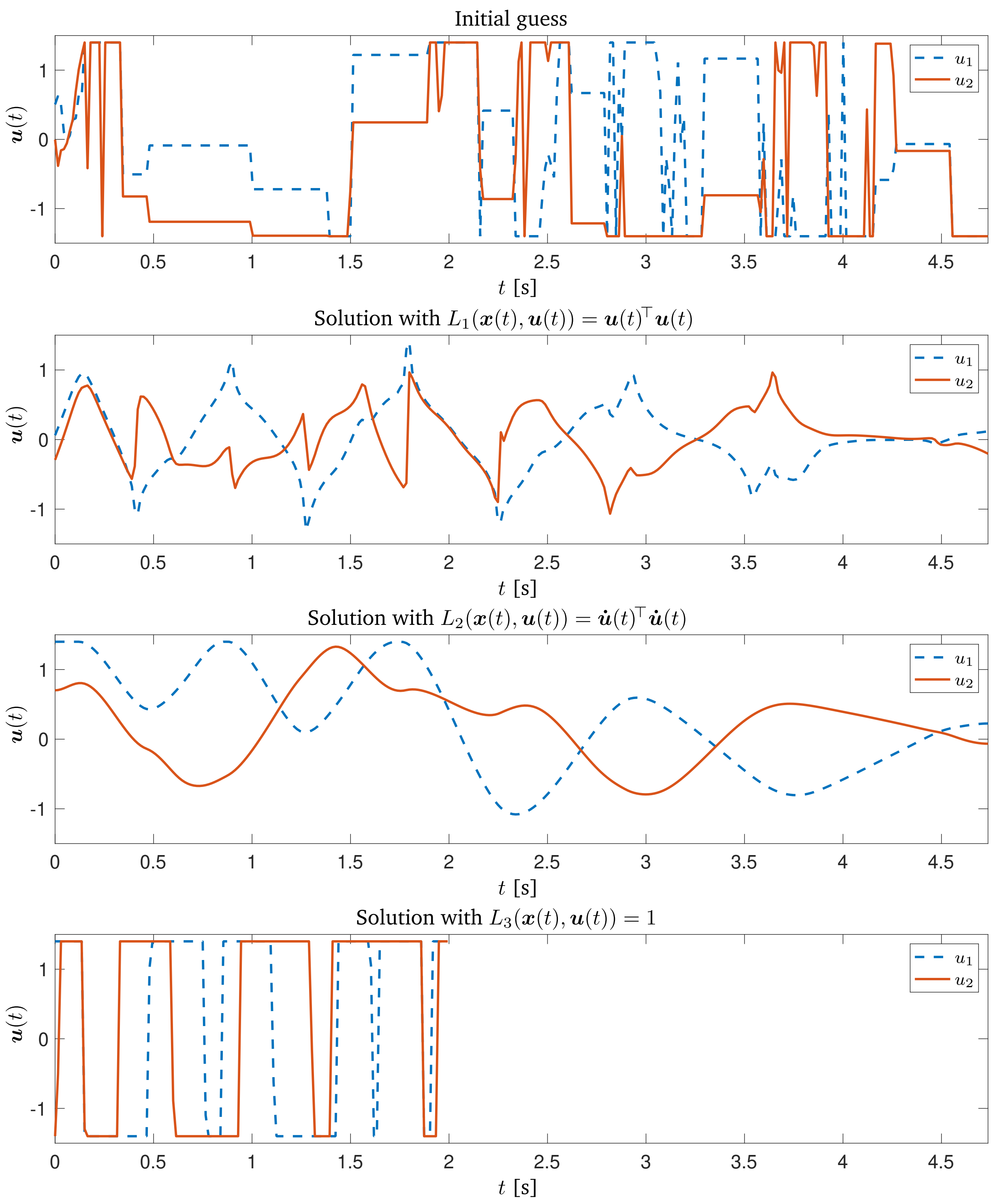


Figure 4.11: Top plot: the initial guess of the action trajectory $\boldsymbol{u}(t)$. Bottom plots: the optimized function $\boldsymbol{u}(t)$ obtained by the projection method with $d = 4$ when using different running costs $L(\boldsymbol{x}(t), \boldsymbol{u}(t))$. See youtu.be/HaUhwB_alCE for the animated trajectories corresponding to each cost.

The different methods obtain solution trajectories in a same homotopy class. The slight differences between the trajectories, and in the values of C^*, can in part be attributed to the violation of the kinematic constraints, as only the local coordinates method can find a solution that is fully compliant with such constraints. Moreover, the solved problems are slightly different in the four methods. All methods include different penalty terms in their cost functions (either to favor $\boldsymbol{x}_N = \boldsymbol{x}_g$, or proper projections on $\mathcal{X}$). We also note that the time t_{opt} is significantly higher in the local coordinates method, but this method obtains high-quality solutions in return.

Figure 4.11 illustrates the effect of different cost functions on the solutions obtained. The figure shows the initial guess of the action trajectory (top plot) and the optimized action trajectories $\boldsymbol{u}^*(t)$ that we obtain when using the following running costs in Eq. (4.3a):

$$\begin{aligned} L_1(\boldsymbol{x}(t), \boldsymbol{u}(t)) &= \boldsymbol{u}(t)^\top \, \boldsymbol{u}(t), \\ L_2(\boldsymbol{x}(t), \boldsymbol{u}(t)) &= \dot{\boldsymbol{u}}(t)^\top \, \dot{\boldsymbol{u}}(t), \\ L_3(\boldsymbol{x}(t), \boldsymbol{u}(t)) &= 1. \end{aligned}$$

As we see, the use of L_1 results in an action trajectory $\boldsymbol{u}^*(t)$ that is smoother in comparison to the initial guess, so the control signal will be easier to follow. If further smoothing is needed, we can use L_2, which will minimize the derivative of $\boldsymbol{u}(t)$. Finally, if we need to minimize the total trajectory time, we can free t_f and Δt and use L_3 to obtain the optimized trajectory $\boldsymbol{u}^*(t)$ of the fourth plot. A bang-bang control function arises in which at least one motor works at its upper or lower torque limit, and the robot achieves the goal state in only two seconds.

5 Trajectory Tracking

The trajectories we have obtained so far are "open loop" so they need to be tracked with a controller when executed in a real robot. We next develop techniques to design such a controller, so we can achieve stable trackings in the presence of disturbances or dynamic model inaccuracies. After putting our work into context (Section 5.1), we formally define the control problem to be solved (Section 5.2) and recall classical computed-torque control methods (Section 5.3). These methods are powerful and of widespread use, but they fail perilously near forward singularities. Therefore, we can only apply them if the trajectory has been actively planned to avoid such singularities using the method in Section 3.8. The avoidance of forward singularities, however, may limit the motion capabilities of the robot, so it is natural to pursue alternative controllers able to cross these critical configurations. We show that this is indeed possible by adapting the classical LQR controller to deal with robots with closed kinematic chains (Section 5.4). We also compare the performance of the computed-torque and LQR controllers by means of illustrative examples (Section 5.5).

5.1 Related Work

As in serial manipulators, stable trajectory trackings in closed-chain robots are typically achieved using computed-torque methods [17, 77, 110–116]. These employ inverse dynamics and feedback linearization to obtain a closed-loop system with easy-to-tune control parameters. Computed-torque laws are effective because they tend to produce global basins of attraction towards the desired trajectory. In such laws, however, the inverse dynamic problem must be solved at each iteration of the control loop, which is an expensive step that hinders the design of high-frequency controllers in time-critical tasks. Still, the biggest drawback of computed-torque methods is that their control law degenerates near forward singularities, because in these configurations the inverse dynamics is almost always unsolvable (Section 2.3.3). Possible workarounds are to limit the robot motion to singularity-free regions of the C-space [77, 110, 111], or to use actuation redundancy to eliminate all or almost all forward singularities [113–116], but such solutions restrict the robot motions and complicate the structural design substantially.

It is important to note that singularity-crossing motions can safely be executed if the kinodynamic constraints of the robot are respected [74, 75, 117, 118]. Recent work on this regard uses multi-control architectures [119, 120], which in the proximity of a singularity switch from a computed-torque law to a controller based on virtual constraints. However, the switching from one controller to the other depends on a heuristic parameter that needs to be tuned experimentally, which is time-consuming and may be harmful to the robot. Alternatively, the controller we introduce in Section 5.4 does not present such limitations. As we shall see, the classical theory of linear quadratic regulators [66] can be extended to cope with closed kinematic chains, providing a single-structure controller that is easy to tune, and does not resort to feedback linearization. Since such a controller does not employ inverse dynamics, it can be used to stabilize a trajectory even across forward singularities. The control laws that result are also cheap to evaluate and produce optimal actions that minimize trajectory errors and control efforts.

Our work is inspired on recent developments in humanoid control [87, 121], but our arguments and context of application are different. Notice that in humanoid robots all joints are actuated, so forward singularities are rarely an issue in them. Instead, these singularities arise naturally in closed-chain robots, so they must be analyzed and carefully considered in the control problem.

5.2 The Trajectory Control Problem

Let $\boldsymbol{x} = \boldsymbol{x}_0(t)$ and $\boldsymbol{u} = \boldsymbol{u}_0(t)$ be the state and action trajectories to be followed by the robot, and assume these trajectories are consistent with Eqs. (2.14). Thus, for any time t, $(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x}_0(t), \boldsymbol{u}_0(t))$ satisfies

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, & (5.1a) \\ \dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}). & (5.1b) \end{cases}$$

The trajectory control problem then consists in designing a control law

$$\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{x}, t)$$

so that the closed-loop system

$$\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x}, t)) \tag{5.2}$$

is stable along $\boldsymbol{x}_0(t)$. This implies that the integral curves $\boldsymbol{x}(t)$ of Eq. (5.2) must be convergent to $\boldsymbol{x}_0(t)$ for arbitrary initial conditions in a neighborhood of $\boldsymbol{x}_0(t)$.

Note that while $\boldsymbol{x}_0(t)$ and $\boldsymbol{u}_0(t)$ could be defined in many ways, in this thesis we generate them using the methods in Chapters 3 and 4, so we will often refer to them as the planned state and action trajectories.

5.3 Computed-torque Control

An extended approach in robotics is to solve the previous problem by means of computed-torque control methods. We first explain these methods for the case of open-chain robots (Section 5.3.1) and then we adapt them to robots with closed kinematic chains (Section 5.3.2). In the latter case we will see that the resulting laws are problematic near forward singularities.

5.3.1 The Feedback Law in Open-chain Robots

Consider an open-chain robot whose coordinates $\boldsymbol{q}$ are all independent and actuated, so $n_u = n_q$. The dynamics of this system is described by an equation of the form

$$\boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{\tau}_{\text{CT}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{u}, \tag{5.3}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is a positive-definite mass matrix, $\boldsymbol{\tau}_{\text{CT}}(\boldsymbol{q},\dot{\boldsymbol{q}})$ includes all Coriolis, gravity, and friction terms, and $\boldsymbol{u} \in \mathbb{R}^{n_u}$ is the control input. To convert the system into a linear one, we can use the feedback law

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{u}' + \boldsymbol{\tau}_{\text{CT}}(\boldsymbol{q},\dot{\boldsymbol{q}}), \tag{5.4}$$

where $\boldsymbol{u}'$ is a new control variable. Certainly, by substituting Eq. (5.4) into Eq. (5.3), we get

$$\boldsymbol{M}(\boldsymbol{q})\,\boldsymbol{u}' = \boldsymbol{M}(\boldsymbol{q})\,\ddot{\boldsymbol{q}},$$

and since $\boldsymbol{M}(\boldsymbol{q})$ is full rank we have

$$\boldsymbol{u}' = \ddot{\boldsymbol{q}}, \tag{5.5}$$

which shows that the system is now equivalent to a double integrator. We then can apply linear control techniques to stabilize this system along a desired trajectory $\boldsymbol{q}_0(t)$. A common approach is to use a proportional-derivative (PD) feedback law for $\boldsymbol{u}'$, i.e.,

$$\boldsymbol{u}' = \ddot{\boldsymbol{q}}_0(t) - \boldsymbol{K}_v\,\underbrace{(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0(t))}_{\dot{\boldsymbol{e}}(t)} - \boldsymbol{K}_p\,\underbrace{(\boldsymbol{q} - \boldsymbol{q}_0(t))}_{\boldsymbol{e}(t)}, \tag{5.6}$$

where $\boldsymbol{K}_v$ and $\boldsymbol{K}_p$ are positive-definite gain matrices and $\boldsymbol{e}(t)$ is the configuration error. By substituting Eq. (5.6) into Eq. (5.5), we obtain the linear differential equation

$$\ddot{\boldsymbol{e}}(t) + \boldsymbol{K}_v\, \dot{\boldsymbol{e}}(t) + \boldsymbol{K}_p\, \boldsymbol{e}(t) = \boldsymbol{0},$$

for which it can be shown that

$$\lim_{t\to\infty} \boldsymbol{e}(t) = \boldsymbol{0}$$

if $\boldsymbol{K}_v \succ \boldsymbol{0}$ and $\boldsymbol{K}_p \succ \boldsymbol{0}$, irrespective of the initial condition $\boldsymbol{e}(0)$ [24]. By substituting Eq. (5.6) into Eq. (5.4) we obtain the usual computed-torque control law:

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \boldsymbol{M}(\boldsymbol{q})\ \underbrace{(\ddot{\boldsymbol{q}}_0(t) - \boldsymbol{K}_v\dot{\boldsymbol{e}}(t) - \boldsymbol{K}_p\boldsymbol{e}(t))}_{\boldsymbol{u}'} + \boldsymbol{\tau}_{\text{CT}}(\boldsymbol{q}, \dot{\boldsymbol{q}}).$$

A strong point of this law is it ensures global stability, so the robot will be able to converge to $\boldsymbol{q}_0(t)$ independently of its initial state.

5.3.2 The Feedback Law in Closed-chain Robots

The previous method requires several modifications to be applicable to closed kinematic chains. Recall that the dynamics of a closed-chain robot evolves according

$$\boldsymbol{M}(\boldsymbol{q})\ \ddot{\boldsymbol{q}} + \boldsymbol{\tau}_{\text{CT}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \boldsymbol{\lambda} = \boldsymbol{Q}_u\, \boldsymbol{u}, \tag{5.7}$$

which now includes the constraint force $\boldsymbol{\Phi}_{\boldsymbol{q}}(\boldsymbol{q})^\top \boldsymbol{\lambda}$ and the term $\boldsymbol{Q}_u\, \boldsymbol{u}$, where $\boldsymbol{Q}_u \in \mathbb{R}^{n_q \times n_u}$ is the matrix we defined in Section 2.2. Also recall that the $\boldsymbol{q}$ coordinates are not independent now. However, since $\mathcal{C}$ is a smooth manifold, the implicit function theorem guarantees that, around any point $\boldsymbol{q}_c \in \mathcal{C}$ traversed by $\boldsymbol{q}_0(t)$, there exists a regular parametrization

$$\boldsymbol{q} = \boldsymbol{m}(\boldsymbol{\theta}), \tag{5.8}$$

where $\boldsymbol{\theta}$ is a vector of $d_{\mathcal{C}}$ independent coordinates. In particular, the first and second time derivatives of Eq. (5.8) will be of the form

$$\begin{aligned}\dot{\boldsymbol{q}} =& \boldsymbol{\Lambda}\dot{\boldsymbol{\theta}},\\ \ddot{\boldsymbol{q}} =& \boldsymbol{\Lambda}\ddot{\boldsymbol{\theta}} + \dot{\boldsymbol{\Lambda}}\dot{\boldsymbol{\theta}},\end{aligned} \tag{5.9}$$

where $\boldsymbol{\Lambda} = \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{\theta}}(\boldsymbol{q})$ is an $n_q \times d_{\mathcal{C}}$ full-rank matrix whose columns provide a basis of $\mathcal{T}_{\boldsymbol{q}_c}\mathcal{C}$, and $\dot{\boldsymbol{\Lambda}} = \frac{\mathrm{d}\boldsymbol{\Lambda}}{\mathrm{d}t}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, (see Section 5.3.3 for details on how $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$ can be computed).

Now, since the columns of $\boldsymbol{\Lambda}$ are vectors of $\mathcal{T}_{\boldsymbol{q}_c}\mathcal{C}$, the multiplication of Eq. (5.7) by $\boldsymbol{\Lambda}^\top$ must make $\boldsymbol{\Lambda}^\top\ \boldsymbol{\Phi}_{\boldsymbol{q}}^\top\ \boldsymbol{\lambda}$ vanish as we saw in Section 2.3.3. After such a multiplication we thus obtain

$$\boldsymbol{\Lambda}^\top \boldsymbol{M}\, \ddot{\boldsymbol{q}} + \boldsymbol{\Lambda}^\top\ \boldsymbol{\tau}_{\text{CT}} = \boldsymbol{\Lambda}^\top \boldsymbol{Q}_u\, \boldsymbol{u}, \tag{5.10}$$

where we omitted the matrix dependencies on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ for simplicity. By substituting Eq. (5.9) into Eq. (5.10) we obtain

$$\underbrace{\boldsymbol{\Lambda}^\top \boldsymbol{M}\boldsymbol{\Lambda}}_{\bar{\boldsymbol{M}}}\ \ddot{\boldsymbol{\theta}} + \underbrace{\boldsymbol{\Lambda}^\top \boldsymbol{M}\dot{\boldsymbol{\Lambda}}\ \dot{\boldsymbol{\theta}} + \boldsymbol{\Lambda}^\top\ \boldsymbol{\tau}_{\text{CT}}}_{\bar{\boldsymbol{\tau}}_{\text{CT}}} = \underbrace{\boldsymbol{\Lambda}^\top \boldsymbol{Q}_u}_{\boldsymbol{N}}\ \boldsymbol{u},$$

which can be compactly written as

$$\bar{\boldsymbol{M}}\ \boldsymbol{u}' + \bar{\boldsymbol{\tau}}_{\text{CT}} = \boldsymbol{N}\, \boldsymbol{u}, \tag{5.11}$$

where $\bar{\boldsymbol{M}} \in d_{\mathcal{C}} \times d_{\mathcal{C}}$, $\bar{\boldsymbol{\tau}}_{\text{CT}} \in \mathbb{R}^{d_{\mathcal{C}}}$, and $\boldsymbol{N} \in \mathbb{R}^{d_{\mathcal{C}} \times n_u}$. At this point it is easy to feedback linearize the system in Eq. (5.11) by using the control law

$$\boldsymbol{u} = \boldsymbol{N}^+ \left(\bar{\boldsymbol{M}}\ \boldsymbol{u}' + \bar{\boldsymbol{\tau}}_{\text{CT}}\right), \tag{5.12}$$

where $\boldsymbol{N}^+$ is the $n_u \times d_{\mathcal{C}}$ right pseudoinverse of $\boldsymbol{N}$. Recall that $\boldsymbol{N}^+ = \boldsymbol{N}^\top(\boldsymbol{N}\boldsymbol{N}^\top)^{-1}$ and $\boldsymbol{N}\boldsymbol{N}^+ = \boldsymbol{I}$, but $\boldsymbol{N}^+$ is undefined when $\boldsymbol{N}$ is not full row rank, which occurs at forward singularities (Section 2.2.3). By substituting Eq. (5.12) into Eq. (5.11) we then obtain

$$\bar{\boldsymbol{M}}\ \ddot{\boldsymbol{\theta}} + \bar{\boldsymbol{\tau}}_{\text{CT}} = \boldsymbol{N}\boldsymbol{N}^+ \left(\bar{\boldsymbol{M}}\ \boldsymbol{u}' + \bar{\boldsymbol{\tau}}_{\text{CT}}\right),$$

which simplifies into

$$\bar{\boldsymbol{M}}\ \ddot{\boldsymbol{\theta}} = \bar{\boldsymbol{M}}\ \boldsymbol{u}'. \tag{5.13}$$

Now, since $\boldsymbol{\Lambda}$ is full rank, $\bar{\boldsymbol{M}} = \boldsymbol{\Lambda}^\top \boldsymbol{M}\boldsymbol{\Lambda}$ must be full rank too, so Eq. (5.13) implies that

$$\ddot{\boldsymbol{\theta}} = \boldsymbol{u}'. \tag{5.14}$$

Then, similarly to what was done in Section 5.3.1, we can globally stabilize the system in Eq. (5.14) by using

$$\boldsymbol{u}' = \ddot{\boldsymbol{\theta}}_0(t) - \boldsymbol{K}_v \underbrace{\left(\dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_0(t)\right)}_{\dot{\boldsymbol{e}}(t)} - \boldsymbol{K}_p \underbrace{\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0(t)\right)}_{\boldsymbol{e}(t)}, \tag{5.15}$$

where $\boldsymbol{K}_v$ and $\boldsymbol{K}_p$ are positive-definite gain matrices of size $d_{\mathcal{C}} \times d_{\mathcal{C}}$.

By replacing $\boldsymbol{u}'$ in Eq. (5.12) by its value in Eq. (5.15), we finally obtain the desired computed-torque control law for closed-chain robotic systems:

$$\boldsymbol{u} = \boldsymbol{N}^{+}\left(\bar{\boldsymbol{M}}\ \underbrace{\left(\ddot{\boldsymbol{\theta}}_0(t) - \boldsymbol{K}_v(\dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_0(t)) - \boldsymbol{K}_p(\boldsymbol{\theta} - \boldsymbol{\theta}_0(t))\right)}_{\boldsymbol{u}'} + \bar{\boldsymbol{\tau}}_{\text{CT}}\right). \tag{5.16}$$

Note that we can also write this law in the following form

$$\boldsymbol{u} = \underbrace{\boldsymbol{N}^{+}\left(\bar{\boldsymbol{M}}\ \ddot{\boldsymbol{\theta}}_0(t) + \bar{\boldsymbol{\tau}}_{\text{CT}}\right)}_{\boldsymbol{u}_0(t)} - \underbrace{\boldsymbol{N}^{+}\bar{\boldsymbol{M}}\left(\boldsymbol{K}_v(\dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_0(t)) + \boldsymbol{K}_p(\boldsymbol{\theta} - \boldsymbol{\theta}_0(t))\right)}_{\boldsymbol{u}_{\text{corr}}(t)},$$

where $\boldsymbol{u}_0(t)$ is the nominal action needed to produce the desired trajectory $\boldsymbol{q}_0(t)$, and $\boldsymbol{u}_{\text{corr}}(t)$ is the corrective action used to compensate deviations from $\boldsymbol{q}_0(t)$ and $\dot{\boldsymbol{q}}_0(t)$.

5.3.3 Computing $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$

The control law in Eq. (5.16) requires choosing a set of d_C independent coordinates $\boldsymbol{\theta}$ to locally parameterize $\boldsymbol{q}$ along the trajectory $\boldsymbol{q}_0(t)$, and then compute $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$. While $\boldsymbol{\theta}$ could be defined using tangent space coordinates, we here follow the common approach of coordinate partitioning [17]. To this end, note that at each point $\boldsymbol{q}$ of the trajectory $\boldsymbol{q}_0(t)$ we can choose a subset of independent and dependent coordinates from $\boldsymbol{q} \in \mathbb{R}^{n_q}$, $\boldsymbol{q}_i \in \mathbb{R}^{d_C}$ and $\boldsymbol{q}_d \in \mathbb{R}^{n_q - d_C}$ respectively, such that

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{q}_i \\ \boldsymbol{q}_d \end{bmatrix},$$

where $\begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix}$ is a permutation matrix. Under such a choice, we have $\boldsymbol{\theta} = \boldsymbol{q}_i$ therefore. Then, the velocity constraint in Eq. (2.2) can be written as

$$\boldsymbol{\Phi}_{\boldsymbol{q}_i}\ \dot{\boldsymbol{q}}_i + \boldsymbol{\Phi}_{\boldsymbol{q}_d}\ \dot{\boldsymbol{q}}_d = \boldsymbol{0}. \tag{5.17}$$

We can now express $\dot{\boldsymbol{q}}_d$ in terms of $\dot{\boldsymbol{q}}_i$ as

$$\dot{\boldsymbol{q}}_d = -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1}\boldsymbol{\Phi}_{\boldsymbol{q}_i}\ \dot{\boldsymbol{q}}_i, \tag{5.18}$$

so

$$\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}_i \\ \dot{\boldsymbol{q}}_d \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_{d_C} \\ -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1}\boldsymbol{\Phi}_{\boldsymbol{q}_i} \end{bmatrix}}_{\boldsymbol{\Lambda}} \dot{\boldsymbol{q}}_i. \tag{5.19}$$

From Eq. (5.19), we directly see that

$$\boldsymbol{\Lambda} = \boldsymbol{Q}_i - \boldsymbol{Q}_d \boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i}. \tag{5.20}$$

On the other hand, if we differentiate Eq. (5.17), we obtain

$$\boldsymbol{\Phi}_{\boldsymbol{q}_i}\, \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_i}\, \dot{\boldsymbol{q}}_i + \boldsymbol{\Phi}_{\boldsymbol{q}_d}\, \ddot{\boldsymbol{q}}_d + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_d}\, \dot{\boldsymbol{q}}_d = \boldsymbol{0},$$

from where we see that

$$\ddot{\boldsymbol{q}}_d = -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \left(\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_d}\, \dot{\boldsymbol{q}}_d + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_i}\, \dot{\boldsymbol{q}}_i + \boldsymbol{\Phi}_{\boldsymbol{q}_i}\, \ddot{\boldsymbol{q}}_i \right),$$

which, if we use Eq. (5.18), can be written as

$$\ddot{\boldsymbol{q}}_d = -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \left(-\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_d}\, \boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_i} \right) \dot{\boldsymbol{q}}_i - \left(\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i} \right) \ddot{\boldsymbol{q}}_i,$$

This equation allows us to express $\ddot{\boldsymbol{q}}$ in terms of $\ddot{\boldsymbol{q}}_d$ and $\dot{\boldsymbol{q}}_d$ as follows

$$\ddot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}}_i \\ \ddot{\boldsymbol{q}}_d \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{0} \\ -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \left(-\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_d}\, \boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_i} \right) \end{bmatrix}}_{\dot{\boldsymbol{\Lambda}}} \dot{\boldsymbol{q}}_i + \underbrace{\begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{Q}_d \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_{d_C} \\ -\boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i} \end{bmatrix}}_{\boldsymbol{\Lambda}} \ddot{\boldsymbol{q}}_i,$$

from where we finally note that

$$\dot{\boldsymbol{\Lambda}} = -\boldsymbol{Q}_d \boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \left(\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_i} - \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}_d} \boldsymbol{\Phi}_{\boldsymbol{q}_d}^{-1} \boldsymbol{\Phi}_{\boldsymbol{q}_i} \right). \tag{5.21}$$

Therefore, Eqs. (5.20) and (5.21) provide the desired expressions for $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$.

5.3.4 Degeneracy of the Feedback Law Near Singularities

In the previous section we showed how to compute $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$, which are necessary to obtain the feedback law in Eq. (5.16). However, by simple inspection of Eqs. (5.20) and (5.21), one can see that these matrices are not well-defined when $\boldsymbol{\Phi}_{\boldsymbol{q}_d}$ is rank deficient. Despite so, the fact that $\boldsymbol{\Phi}_{\boldsymbol{q}}$ is always full rank guarantees that at every time instant we can choose a different partition into $\boldsymbol{q}_d$ and $\boldsymbol{q}_i$ coordinates that does not result in a singular $\boldsymbol{\Phi}_{\boldsymbol{q}_d}$ matrix. While this approach seems to solve the degeneracy of the feedback law in Eq. (5.16), note that this law still requires computing the pseudoinverse of $\boldsymbol{N}$, which is undefined at forward singularities because $\operatorname{rank}(\boldsymbol{N}) < d_{\mathcal{C}}$ in them (Section 2.2.3). Therefore, regardless of the $\boldsymbol{q}_i$ coordinates chosen to compute $\boldsymbol{\Lambda}$ and $\dot{\boldsymbol{\Lambda}}$, the computed-torque control law in Eq. (5.16) will inevitably fail at forward singularities. We next develop an alternative to the controllers in Section 5.3 that addresses the trajectory control problem while being robust to forward singularities.

5.4 Linear Quadratic Regulators

Consider a trajectory controller that minimizes the additive cost function

$$J = \int_0^{t_f} \Big(\bar{\boldsymbol{x}}(t)^\top \boldsymbol{Q}\, \bar{\boldsymbol{x}}(t) + \bar{\boldsymbol{u}}(t)^\top \boldsymbol{R}\, \bar{\boldsymbol{u}}(t)\Big) \mathrm{d}t, \tag{5.22}$$

where $\bar{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0(t)$, $\bar{\boldsymbol{u}} = \boldsymbol{u} - \boldsymbol{u}_0(t)$, $\boldsymbol{Q}$ is a positive semi-definite matrix penalizing deviations from the planned trajectory $\boldsymbol{x}_0(t)$, and $\boldsymbol{R}$ is a positive-definite matrix penalizing deviations from the planned actions $\boldsymbol{u}_0(t)$. Unfortunately, for general nonlinear systems $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$, there is no known expression for such a controller. However, for linear time-varying systems with the form

$$\dot{\bar{\boldsymbol{x}}} = \boldsymbol{A}(t)\bar{\boldsymbol{x}} + \boldsymbol{B}(t)\bar{\boldsymbol{u}}, \tag{5.23}$$

where the $\boldsymbol{x}$ coordinates are independent, the controller is given by the linear quadratic regulator (LQR) [66]. The result is the optimal controller

$$\bar{\boldsymbol{u}} = \boldsymbol{\pi}(\boldsymbol{x}, t) = -\boldsymbol{K}(t)\, \bar{\boldsymbol{x}}, \tag{5.24}$$

or, equivalently,

$$\boldsymbol{u} = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\, (\boldsymbol{x} - \boldsymbol{x}_0(t)),$$

where $\boldsymbol{K}(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{S}(t)$, and $\boldsymbol{S}(t)$ is the solution to the differential Riccati equation

$$-\dot{\boldsymbol{S}} = \boldsymbol{S}\boldsymbol{A}(t) + \boldsymbol{A}(t)^\top \boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}(t)\boldsymbol{R}^{-1}\boldsymbol{B}(t)^\top \boldsymbol{S} + \boldsymbol{Q}. \tag{5.25}$$

Note that $\boldsymbol{S}(t)$ can be obtained by integrating Eq. (5.25) backwards in time using the terminal condition $\boldsymbol{S}(t_f) = \boldsymbol{Q}$ [66].

Despite being restricted to the linear case, the previous controller is powerful, as one can apply it to the time-varying linearization of $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ along $\big(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)\big)$, which takes the form of Eq. (5.23). Such an approach has provided effective tracking controllers for nonlinear systems with independent $\boldsymbol{x}$ coordinates [66], but note it cannot directly be applied to closed-chain robots because in them $\boldsymbol{x}$ is subject to Eq. (5.1a). In fact, if in such robots we obtain Eq. (5.23) by a direct linearization of Eq. (5.1b) along $\big(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)\big)$, we will find them to be not controllable in the usual sense, as this linearization ignores Eq. (5.1a) [87]. One might argue that the differential equations could always be expressed in terms of independent task- or joint-space coordinates to apply the law in Eq. (5.24), but then we would be faced with the singularities introduced by such coordinates as explained in Section 5.3.4. While similar, the route we next take is not affected by such singularities, as we use tangent-space coordinates that are always well-defined around any $\boldsymbol{x} \in \mathcal{X}$. We first explain our approach for the case in which $\boldsymbol{x}_0(t)$ is an equilibrium point, and then provide the controller for a general trajectory $\boldsymbol{x}_0(t)$.

5.4.1 Stabilization at an Equilibrium Point

Suppose $\boldsymbol{x}_0(t) = \boldsymbol{x}_0 \ \forall t$, where $\boldsymbol{x}_0$ is an equilibrium point of $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ for $\boldsymbol{u} = \boldsymbol{u}_0$, i.e., $\boldsymbol{g}(\boldsymbol{x}_0, \boldsymbol{u}_0) = \boldsymbol{0}$. To stabilize the robot at $\boldsymbol{x}_0$, we first rewrite $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ using the independent $\boldsymbol{y}$ coordinates of $\mathcal{T}_{\boldsymbol{x}_0}\mathcal{X}$, and then apply the controller in Eq. (5.24) using such coordinates.

Recall that the map $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{x})$ that provides the $\boldsymbol{y}$ coordinates of $\mathcal{T}_{\boldsymbol{x}_0}\mathcal{X}$ is given by

$$\boldsymbol{y} = \boldsymbol{U}^{\top} \underbrace{(\boldsymbol{x} - \boldsymbol{x}_0)}_{\bar{\boldsymbol{x}}}, \tag{5.26}$$

where $\boldsymbol{U}$ is an $n_x \times d_{\mathcal{X}}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_0}\mathcal{X}$. Also recall that the inverse map $\boldsymbol{x} = \boldsymbol{\psi}(\boldsymbol{y})$ is given by the solution of

$$\begin{cases} \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}, \\ \boldsymbol{U}^{\top} \underbrace{(\boldsymbol{x} - \boldsymbol{x}_0)}_{\bar{\boldsymbol{x}}} - \boldsymbol{y} = \boldsymbol{0}. \end{cases}$$

As in earlier chapters, to obtain $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ in $\boldsymbol{y}$ coordinates we take the time derivative of Eq. (5.26),

$$\dot{\boldsymbol{y}} = \boldsymbol{U}^{\top} \dot{\boldsymbol{x}},$$

and apply the substitutions $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ and $\boldsymbol{x} = \boldsymbol{\psi}(\boldsymbol{y})$ to get

$$\dot{\boldsymbol{y}} = \boldsymbol{U}^{\top} \boldsymbol{g}(\boldsymbol{\psi}(\boldsymbol{y}), \boldsymbol{u}),$$

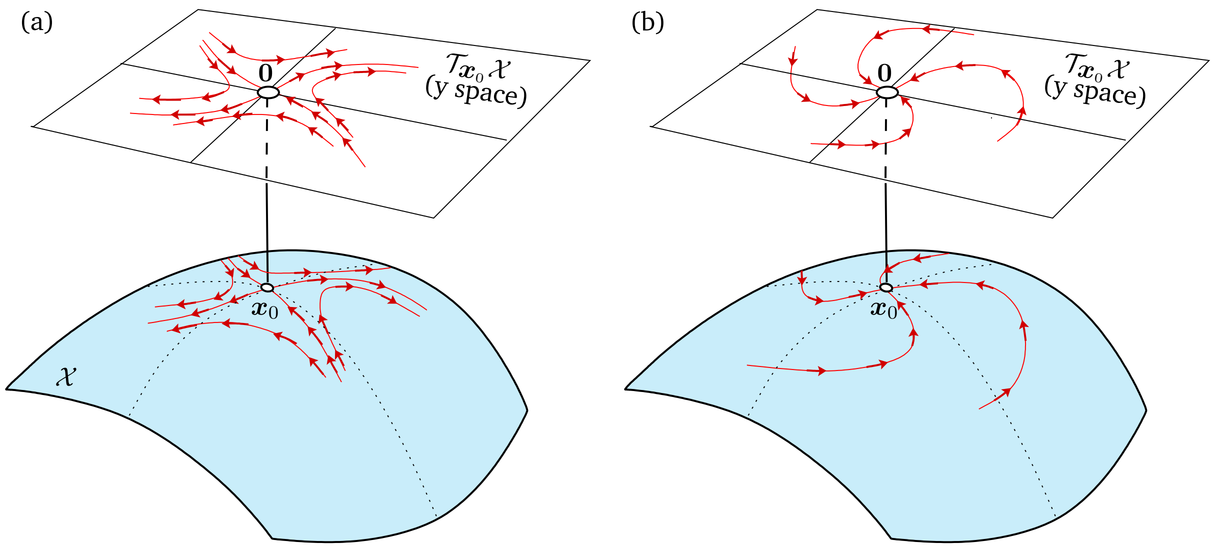


Figure 5.1: Stabilization at an equilibrium point $\boldsymbol{x}_0$ for $\boldsymbol{u} = \boldsymbol{u}_0$. (a) The vector field $\dot{\boldsymbol{x}} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ around $\boldsymbol{x}_0$ is mapped into a vector field $\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u})$ around the origin $\boldsymbol{y} = \mathbf{0}$ of $\mathcal{T}_{\boldsymbol{x}_0}\mathcal{X}$. This vector field is action varying in fact, but we draw it for $\boldsymbol{u} = \boldsymbol{u}_0$. (b) Using LQR techniques, we can design a feedback law to stabilize the system in $\boldsymbol{y}$ coordinates at $\boldsymbol{y} = \mathbf{0}$. This law makes the integral curves of the closed-loop system locally convergent to $\boldsymbol{y} = \mathbf{0}$, so the original system will converge to $\boldsymbol{x} = \boldsymbol{x}_0$.

which we compactly write as

$$\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u}). \tag{5.28}$$

Note now that, since $\boldsymbol{x}_0$ gets mapped into $\boldsymbol{y} = \mathbf{0}$ by Eq. (5.26), our goal has turned into stabilizing the system in Eq. (5.28) at the origin $\boldsymbol{y} = \mathbf{0}$ of $\mathbb{R}^{d_\mathcal{X}}$ (Fig. 5.1). Since we wish asymptotic convergence to $\boldsymbol{y} = \mathbf{0}$, we are in the context of infinite-horizon optimal control, so we set t_f to infinity in Eq. (5.22). Moreover, by using the fact that $\boldsymbol{y} = \boldsymbol{U}^\top \bar{\boldsymbol{x}}$, Eq. (5.22) can be written in terms of the $\boldsymbol{y}$ coordinates as follows

$$J = \int_0^\infty \left(\boldsymbol{y}^\top \tilde{\boldsymbol{Q}}\, \boldsymbol{y} + \bar{\boldsymbol{u}}^\top \boldsymbol{R}\, \bar{\boldsymbol{u}}\right) \mathrm{d}t, \tag{5.29}$$

where $\tilde{\boldsymbol{Q}} = \boldsymbol{U}^\top \boldsymbol{Q} \boldsymbol{U}$.

To obtain an LQR controller, we linearize Eq. (5.28) around $\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{u} = \boldsymbol{u}_0$. The linearized system has a form similar to the one we derived in Section 3.5.2 for the LQR steering method, but now the Taylor expansion of Eq. (5.28) is

$$\dot{\boldsymbol{y}} \approx \tilde{\boldsymbol{g}}(\boldsymbol{0}, \boldsymbol{u}_0) + \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{y}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{u}_0}}}_{\boldsymbol{A}} (\boldsymbol{y} - \boldsymbol{0}) + \underbrace{\left.\frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{u}}\right|_{\substack{\boldsymbol{y}=\boldsymbol{0}\\ \boldsymbol{u}=\boldsymbol{u}_0}}}_{\boldsymbol{B}} \underbrace{(\boldsymbol{u} - \boldsymbol{u}_0)}_{\bar{\boldsymbol{u}}},$$

where $\tilde{\boldsymbol{g}}(\boldsymbol{0}, \boldsymbol{u}_0) = \boldsymbol{0}$ because $\boldsymbol{y} = \boldsymbol{0}$ is an equilibrium point for $\boldsymbol{u} = \boldsymbol{u}_0$. In $\boldsymbol{y}$ coordinates, thus, the linearized system takes the form

$$\dot{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\bar{\boldsymbol{u}}. \tag{5.30}$$

As in Section 3.5.2, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices can be computed as

$$\boldsymbol{A} = \boldsymbol{U}^\top \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \boldsymbol{U},$$
$$\boldsymbol{B} = \boldsymbol{U}^\top \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}},$$

the only difference being that $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}$ and $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}$ must now be evaluated at $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$.

From the results presented in Section 5.4, the optimal control that minimizes the cost in Eq. (5.29) for the system in Eq. (5.30) is given by

$$\boldsymbol{u} = \boldsymbol{u}_0 - \boldsymbol{K}\,\boldsymbol{y},$$

which, using Eq. (5.26), can be written in terms of the $\boldsymbol{x}$ variables as

$$\boldsymbol{u} = \boldsymbol{u}_0 - \boldsymbol{K}\boldsymbol{U}^\top (\boldsymbol{x} - \boldsymbol{x}_0).$$

Since we are solving an infinite-horizon control problem, the solution of Eq. (5.25) boils down to solving for $\boldsymbol{S}$ the algebraic Riccati equation [66]

$$\boldsymbol{0} = \boldsymbol{S}\boldsymbol{A} + \boldsymbol{A}^\top \boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{S} + \tilde{\boldsymbol{Q}},$$

which can be done using generalized eigenvalue methods [122].

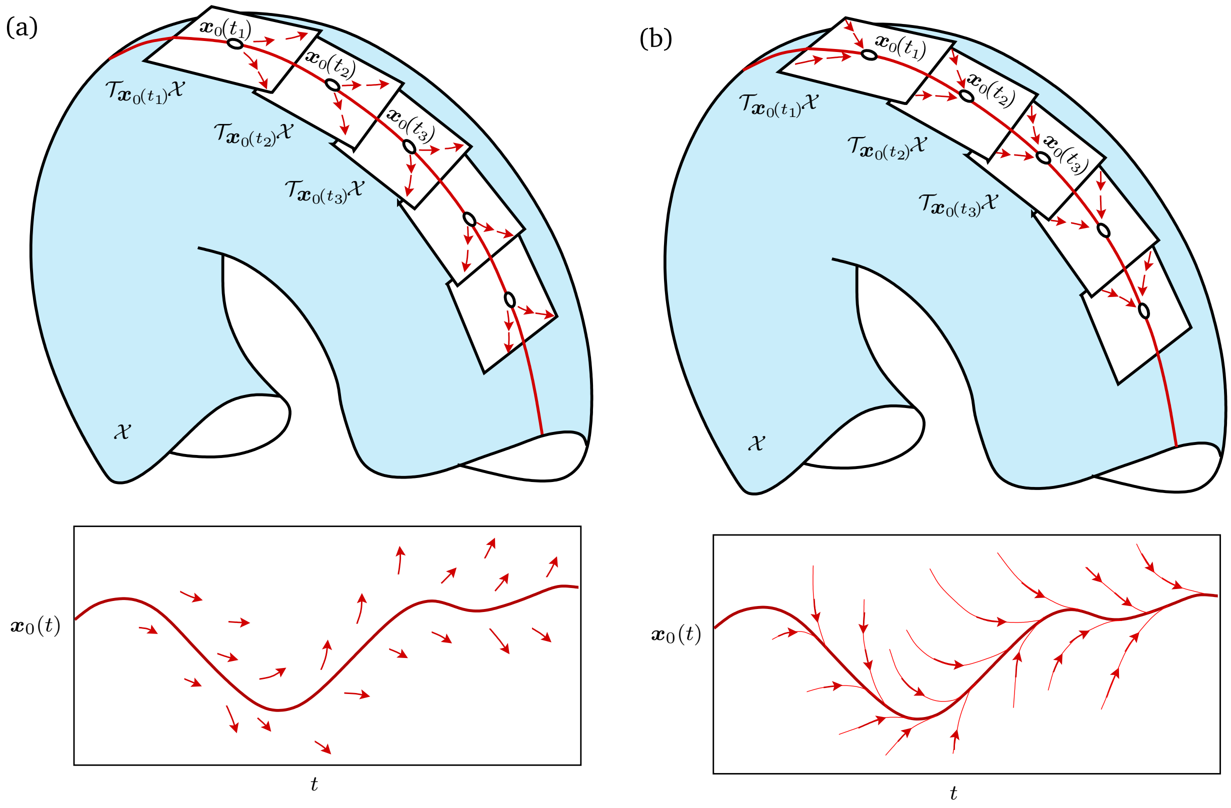

(a)
$x_0(t_1)$
$x_0(t_2)$
$x_0(t_3)$
$\mathcal{T}_{x_0(t_1)}\mathcal{X}$
$\mathcal{T}_{x_0(t_2)}\mathcal{X}$
$\mathcal{T}_{x_0(t_3)}\mathcal{X}$
$\mathcal{X}$
$x_0(t)$
t
(b)
$x_0(t_1)$
$x_0(t_2)$
$x_0(t_3)$
$\mathcal{T}_{x_0(t_1)}\mathcal{X}$
$\mathcal{T}_{x_0(t_2)}\mathcal{X}$
$\mathcal{T}_{x_0(t_3)}\mathcal{X}$
$\mathcal{X}$
$x_0(t)$
t

Figure 5.2: Stabilization of a given trajectory $\boldsymbol{x}_0(t)$. (a) Without a feedback law, the vector field $\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u}, t)$ may be divergent from $\boldsymbol{y} = \boldsymbol{0}$ at each time $t \in [0, t_f]$ for $\boldsymbol{u} = \boldsymbol{u}_0(t)$. Thus, the slightest perturbation may move the robot away from $\boldsymbol{x}_0(t)$. (b) Our control problem boils down to designing a feedback law $\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{y}, t)$ such that the integral curves of the closed loop system $\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{\pi}(\boldsymbol{y}, t), t)$ are all convergent to the desired trajectory.

5.4.2 Trajectory Stabilization

Now consider the general case of stabilizing an arbitrary trajectory $\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)$. The problem is similar to the one in the earlier section, but now $\boldsymbol{x}_0$ and $\boldsymbol{u}_0$ are time-varying, so the tangent space basis $\boldsymbol{U}$ and the mappings $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ will also be time-varying. In what follows, thus, we will write $\boldsymbol{U}(t)$, $\boldsymbol{\varphi}(\boldsymbol{x}, t)$, and $\boldsymbol{\psi}(\boldsymbol{y}, t)$ respectively. In particular, the map $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{x}, t)$ will be given by

$$\boldsymbol{y} = \boldsymbol{U}(t)^\top \, (\boldsymbol{x} - \boldsymbol{x}_0(t)), \tag{5.32}$$

so its time derivative will now be

$$\dot{\boldsymbol{y}} = \dot{\boldsymbol{U}}(t)^\top \big(\boldsymbol{x} - \boldsymbol{x}_0(t)\big) + \boldsymbol{U}(t)^\top \big(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_0(t)\big). \tag{5.33}$$

By applying the usual substitutions we get

$$\dot{\boldsymbol{y}} = \dot{\boldsymbol{U}}(t)^\top \big(\boldsymbol{\psi}(\boldsymbol{y}, t) - \boldsymbol{x}_0(t)\big) + \boldsymbol{U}(t)^\top \big(\boldsymbol{g}(\boldsymbol{\psi}(\boldsymbol{y}, t), \boldsymbol{u}) - \boldsymbol{g}(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t))\big),$$

so the system in $\boldsymbol{y}$ coordinates takes the form

$$\dot{\boldsymbol{y}} = \tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u}, t). \tag{5.34}$$

Note that, since Eq. (5.32) maps $\boldsymbol{x} = \boldsymbol{x}_0(t)$ to $\boldsymbol{y} = \boldsymbol{0}$ for all t, our control problem boils down to stabilizing the system in Eq. (5.34) at $\boldsymbol{y} = \boldsymbol{0}$ for $t \in [0, t_f]$ (Fig. 5.2). Note also that, since t_f is fixed now (it is the total trajectory time) we have to design a finite-horizon LQR controller. In $\boldsymbol{y}$ coordinates, the trajectory cost in Eq. (5.22) will be

$$J = \int_0^{t_f} \Big(\boldsymbol{y}(t)^\top \tilde{\boldsymbol{Q}}(t)\, \boldsymbol{y}(t) + \bar{\boldsymbol{u}}(t)^\top \boldsymbol{R}\, \bar{\boldsymbol{u}}(t)\Big) \mathrm{d}t, \tag{5.35}$$

where $\tilde{\boldsymbol{Q}}(t) = \boldsymbol{U}(t)^\top \boldsymbol{Q}\, \boldsymbol{U}(t)$, and the linearization of Eq. (5.33) about $\big(\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)\big)$ will result in

$$\dot{\boldsymbol{y}} \approx \boldsymbol{A}(t)\, \boldsymbol{y} + \boldsymbol{B}(t)\, \bar{\boldsymbol{u}}, \tag{5.36}$$

where

$$\boldsymbol{A}(t) = \frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{y}} = \dot{\boldsymbol{U}}(t)^\top\, \boldsymbol{U}(t) + \boldsymbol{U}(t)^\top\, \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\, \boldsymbol{U}(t), \tag{5.37a}$$

$$\boldsymbol{B}(t) = \frac{\partial \tilde{\boldsymbol{g}}}{\partial \boldsymbol{u}} = \boldsymbol{U}(t)^\top\, \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}}.$$

From the results in Section 5.4, the optimal control that minimizes the cost in Eq. (5.35) for the system in Eq. (5.36) is then given by

$$\bar{\boldsymbol{u}}(t) = -\boldsymbol{K}(t)\, \boldsymbol{y},$$

or, equivalently,

$$\boldsymbol{u}(t) = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\, \boldsymbol{y},$$

which in terms of the $\boldsymbol{x}$ variables results in the control law

$$\boldsymbol{u} = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\, \boldsymbol{U}(t)^\top \big(\boldsymbol{x} - \boldsymbol{x}_0(t)\big).$$

A final point must be noted regarding the computation of $\boldsymbol{U}(t)$ and $\dot{\boldsymbol{U}}(t)$ needed to obtain $\boldsymbol{A}(t)$ in Eq. (5.37a). Recall from Section 4.6.3 that the tangent space basis $\boldsymbol{U}(t)$ could be computed from a QR decomposition of $\boldsymbol{F}_{\boldsymbol{x}}(t)$. However, this approach does not guarantee that $\boldsymbol{U}(t)$ is differentiable, which is needed to obtain $\dot{\boldsymbol{U}}(t)$ using, for instance, automatic differentiation. Thus, similarly to Section 4.6.3, we use the Gram-Schmidt orthonormalization to compute a differentiable basis $\boldsymbol{U}(t)$. To this end, we first compute $\boldsymbol{U}(0)$ from a QR decomposition of the Jacobian $\boldsymbol{F}_{\boldsymbol{x}}(0)$ at time 0. Second, if we consider two consecutive time instants t_{k-1} and t_k, we obtain $\boldsymbol{U}(t_k)$ by applying the Gram-Schmidt orthonormalization to $\boldsymbol{U}(t_{k-1})$ with respect to $\boldsymbol{F}_{\boldsymbol{x}}(t_k)$, as explained in Section 4.5.2. Since two consecutive states $\boldsymbol{x}(t_{k-1})$ and $\boldsymbol{x}(t_k)$ are always close to each other, this process is well-defined and yields continuous smooth values for $\boldsymbol{U}(t)$. The time derivative of these values can thus be computed using finite differences in the end.

5.4.3 Integral Action

In principle, the LQR controllers presented in Sections 5.4.1 and 5.4.2 would be sufficient to stabilize an equilibrium point or a trajectory, but this is only true if the model in Eq. (5.1b) is accurate. In practice, there will be uncertainties in the system parameters that will result in steady-state errors. To eliminate these errors, an integral action can be incorporated as follows to the LQR formulation. We first define the extended state $\boldsymbol{y}_e = (\boldsymbol{y}, \boldsymbol{y}_i)$, where

$$\boldsymbol{y}_i = \int_0^{t_f} \boldsymbol{C} \underbrace{(\boldsymbol{x} - \boldsymbol{x}_0(t))}_{\bar{\boldsymbol{x}}_0(t)} \, \mathrm{d}t \tag{5.38}$$

and $\boldsymbol{C}$ is an $d_{\mathcal{C}} \times n_x$ matrix selecting the $d_{\mathcal{C}}$ desired components of the error signal $\bar{\boldsymbol{x}}(t)$. Then, we note that Eq. (5.38) can be written in first-order form as

$$\dot{\boldsymbol{y}}_i = \boldsymbol{C}\,(\boldsymbol{x} - \boldsymbol{x}_0(t)) = \boldsymbol{C}\,(\boldsymbol{\psi}(\boldsymbol{y}, t) - \boldsymbol{x}_0(t))\,,$$

which can be linearized at $\boldsymbol{y} = \boldsymbol{0}$ to obtain

$$\dot{\boldsymbol{y}}_i \approx \boldsymbol{C}\,\frac{\partial \boldsymbol{\psi}(\boldsymbol{y}, t)}{\partial \boldsymbol{y}}\,\boldsymbol{y}, \tag{5.39}$$

where, as explained in Section 3.5.2,

$$\frac{\partial \boldsymbol{\psi}(\boldsymbol{y}, t)}{\partial \boldsymbol{y}} = \boldsymbol{U}(t).$$

Third, we can combine Eqs. (5.36) and (5.39) to write the extended linearized system

$$\dot{\boldsymbol{y}}_e = \underbrace{\begin{bmatrix} \boldsymbol{A}(t) & \boldsymbol{0} \\ \boldsymbol{C}\,\boldsymbol{U}(t) & \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{A}_e(t)} \underbrace{\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_i \end{bmatrix}}_{\boldsymbol{y}_e} + \underbrace{\begin{bmatrix} \boldsymbol{B}(t) \\ \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{B}_e(t)} \bar{\boldsymbol{u}}, \tag{5.40}$$

where $\boldsymbol{A}_e$ and $\boldsymbol{B}_e$ are referred to as the extended linearized matrices. Finally, we find the optimal control policy that minimizes the following extended cost for the system in Eq. (5.40)

$$J_e = \int_0^{t_f} \Big(\boldsymbol{y}(t)^\top \tilde{\boldsymbol{Q}}(t)\,\boldsymbol{y}(t) + \boldsymbol{y}_i(t)^\top \tilde{\boldsymbol{Q}}_i\,\boldsymbol{y}_i(t) + \bar{\boldsymbol{u}}(t)^\top \boldsymbol{R}\,\bar{\boldsymbol{u}}(t)\Big)\mathrm{d}t, \tag{5.41}$$

where $\tilde{\boldsymbol{Q}}_i$ is a positive-definite matrix penalizing deviations from the integral state $\boldsymbol{y}_i$. From the results in Section 5.4, such a policy is given by

$$\bar{\boldsymbol{u}} = -\underbrace{\begin{bmatrix} \boldsymbol{K}(t) & \boldsymbol{K}_i(t) \end{bmatrix}}_{\boldsymbol{K}_e(t)} \underbrace{\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_i \end{bmatrix}}_{\boldsymbol{y}_e},$$

or equivalently by

$$\boldsymbol{u} = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\,\boldsymbol{y} - \boldsymbol{K}_i(t)\,\boldsymbol{y}_i,$$

which in terms of the $\boldsymbol{x}$ variables results in the control law

$$\boldsymbol{u} = \boldsymbol{u}_0(t) - \boldsymbol{K}(t)\,\boldsymbol{U}(t)^\top\big(\boldsymbol{x} - \boldsymbol{x}_0(t)\big) - \boldsymbol{K}_i(t)\,\boldsymbol{y}_i.$$

5.5 Examples

We next illustrate the behavior of the computed-torque and LQR controllers on stabilizing an equilibrium point and a trajectory in the five-bars robot of Fig. 3.19. In the former controller, we have chosen the motor angles q_1 and q_5 for the $\boldsymbol{\theta}$ coordinates in Eq. (5.8). For both controllers, we show the effect of forward singularities, model errors, and disturbances.

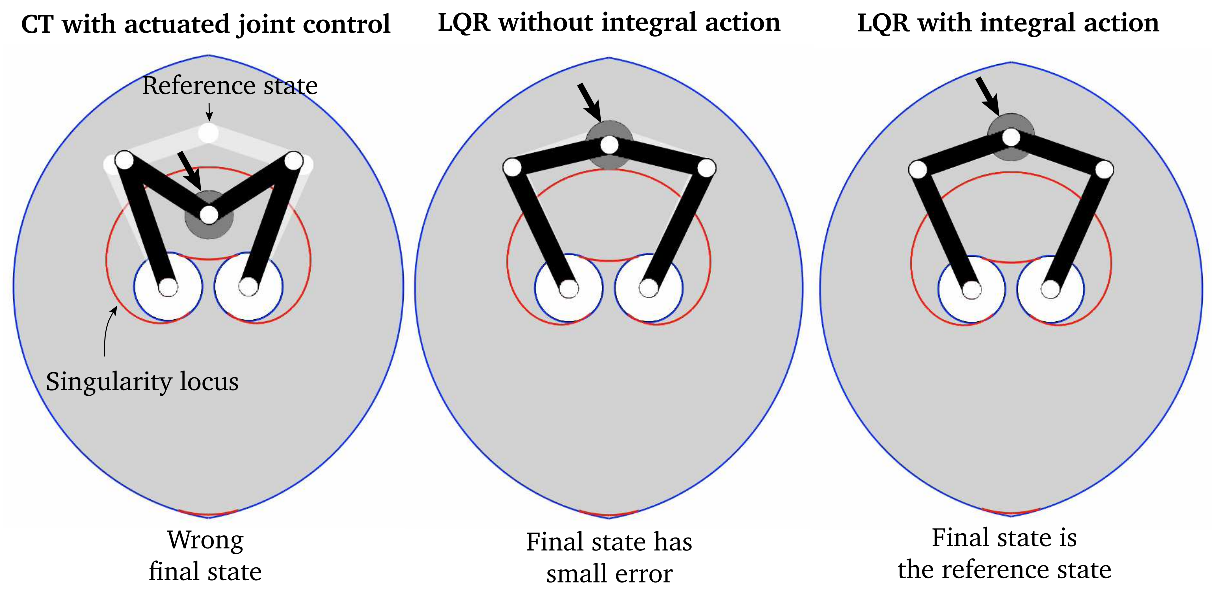

CT with actuated joint control
LQR without integral action
LQR with integral action
Reference state
Singularity locus
Wrong final state
Final state has small error
Final state is the reference state

Figure 5.3: Stabilization of an equilibrium point with the computed-torque and LQR controllers. In all cases the robot must reach the reference state shown in white, while counteracting model errors (the gray load of the end-effector is not accounted for in the model) and force disturbances applied along the black arrow. The computed-torque controller fails not only at a forward singularity, but close to it too. As a result, the full reference state cannot be recovered as the feedback law is only able to control the actuated-joint coordinates. The LQR controller without integral action is robust to forward singularities, but still presents steady-state errors due to model errors. Finally, the LQR controller with integral action stabilizes the system precisely as it is robust to forward singularities, disturbances and model uncertainties. An animation of the simulated motions can be seen in youtu.be/-zVl6B5mg-g.

5.5.1 Stabilization of an Equilibrium Point

In this task, the controller has to keep a load of 0.5 [Kg] at an upright position while compensating gravity (Fig. 5.3). The reference state (depicted in white) is an unstable equilibrium point near the forward singularity locus (depicted in red). During the simulation, we apply a disturbance force of 7 [N] to the end-effector every 3 seconds and maintain it during 1 second (black arrow). Moreover, we simulate the effect of model errors by disregarding the load of the end-effector in the controller model.

The end state of the stabilization task is depicted in black in Fig. 5.3. Due to model errors, the closeness to forward singularities, and the disturbances, the computed-torque controller (Fig. 5.3, left) fails to control the reference state. In fact, since we only control an independent set of coordinates (the actuated joints angles in this case), the system converges to a final state with the same actuated coordinates but different values for the remaining coordinates. In contrast,

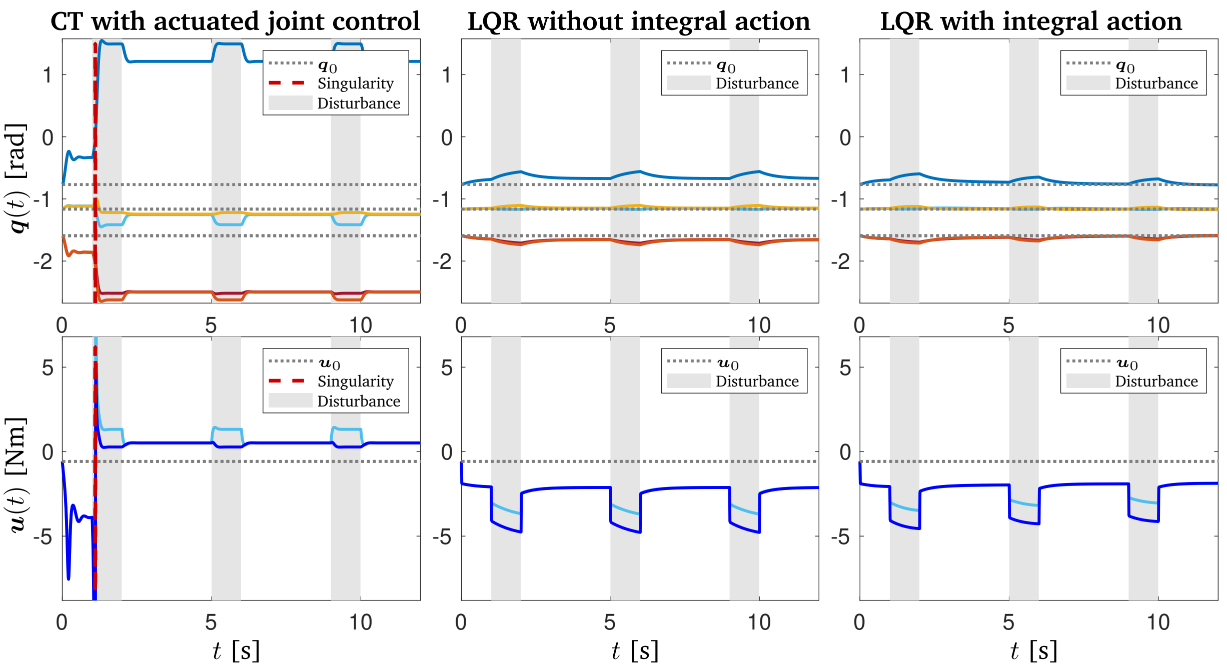


Figure 5.4: Stabilization of an equilibrium point. The plots correspond to the task in Fig. 5.3 subject to disturbances applied during the grey time windows indicated. The CT controller yields very large control actions even near a forward singularity (red dashed line) and fails to stabilize the reference state (dotted line). Instead, it stabilizes a different state, as it is only able to control the actuated joint coordinates. The LQR controller takes into account the full robot state but presents steady state errors due to model uncertainties. Finally, the integral action in the LQR controller removes these steady state errors.

the LQR controller (Fig. 5.3, middle) is robust to forward singularities and, as it controls the full state through the $\boldsymbol{y}$ coordinates, it can keep the robot near the reference state. Note however that with this controller we can still see steady-state errors due to model errors, as the reference and final states are slightly different. To circumvent this issue, we add an integral action to the LQR controller (Fig. 5.3, right), which allows us to finally converge to the reference state.

Fig. 5.4 shows the joint positions $\boldsymbol{q}(t)$ and actions $\boldsymbol{u}(t)$ corresponding to the stabilization task in Fig. 5.3. The reference signals $\boldsymbol{q}_0(t)$ and $\boldsymbol{u}_0(t)$ are the dotted lines, while $\boldsymbol{q}(t)$ and $\boldsymbol{u}(t)$ are the colored lines. Disturbances are applied during the gray time windows indicated. Due to the closeness to the forward singularity locus, the computed-torque controller (Fig. 5.4, left) is not able to compensate the first disturbance and produces an unbounded action when there is a singularity crossing at 1 seconds (red vertical line). As a result, the controller is only able to stabilize the actuated coordinates (yellow and light blue lines) near their reference value, while

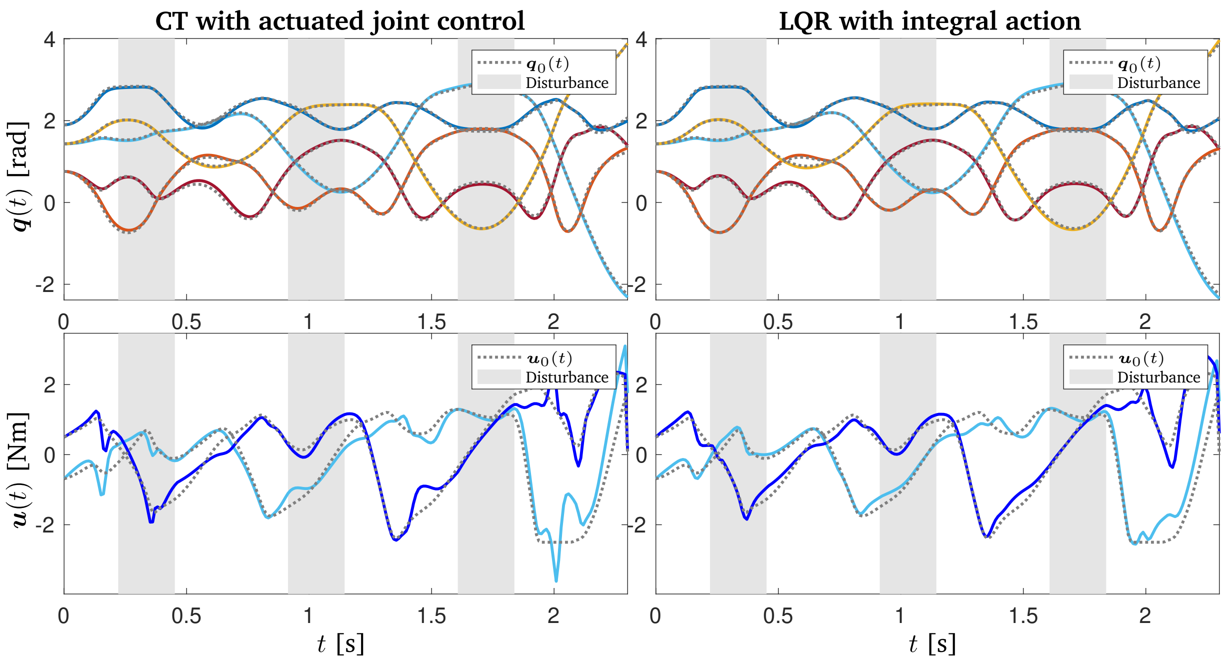


Figure 5.5: Stabilization of a singularity-free trajectory. The robot is subject to disturbances applied at the end-effector during the gray time span, and model errors (due to the load being neglected). Both the computed-torque and LQR controller with integral action are able to stabilize a singularity-avoidance trajectory. An animation of the motion can be seen in youtu.be/cmqpQOc9lMg.

the remaining coordinates never converge to their reference value. In contrast, both variants of the LQR controller stabilize the full robot state (Fig. 5.4 middle and right). Note that the integral action (Fig. 5.4, right) also eliminates the steady-state error.

The convergence rate of the CT controller can be adjusted by changing $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$, while the convergence rate of the LQR controllers can be tuned by changing the $\boldsymbol{Q}$, $\tilde{\boldsymbol{Q}}_i$ and $\boldsymbol{R}$ in Eqs. (5.22) and (5.41). The plot of Fig. 5.4 was obtained with

$$\boldsymbol{K}_p = 500\boldsymbol{I}_2, \quad \boldsymbol{K}_v = 50\boldsymbol{I}_2$$

and

$$\boldsymbol{Q} = \begin{bmatrix} 250\boldsymbol{I}_5 & \boldsymbol{0} \\ \boldsymbol{0} & 50\boldsymbol{I}_5 \end{bmatrix}, \quad \tilde{\boldsymbol{Q}}_i = 200\boldsymbol{I}_2, \quad \boldsymbol{R} = \boldsymbol{I}_2.$$

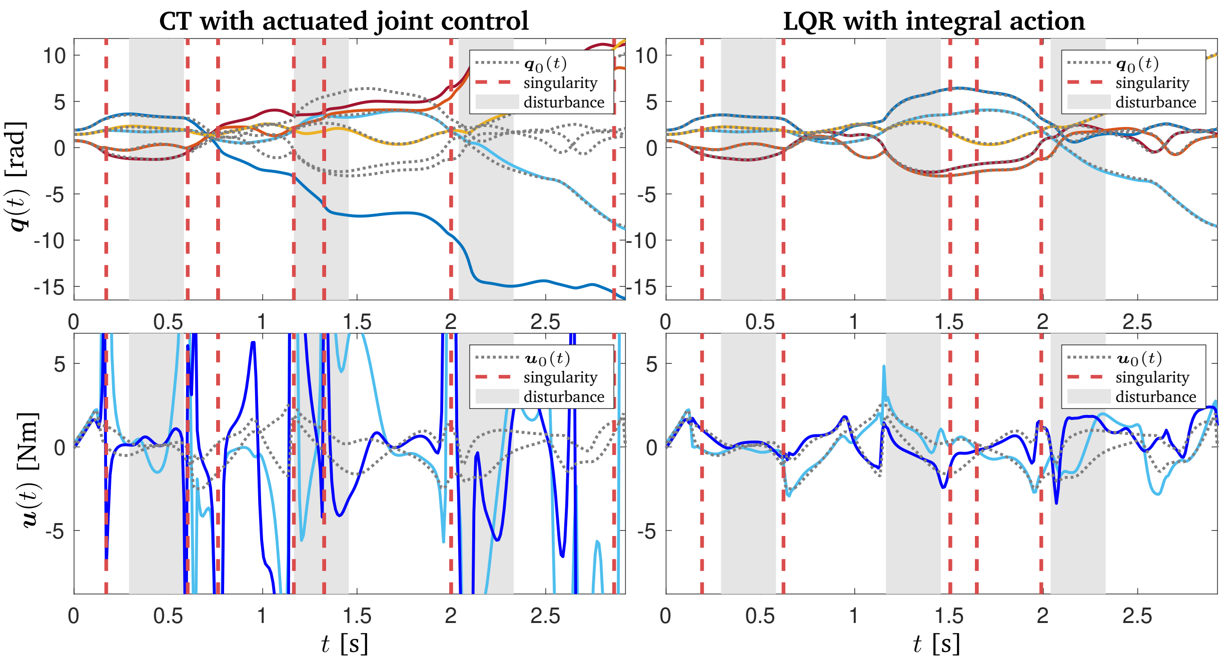


Figure 5.6: Stabilization of a singularity-crossing trajectory. In both cases the robot is subject to disturbances applied at the end-effector during the gray time spans, and to model errors due to neglecting the end-effector load. The computed-torque controller fails when crossing forward singularities (red dashed vertical lines) and generates unbounded actions that can be dangerous for the mechanism structure (left bottom plot). Instead, the LQR controller with integral action is robust to forward singularities while using moderate actions. An animation of the motion can be seen in youtu.be/ZUb0rhmmFgY.

5.5.2 Trajectory Tracking

We next compare the performance of the computed-torque controller with the LQR controller with integral action on tracking trajectories avoiding and crossing forward singularities (Fig. 5.5 and 5.6, respectively). We use both the planner and the optimizer in Chapters 3 and 4 to obtain the reference trajectories $\boldsymbol{x}_0(t)$ and $\boldsymbol{u}_0(t)$ that minimize $\int_0^{t_f} \boldsymbol{u}_0(t)^\top \boldsymbol{u}_0(t)\,\mathrm{d}t$. The trajectories correspond to the weight-throwing task presented in Section 3.9.2.

In both simulations we include force disturbances, which are applied to the end-effector and take place during the gray time windows indicated, and model errors, which are caused by neglecting the weight of the load in the robot model. The time instants in which the singularities are crossed are indicated with a red dashed vertical line. The reference trajectories for all angles q_i are drawn with dotted lines. In the singularity-free trajectory of Fig. 5.5, both the computed-torque and LQR controllers are able to track the trajectory. Note that $\boldsymbol{u}(t)$ is not the same as $\boldsymbol{u}_0(t)$ due to the model errors. Instead, in the singularity-crossing trajectory of Fig. 5.6

the computed-torque controller fails as soon as it is perturbed close to a singularity, at $t = 0.7$ seconds. At this configuration, the inverse dynamic problem produces unbounded torques, so the control of the robot is lost, and the rest of the trajectory cannot be tracked anymore. The computed-torque controller has been implemented in the actuated joint space, and thus $q_1(t)$ and $q_5(t)$ can reliably be tracked, but note that the remaining angles evolve differently because of their bifurcations at forward singularities. On the contrary, the LQR controller with integral action is, as we see, quite robust to disturbances, model uncertainties, and singularity crossings. Even if a disturbance is applied when the system is close to a singularity, the LQR controller is able to converge to the desired trajectory $\boldsymbol{x}_0(t)$.

6 Conclusions

This work has addressed the motion planning and control problems for robotic systems with loop-closure constraints. Such constraints appear frequently in the challenging scenarios where robots are due to operate nowadays, and significantly complicate the planning and execution of motions. Most of the existing motion planning and control algorithms are explicitly designed for systems whose configuration coordinates are independent. Thus, they are not directly applicable to systems with loop-closure constraints. In this thesis, this issue has been overcome by resorting to coordinates that locally parameterize the state space manifold of such systems. By constructing such coordinates around selected states, we have been able to provide 1) a sampling-based planner that uses an efficient steering method based on linear quadratic regulators, 2) trajectory optimization techniques that keep the trajectory on the manifold, and 3) an approach to control the robot motions along the planned trajectories, while being robust to perturbations or model inaccuracies. Thus, this thesis provides a principled and comprehensive toolbox for the planning and control of robot motions under loop-closure constraints.

To develop a sampling-based planner for closed-chain robots we had to address three major hurdles: the generation of random samples on the state-space manifold; the accurate simulation of robot trajectories within such manifold; and the steering of the system towards random states. The three issues have been tackled by constructing an atlas of the manifold in parallel to an RRT. The result is a planner that can explore the state space manifold in an effective manner, while also respecting the dynamic constraints imposed by the equations of motion and the force bounds of the robot. In its fully randomized version (i.e., using randomized steering), the planner is probabilistically complete. We also conjecture it to be probabilistically complete if LQR steering is used, but proving this point has remained elusive so far. The examples included in this work show that the planner can solve significantly complex problems that require the computation of swinging motions between start and goal states, under limited motor torques. The planner can also be used in systems where limits on the constraint forces have to be fulfilled, like in cable-suspended parallel robots. In these robots we have seen that the use of dynamic motions enlarges their workspace substantially, allowing them to reach points that could not be attained otherwise. Moreover, since the proposed planner relies on the simulation of robot actions, it can traverse forward singularities without any trouble. However, these special configurations may be problematic if the planned motions are to be stabilized with traditional computed-torque controllers. Such controllers are powerful because they define global basins of attraction towards the desired trajectories and, when using them, it suffices to sense the positions and velocities of the actuators in order to keep track of the full robot state. However, such controllers generate unbounded torques near forward singularities, and so cannot be used to track trajectories across such configurations. To take advantage of the benefits of these controllers while avoiding their drawbacks, we have proposed an extension of the planner that computes dynamic trajectories avoiding forward singularities. To achieve so, the planner explores the extended state space of singularity-free states that satisfy the loop-closure constraints. The joint consideration of obstacle- and singularity-avoidance, and force and joint limit constraints makes our planner applicable to challenging scenarios.

The trajectories obtained with the planner are both kinematically and dynamically feasible, but not optimal in any particular sense. For this reason, we have developed trajectory optimization methods that improve the planner trajectories according to a Lagrangian cost function. In particular, we have shown that existing trajectory optimization methods may generate trajectories that significantly drift away from the state space manifold. Such a drift typically translates into difficulties when trying to control the trajectory on a real robot, eventually invalidating the designed trajectories. To address this issue, we introduced two optimization methods that do not incur in drift along the obtained trajectories. The first method is simpler, but it only cancels the drift at the discretization points of the trajectory. In contrast, the second method is more costly, but it obtains a drift-free, continuous-time trajectory.

Finally, to robustly execute the obtained trajectories, we have extended the LQR controller to deal with closed-kinematic chains. To achieve so we have used the tangent space parameterization, which allows formulating the LQR controller in independent coordinates. With this technique, we are able to control the system across forward singularities, whereas, as mentioned, existing computed-torque controllers cannot. Specifically, we are able to traverse such singularities under significant disturbances at the end-effector. In contrast, computed-torque controllers easily deviate from the desired trajectory at such critical configurations.

Several points should be considered in further extensions of the work we present. In the planning process, the metric employed to measure the distance between two states certainly deserves consideration. This is a general concern in any motion planner, but it is more difficult to address in our context as the metric should consider the vector flows defined by the equations of motion, but also the curvature of the state space manifold defined by the loop-closure constraints. Using a metric derived from geometric insights provided by such constraints might result in substantial performance improvements. The optimized trajectory, in turn, is only locally-optimal in the set of trajectories homotopic to the initial guess, i.e., the one obtained with the planner. To obtain a globally-optimal trajectory one should resort to asymptotically-optimal planners like those in [60, 61]. In this context, the optimization methods introduced in this thesis could be used as a steering method for the planner, so the local connections could comply with the kinematic and dynamic constraints of the problem, while also being optimal under the considered cost function. Such locally-optimal connections would give rise to globally-optimal trajectories when used in the context of asymptotically-optimal planners. Finally, an effort should be made to validate the proposed motion generation pipeline on real robots.

From a wider perspective, the work presented in this thesis could be continued in different directions. For instance, one might consider situations in which the loop-closure constraints vary over time. This happens in walking robots or in robotic hands that perform fine object manipulations. Since our methods assume such constraints to be permanent, they should be extended with procedures to determine the constraint sequences that are necessary to fulfill a given task [123]. Two additional problems have to be dealt with in this case: eventual impacts that may appear when establishing a new contact and non-holonomic constraints needed to model rolling contacts. While the former may involve substantial research, the latter should easily fit in our algorithms. Non-holonomic constraints are formalized as an additional set of velocity constraints that can be directly considered in the proposed optimization methods. In the fully randomized version of the planner, moreover, such constraints just limit the kind of trajectories that can be simulated from a given state. The consideration of such constraints in the LQR steering method, however, would require a more detailed analysis. We also could extend our methods by considering the possibility of planning in environments with moving obstacles. If the dynamics of the obstacles is known, the extension is relatively straightforward. Otherwise, the dynamics must be learned during planning, and the problem is much challenging. Learning can also play a relevant role if the kinematic and dynamic models are not known beforehand, but must be estimated on-line. We certainly think that all these points deserve further attention.

A
List of Publications

The following is a list of publications resulting from the research reported in this thesis:

Accepted journal papers

[J1] R. Bordalba, L. Ros, and J.M. Porta. "A randomized kinodynamic planner for closed-chain robotic systems," *IEEE Transactions on Robotics*, Vol. 37, No. 1, pp. 99-115, 2021.

Submitted journal papers

[J2] R. Bordalba, T. Schoels, L. Ros, J. M. Porta, and M. Diehl. "Direct Collocation Methods for Trajectory Optimization in Constrained Robotic Systems", *IEEE Transactions on Robotics*, Submitted, 2020.

Journal papers in preparation

[J3] R. Bordalba, J. M. Porta, and L. Ros. "A single-structure controller for singularity-crossing trajectories in closed-chain robots".

[J4] R. Bordalba, J. M. Porta, and L. Ros. "Planning and control of singularity-avoiding motions in closed-chain robots".

Accepted conference papers

[C1] R. Bordalba, L. Ros, and J. M. Porta. "Randomized planning of dynamic motions avoiding forward singularities," *International Symposium on Advances in Robot Kinematics*, Bologna, Italy, pp. 170-178, 2019.

[C2] R. Bordalba, J. M. Porta, and L. Ros. "A singularity-robust LQR controller for parallel robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Spain, pp. 270-276, 2018.

[C3] R. Bordalba, L. Ros, and J. M. Porta. "Randomized kinodynamic planning for constrained systems," *IEEE International Conference on Robotics and Automation*, Brisbane, Australia, pp. 7079-7086, 2018.

[C4] R. Bordalba, L. Ros, and J. M. Porta. "Randomized kinodynamic planning for cable-suspended parallel robots," *International Conference on Cable-Driven Parallel Robots*, Quebec, Canada, pp. 195-206, 2017.

References

[1] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM*, vol. 40, no. 5, pp. 1048—-1066, 1993.

[2] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 657–667, 2005.

[3] I. Bonev, "Delta parallel robot - the story of success," *Newsletter, available at http://www.parallemic.org*, 2001.

[4] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, "Optimization based full body control for the atlas robot," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 120–127, 2014.

[5] M. A. Diftler, J. S. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. A. Permenter, B. K. Hargrave, R. Platt, R. T. Savely, and R. O. Ambrose, "Robonaut 2 - The first humanoid robot in space," in *IEEE International Conference on Robotics and Automation*, pp. 2178–2183, 2011.

[6] R. P. Hoyt, "SpiderFab: An architecture for self-fabricating space systems," in *AIAA Space 2013 Conference and Exposition*, p. 5509, 2013.

[7] J. M. Porta, L. Jaillet, and O. Bohigas, "Randomized path planning on manifolds based on higher-dimensional continuation," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 201–215, 2012.

[8] L. Jaillet and J. M. Porta, "Path planning with loop closure constraints using an atlas-based RRT," *International Symposium on Robotics Research*, 2011.

[9] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[10] K. Hauser, "Fast interpolation and time-optimization with contact," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1231–1250, 2014.

[11] 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, 2015.

[12] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 159–185, 2018.

[13] S. M. LaValle, *Planning Algorithms*. New York: Cambridge University Press, 2006.

[14] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation*, pp. 4569–4574, 2011.

[15] N. Ratliff, M. Zucker, J. A. Bagnell, S. S. Srinivasa, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation*, no. 9-10, pp. 489–494, 2013.

[16] J. Schulman, D. Y, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[17] F. Aghili, "A unified approach for inverse and direct dynamics of constrained multibody systems based on linear projection operator: applications to control and simulation," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 834–849, 2005.

[18] J. G. de Jalón and E. Bayo, *Kinematic and dynamic simulation of multibody systems*. Springer Verlag, 1993.

[19] O. Bohigas, M. Manubens, and L. Ros, *Singularities of robot mechanisms: Numerical computation and avoidance path planning*, vol. 41 of *Mechanisms and Machine Science*. Springer, 2017.

[20] F. C. Park and J. W. Kim, "Singularity Analysis of Closed Kinematic Chains," *ASME Journal of Mechanical Design*, vol. 121, no. 1, pp. 32–38, 1999.

[21] D. Zlatanov, *Generalized Singularity Analysis of Mechanisms*. PhD thesis, University of Toronto, 1998.

[22] O. Bohigas, M. Manubens, and L. Ros, "Singularities of Non-redundant Manipulators: A Short Account and a Method for their Computation in the Planar Case," *Mechanism and Machine Theory*, vol. 68, pp. 1–17, 2013.

[23] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[24] R. M. Murray, Z. Li, H. Kong, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[25] E. Hairer, "Geometric integration of ordinary differential equations on manifolds," *BIT Numerical Mathematics*, vol. 41, no. 5, pp. 996–1007, 2001.

[26] E. Hairer, C. Lubich, and G. Wanner, "Geometric numerical integration: structure-preserving algorithms for ordinary differential equations," 2006.

[27] F. A. Potra and W. C. Rheinboldt, "On the Numerical Solution of Euler-Lagrange Equations∗," *Mechanics of Structures and Machines*, vol. 19, no. 1, pp. 1–18, 1991.

[28] E. J. Haug, "An Ordinary Differential Equation Formulation for Multibody Dynamics: Nonholonomic Constraints," *Journal of Computing and Information Science in Engineering*, vol. 17, no. 1, 2016.

[29] E. J. Haug, "Multibody Dynamics on Differentiable Manifolds," *Journal of Computational and Nonlinear Dynamics*, vol. 16, no. 4, 2021.

[30] F. A. Potra and J. Yen, "Implicit numerical integration for Euler-Lagrange equations via tangent space parametrization," *Journal of Structural Mechanics*, vol. 19, no. 1, pp. 77–98, 1991.

[31] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[32] G. H. Golub and C. F. V. Loan, *Matrix Computations*. John Hopkins University Press, 2013.

[33] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[34] R. Featherstone, *Robot dynamics algorithms*. Kluwer, Norwell, MA, 1987.

[35] R. Featherstone and D. E. Orin, "Dynamics," in *Springer Handbook of Robotics*, pp. 37–66, Springer, 2016.

[36] F. C. Park, B. Kim, C. Jang, and J. Hong, "Geometric algorithms for robot dynamics: A tutorial review," *Applied Mechanics Reviews*, vol. 70, no. 1, 2018.

[37] J. Duffy, *Statics and kinematics with applications to robotics*. Cambridge University Press, 1996.

[38] J. K. Davidson and K. H. Hunt, *Robots and Screw Theory: Applications of Kinematics and Statics to Robotics*. Oxford University Press, 2004.

[39] R. Featherstone, "The acceleration vector of a rigid body," *The International Journal of Robotics Research*, vol. 20, pp. 841–846, nov 2001.

[40] J.-P. Samin and P. Fisette, *Symbolic modeling of multibody systems*. Springer, 2003.

[41] S. M. Lavalle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[42] B. Kim, T. T. Um, C. Suh, and F. C. Park, "Tangent bundle RRT: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, no. 1, pp. 202–225, 2016.

[43] Z. Kingston, M. Moll, and L. E. Kavraki, "Decoupling constraints from sampling-based planners," in *International Symposium of Robotics Research*, 2017.

[44] 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, 1985.

[45] 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, 1985.

[46] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal on Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[47] 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.

[48] 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.

[49] 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, 2014.

[50] Q.-C. Pham, S. Caron, P. Lertkultanon, and Y. Nakamura, “Admissible velocity propagation: Beyond quasi-static path planning for high-dimensional robots,” *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 44–67, 2017.

[51] Q.-C. Pham, S. Caron, and Y. Nakamura, “Kinodynamic planning in the configuration space via admissible velocity propagation,” in *Proceedings of Robotics: Science and Systems*, 2013.

[52] T. Kunz and M. Stilman, “Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3713–3719, 2014.

[53] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, “LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics,” in *IEEE International Conference on Robotics and Automation*, pp. 2537–2542, 2012.

[54] R. Tedrake, “LQR-Trees: Feedback motion planning on sparse randomized trees,” in *Robotics: Science and Systems*, 2009.

[55] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, “Optimal sampling-based planning for linear-quadratic kinodynamic systems,” in *IEEE International Conference on Robotics and Automation*, pp. 2429–2436, 2013.

[56] D. J. Webb and J. van den Berg, “Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics,” in *IEEE International Conference on Robotics and Automation*, pp. 5054–5061, 2013.

[57] J. H. Yakey, S. M. LaValle, and L. E. Kavraki, “Randomized path planning for linkages with closed kinematic chains,” *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 951–959, 2001.

[58] M. Stilman, “Task constrained motion planning in robot joint space,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3074–3081, 2007.

[59] D. Berenson, S. Srinivasa, and J. J. Kuffner, “Task space regions: A framework for pose-constrained manipulation planning,” *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[60] Y. Li, Z. Littlefield, and K. E. Bekris, “Asymptotically optimal sampling-based kinodynamic planning,” *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.

[61] K. Hauser and Y. Zhou, “Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1431–1443, 2016.

[62] L. R. Petzold, “Numerical solution of differential-algebraic equations in mechanical systems simulation,” *Physica D: Nonlinear Phenomena*, vol. 60, no. 1–4, pp. 269–279, 1992.

[63] W. Blajer, “Methods for constraint violation suppression in the numerical simulation of constrained multibody systems – A comparative study,” *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 13-16, pp. 1568–1576, 2011.

[64] J. M. Lee, *Introduction to Smooth Manifolds*. Springer, 2001.

[65] M. E. Henderson, “Multiple parameter continuation: computing implicitly defined k-manifolds,” *International Journal of Bifurcation and Chaos*, vol. 12, no. 3, pp. 451–476, 2002.

[66] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[67] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi – A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[68] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific, 2005.

[69] J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *IEEE Int. Conf. on Robotics and Automation*, vol. 2, pp. 995–1001, 2000.

[70] J. M. Porta and L. Jaillet, “Sampling Strategies for Path Planning under Kinematic Constraints,” *CoRR*, vol. abs/1407.2, 2014.

[71] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation and Control*. Taylor Francis, 1975.

[72] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, “Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 277–283, 2019.

[73] L. Jaillet and J. M. Porta, “Efficient Asymptotically-optimal Path Planning on Manifolds,” *Robotics and Autonomous Systems*, vol. 61, no. 8, pp. 797–807, 2013.

[74] S. Briot and V. Arakelian, “Optimal force generation in parallel manipulators for passing through the singular positions,” *The International Journal of Robotics Research*, vol. 27, no. 8, pp. 967–983, 2008.

[75] M. Özdemir, “Removal of singularities in the inverse dynamics of parallel robots,” *Mechanism and Machine Theory*, vol. 107, pp. 71–86, 2017.

[76] J. M. Porta, L. Ros, O. Bohigas, M. Manubens, C. Rosales, and L. Jaillet, “The Cuik Suite: Analyzing the motion of closed-chain multibody systems,” *IEEE Robotics and Automation Magazine*, vol. 21, no. 3, pp. 105–114, 2014.

[77] F. Bourbonnais, P. Bigras, and I. A. Bonev, “Minimum-time trajectory planning and control of a pick-and-place five-bar parallel robot,” *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 2, pp. 740–749, 2015.

[78] J. Albus, R. V. Bostelman, and N. Dagalakis, “The NIST Robocrane,” *Journal of Robotic Systems*, vol. 10, no. 5, pp. 709–724, 1993.

[79] O. Bohigas, M. Manubens, and L. Ros, “Planning wrench-feasible motions for cable-driven hexapods,” *IEEE Transactions on Robotics*, vol. 32, no. 2, pp. 442–451, 2016.

[80] C. Gosselin, P. Ren, and S. Foucault, “Dynamic trajectory planning of a two-DOF cable-suspended parallel robot,” in *IEEE International Conference on Robotics and Automation*, pp. 1476–1481, 2012.

[81] G. Barrette and C. Gosselin, “Determination of the dynamic workspace of cable-driven planar parallel mechanisms,” *Transactions of the ASME-R-Journal of Mechanical Design*, vol. 127, no. 2, pp. 242–248, 2005.

[82] C. Gosselin and S. Foucault, “Dynamic point-to-point trajectory planning of a two-DOF cable-suspended parallel robot,” *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 728–736, 2014.

[83] X. Jiang and C. Gosselin, “Dynamic point-to-point trajectory planning of a three-DOF cable-suspended parallel robot,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1550–1557, 2016.

[84] D. A. Benson, G. T. Huntington, T. P. Thorvaldsen, and A. V. Rao, “Direct trajectory optimization and costate estimation via an orthogonal collocation method,” *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 6, pp. 1435–1440, 2006.

[85] J. T. Betts, “Survey of numerical methods for trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[86] M. Kelly, “An introduction to trajectory optimization: How to do your own direct collocation,” *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[87] M. Posa, S. Kuindersma, and R. Tedrake, “Optimization and stabilization of trajectories for constrained dynamical systems,” *IEEE International Conference on Robotics and Automation*, pp. 1366–1373, 2016.

[88] D. Pardo, M. Neunert, A. Winkler, R. Grandia, and J. Buchli, “Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion,” in *Robotics: Science and Systems*, 2017.

[89] M. Posa, C. Cantu, and R. Tedrake, “A direct method for trajectory optimization of rigid bodies through contact,” *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[90] A. Patel, S. L. Shield, S. Kazi, A. M. Johnson, and L. T. Biegler, “Contact-implicit trajectory optimization using orthogonal collocation,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2242–2249, 2019.

[91] M. L. Felis, K. Mombaur, and A. Berthoz, “An optimal control approach to reconstruct human gait dynamics from kinematic data,” in *IEEE-RAS International Conference on Humanoid Robots*, pp. 1044–1051, 2015.

[92] W. Xi, Y. Yesilevskiy, and C. D. Remy, “Selecting gaits for economical locomotion of legged robots,” *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1140–1154, 2016.

[93] W. Xi and C. D. Remy, “Optimal gaits and motions for legged robots,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3259–3265, 2014.

[94] R. Bonalli, A. Bylard, A. Cauligi, T. Lew, and M. Pavone, “Trajectory optimization on manifolds: A theoretically-guaranteed embedded sequential convex programming approach,” in *Robotics: Science and Systems*, 2019.

[95] S. Gros, M. Zanon, and M. Diehl, “Baumgarte stabilisation over the SO(3) rotation group for control,” in *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 620–625, IEEE, 2015.

[96] S. Gros, M. Zanon, M. Vukov, and M. Diehl, “Nonlinear MPC and MHE for mechanical multi-body systems with application to fast tethered airplanes,” *IFAC Proceedings Volumes*, vol. 45, no. 17, pp. 86–93, 2012.

[97] M. Erhard, G. Horn, and M. Diehl, “A quaternion-based model for optimal control of an airborne wind energy system,” *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 97, no. 1, pp. 7–24, 2016.

[98] J. De Schutter, R. Leuthold, and M. Diehl, “Optimal control of a rigid-wing rotary kite system for airborne wind energy,” in *2018 European Control Conference (ECC)*, pp. 1734–1739, IEEE, 2018.

[99] J. De Schutter, R. Leuthold, T. Bronnenmeyer, R. Paelinck, and M. Diehl, “Optimal control of stacked multi-kite systems for utility-scale airborne wind energy,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4865–4870, IEEE, 2019.

[100] J. Baumgarte, “Stabilization of constraints and integrals of motion in dynamical systems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 1, no. 1, pp. 1–16, 1972.

[101] D. Liberzon, *Calculus of variations and optimal control theory: a concise introduction*. Princeton university press, 2011.

[102] J. T. Betts, “Practical Methods for Optimal Control and Estimation Using Nonlinear Programming,” *SIAM Review*, 2010.

[103] J.-P. Berrut and L. N. Trefethen, “Barycentric lagrange interpolation,” *SIAM review*, vol. 46, no. 3, pp. 501–517, 2004.

[104] B. Jones, *Orbit Propagation Using Gauss-Legendre Collocation*, pp. AIAA–2012–4967–1–16. 2012.

[105] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer Series in Operations Research and Financial Engineering, Science & Business Media, 2006.

[106] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[107] HSL, “A collection of Fortran codes for large-scale scientific computation,” *http://www. hsl. rl. ac. uk*.

[108] R. Geraerts and M. H. Overmars, “Creating High-quality Paths for Motion Planning,” *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[109] T. Schoels, L. Palmieri, K. O. Arras, and M. Diehl, “An NMPC approach using convex inner approximations for online motion planning with guaranteed collision avoidance,” in *IEEE International Conference on Robotics and Automation*, pp. 3574–3580, 2020.

[110] Z. Wang and F. H. Ghorbel, “Control of closed kinematic chains using a singularly perturbed dynamics model,” *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, vol. 128, no. 1, pp. 142–151, 2006.

[111] F. H. Ghorbel, O. Chételat, R. Gunawardana, and R. Longchamp, “Modeling and set point control of closed-chain mechanisms: Theory and experiment,” *IEEE Transactions on Control Systems Technology*, vol. 8, no. 5, pp. 801–815, 2000.

[112] A. Codourey, “Dynamic Modeling of Parallel Robots for Computed-Torque Control Implementation,” *The International Journal of Robotics Research*, vol. 17, no. 12, pp. 1325–1336, 1998.

[113] W. W. Shang, S. Cong, and Y. Ge, “Adaptive computed torque control for a parallel manipulator with redundant actuation,” *Robotica*, vol. 30, no. 3, pp. 457–466, 2012.

[114] S. S. Parsa, R. Boudreau, and J. A. Carretero, “Reconfigurable mass parameters to cross direct kinematic singularities in parallel manipulators,” *Mechanism and Machine Theory*, vol. 85, pp. 53–63, 2015.

[115] H. Cheng, Y. K. Yiu, and Z. Li, “Dynamics and Control of Redundantly Actuated Parallel Manipulators,” *IEEE/ASME Transactions on Mechatronics*, vol. 8, no. 4, pp. 483–491, 2003.

[116] J. H. Lee, B. J. Yi, S. R. Oh, and I. H. Suh, “Optimal design of a five-bar finger with redundant actuation,” in *IEEE International Conference on Robotics and Automation*, pp. 2068–2074, 1998.

[117] C. K. Kevin Jui and Q. Sun, “Path tracking of parallel manipulators in the presence of force singularity,” *Journal of dynamic systems, measurement, and control*, vol. 127, no. 4, pp. 550–563, 2005.

[118] S. K. Ider, “Inverse dynamics of parallel manipulators in the presence of drive singularities,” *Mechanism and Machine Theory*, vol. 40, no. 1, pp. 33–44, 2005.

[119] R. B. Hill, D. Six, A. Chriette, S. Briot, and P. Martinet, “Crossing type 2 singularities of parallel robots without pre-planned trajectory with a virtual-constraint-based controller,” in *IEEE International Conference on Robotics and Automation*, pp. 6080–6085, 2017.

[120] G. Pagis, N. Bouton, S. Briot, and P. Martinet, “Enlarging parallel robot workspace through Type-2 singularity crossing,” *Control Engineering Practice*, vol. 39, pp. 1–11, 2015.

[121] S. Mason, N. Rotella, S. Schaal, and L. Righetti, “Balancing and walking using full dynamics LQR control with contact constraints,” in *IEEE-RAS International Conference on Humanoid Robots*, pp. 63–68, 2016.

[122] W. F. Arnold and A. J. Laub, “Generalized eigenproblem algorithms and software for algebraic Riccati equations,” *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1746–1754, 1984.

[123] J. Xiao and X. Ji, “Automatic generation of high-level contact state space,” *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 584–606, 2001.

Notation

Scalars, Vectors, and Matrices

x	A scalar variable, typically a coordinate.
t	The time parameter.
$\boldsymbol{x}$	A vector.
$\boldsymbol{x}^\top$	The transpose of vector $\boldsymbol{x}$.
$\boldsymbol{x}(t)$	A time-dependent vector function
$\dot{\boldsymbol{x}}(t)$	The time derivative of $\boldsymbol{x}(t)$. Sometimes the dependency on t is omitted.
$\boldsymbol{X}$	A matrix.
$\boldsymbol{X}^\top$	The transpose of matrix $\boldsymbol{X}$.
$\boldsymbol{X}^{-1}$	The inverse of matrix $\boldsymbol{X}$.
$\boldsymbol{X}^{+}$	The the Moore-Penrose pseudoinverse of matrix $\boldsymbol{X}$.
$\boldsymbol{X}(t)$	A time-dependent matrix function
$\dot{\boldsymbol{X}}(t)$	The time derivative of $\boldsymbol{X}(t)$. Sometimes the dependency on t is omitted.
$\mathbf{0}$	A vector or matrix of zeros, understood by context.
$\boldsymbol{I}$	The identity matrix.
$\boldsymbol{I}_n$	The $n \times n$ identity matrix.

Kinematic Spaces

$\boldsymbol{q}$	Generalized coordinates.
n_q	Number of generalized coordinates.
$\boldsymbol{\Phi}(\boldsymbol{q})$	Loop-closure constraint.
n_e	Number of loop-closure constraints.
$\boldsymbol{\Phi}_{\boldsymbol{q}}$	Jacobian $\partial\boldsymbol{\Phi}/\partial\boldsymbol{q}$ of the loop-closure constraint.
$\mathcal{C}$	Configuration space, or C-space for short.
$d_{\mathcal{C}}$	Dimension of the C-space given by $d_{\mathcal{C}} = n_q - n_e$.
$\dot{\boldsymbol{q}}$	Generalized velocity. Time derivative of $\boldsymbol{q}$.
$\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$	Tangent space of $\mathcal{C}$ at $\boldsymbol{q}$.
$\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$	Normal space to $\mathcal{C}$ at $\boldsymbol{q}$.
$\boldsymbol{x}$	Robot state composed by $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.
n_x	Number of states.
$\boldsymbol{F}(\boldsymbol{x})$	System of equations defining the state space.
$\boldsymbol{F}_{\boldsymbol{x}}$	Jacobian $\partial\boldsymbol{F}/\partial\boldsymbol{x}$.
$\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}$	Time derivative of the Jacobian $\partial\boldsymbol{\Phi}/\partial\boldsymbol{q}$.

$\mathcal{X}$	State space.
$d_{\mathcal{X}}$	Dimension of the state space given by $d_{\mathcal{X}} = n_x - 2n_e$.
$\mathcal{T}_{\boldsymbol{x}}\mathcal{X}$	Tangent space of $\mathcal{X}$ at $\boldsymbol{x}$.
$\ddot{\boldsymbol{q}}$	Generalized acceleration. Time derivative of $\dot{\boldsymbol{q}}$.
$\boldsymbol{\xi}$	Kinematic term $-\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\,\dot{\boldsymbol{q}}$ in the acceleration constraint.
$\mathcal{A}_{\boldsymbol{x}}$	The acceleration space.
$\ddot{\boldsymbol{q}}_{\perp}$	Component decomposing the acceleration $\ddot{\boldsymbol{q}}$ lying in $\mathcal{N}_{\boldsymbol{q}}\mathcal{C}$.
$\ddot{\boldsymbol{q}}_{\|}$	Component decomposing the acceleration $\ddot{\boldsymbol{q}}$ lying in $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$.
$\boldsymbol{\Phi}_{\boldsymbol{q}}^{+}$	Moore-Penrose pseudoinverse of $\boldsymbol{\Phi}_{\boldsymbol{q}}$.
$\boldsymbol{\Lambda}$	An $n_q \times d_{\mathcal{C}}$ matrix that has, by columns, a basis of $\mathcal{T}_{\boldsymbol{q}}\mathcal{C}$.
$\boldsymbol{q}_u$	Actuated coordinates of $\boldsymbol{q}$.
n_u	Number of actuated coordinates.
$\boldsymbol{Q}_u$	Matrix selecting the actuated coordinates.
$\boldsymbol{q}_r$	Remaining (non-actuated) coordinates of $\boldsymbol{q}$.
n_r	Number $n_q - n_u$ of remaining coordinates.
$\boldsymbol{Q}_r$	Matrix selecting the remaining coordinates.
$\boldsymbol{\Phi}_{\boldsymbol{q}_u}$	Jacobian $\partial\boldsymbol{\Phi}/\partial\boldsymbol{q}_u$ of the loop-closure constraint $\boldsymbol{\Phi}$.
$\boldsymbol{\Phi}_{\boldsymbol{q}_r}$	Jacobian $\partial\boldsymbol{\Phi}/\partial\boldsymbol{q}_r$ of the loop-closure constraint $\boldsymbol{\Phi}$.
$\dot{\boldsymbol{q}}_u$	Velocity of actuated coordinates.
$\dot{\boldsymbol{q}}_r$	Velocity of remaining coordinates.

Dynamics

K	Kinetic energy of the robot.
U	Potential energy of the robot.
$\boldsymbol{\lambda}$	Vector of Lagrange multipliers.
$\boldsymbol{\tau}_{\text{fric}}$	Generalized force of friction.
$\boldsymbol{u}$	Action vector.
$\boldsymbol{M}$	Mass matrix.
$\boldsymbol{G}$	Generalized gravity vector. It is the only vector denoted in capital letter for consistency with the literature.
$\boldsymbol{C}$	Coriolis matrix.
$\boldsymbol{\tau}_{\text{FD}}$	Generalized vector including forward dynamics terms.
$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$	First order forward dynamics in $\boldsymbol{x}$ coordinates.
$\boldsymbol{\tau}_{\text{ID}}$	Generalized vector including inverse dynamics terms.
$\boldsymbol{N}$	Matrix defined as $\boldsymbol{\Lambda}^{\top}\boldsymbol{Q}_u$ in the inverse dynamics.

Recursive Dynamic Algorithms

$\boldsymbol{w}$	Angular velocity of the body about point O.
$\boldsymbol{v}_O$	Linear velocity of the body-fixed point that coincide with point O at the current instant.
$\boldsymbol{\tau}_O$	Pure couple about point O.
$\boldsymbol{f}$	Force acting through point O.
${}^A\hat{\boldsymbol{v}}$	Spatial velocity expressed in the A coordinate system.
${}^A\boldsymbol{X}_B$	Spatial motion transform from coordinate B to A.
${}^A\boldsymbol{R}_B$	Orthogonal rotation matrix from B to A coordinates.
${}^A\vec{\boldsymbol{p}}_B$	Translation locating the origin of B relative to A.
$\boldsymbol{S}(\vec{\boldsymbol{p}})$	Skew-symmetric matrix.
${}^A\hat{\boldsymbol{f}}$	Spatial force expressed in the A coordinate system.
${}^A\boldsymbol{X}_B^*$	Spatial force transform from B to A coordinates.
$\boldsymbol{\mathcal{I}}_{\text{CM}}$	Tensor of inertia about the body's center of mass.
${}^O\hat{\boldsymbol{I}}$	Spatial inertia about O.
$\mathcal{L}_i$	Link i.
J_i	Joint i.
$\boldsymbol{X}_z(q_i)$	Joint transform about Z_i (Table 2.2).
$\boldsymbol{X}_{\mathcal{L}_i}$	Spatial transformation locating the $(i-1)'$ coordinate relative to the $(i-1)$ one.
L_i	Distance between two revolute joints of link $\mathcal{L}_i$.
$\hat{\boldsymbol{v}}_n$	Spatial velocity of the tip link expressed in 0 coordinates.
${}^0\hat{\boldsymbol{S}}_i$	Unit twist of link $\mathcal{L}_i$ as observed from link $\mathcal{L}_{i-1}$ expressed in 0 coordinates.
${}^i\hat{\boldsymbol{S}}_i$	Unit twist of link $\mathcal{L}_i$ as observed from link $\mathcal{L}_{i-1}$ expressed in link $\mathcal{L}_i$ coordinates (Table 2.2).
$\boldsymbol{J}_n$	Screw Jacobian of the kinematic loop.
$\dot{\boldsymbol{J}}_n$	Time derivative of the screw Jacobian of the kinematic loop.
$\hat{\boldsymbol{a}}_n$	Spatial acceleration of the tip link expressed in 0 coordinates.
$\hat{\boldsymbol{f}}_c$	Spatial constraint force closing the kinematic loop.
$\boldsymbol{\tau}_c$	Generalized constraint force.
${}^i\hat{\boldsymbol{v}}_i$	Spatial velocity of $\mathcal{L}_i$ expressed in i coordinates.
${}^i\hat{\boldsymbol{a}}_i$	Spatial acceleration of $\mathcal{L}_i$ expressed in i coordinates.
${}^i\hat{\boldsymbol{f}}_{i,\text{net}}$	Spatial net force acting on $\mathcal{L}_i$ expressed in i coordinates.
${}^i\hat{\boldsymbol{f}}_i$	Spatial force transmitted across joint J_i expressed in i coordinates.
${}^i\hat{\boldsymbol{f}}_{i,\text{ext}}$	Spatial external force acting on $\mathcal{L}_i$ expressed in i coordinates.

Atlas

V_c	Open set in $\mathcal{X}$.
P_c	Open set in $\mathbb{R}^{d_\mathcal{X}}$.
$\boldsymbol{\varphi}_c$	Chart from an open set $V_c \in \mathcal{X}$ to an open set $P_c \in \mathbb{R}^{d_\mathcal{X}}$, such that $\boldsymbol{y} = \boldsymbol{\varphi}_c(\boldsymbol{x})$.
$\boldsymbol{y}$	Local coordinates, or parameters, of $\boldsymbol{x}$ in a given chart.
$\boldsymbol{\psi}_c$	Inverse map of $\boldsymbol{\varphi}_c$ providing a local parametrization of $V_c \subset \mathcal{X}$, such that $\boldsymbol{x} = \boldsymbol{\psi}_c(\boldsymbol{y})$.
$\boldsymbol{x}_c$	Chart center.
$\boldsymbol{U}_c$	An $n_x \times d_\mathcal{X}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$.
$\tilde{\boldsymbol{g}}(\boldsymbol{y}, \boldsymbol{u})$	First order forward dynamics expressed in $\boldsymbol{y}$ coordinates.

Sampling-based Planner

$\mathcal{U}$	Action space.
$u_{i,\text{max}}$	Maximum action in the i-th actuator.
$\mathcal{X}_\text{feas}$	Feasible state space $\mathcal{X}$.
$\boldsymbol{x}_s$	Start state.
$\boldsymbol{x}_g$	Goal state.
t_g	Trajectory goal time.
$\boldsymbol{x}_\text{rand}$	Guiding random state in the RRT algorithm.
$\boldsymbol{x}_\text{near}$	The RRT state closest to random state $\boldsymbol{x}_\text{rand}$.
ϵ	RRT parameter to limit the maximal distance between $\mathcal{T}_{\boldsymbol{x}_c}\mathcal{X}$ and the manifold $\mathcal{X}$.
α	RRT parameter to ensure bounded curvature in the part of $\mathcal{X}$ covered by a chart.
ρ	RRT parameter to guarantee the generation of new charts as the RRT grows.
$\mathcal{U}_s$	Discrete set to approximate the action space $\mathcal{U}$.
Δt	Time increment.
σ	RRT parameter determining the sampling radius in local coordinates.
A	Atlas.
T_s	Tree rooted at $\boldsymbol{x}_s$.
T_g	Tree rooted at $\boldsymbol{x}_g$.
δ	RRT parameter to measure closeness between states.
β	RRT parameter to measure closeness between trees.
b	Auxiliar variable used to plan in the singularity-free state space.
$d(\boldsymbol{q})$	Determinant of the matrix that must remain full rank in the singularity-free state space.
$\mathcal{X}_\text{sfree}$	Singularity-free state space.
N_s	Number of samples generated during the RRT algorithm.
N_c	Number of charts generated during the RRT algorithm.

Trajectory Optimization

$\boldsymbol{h}(\boldsymbol{x},\boldsymbol{u})$	Differentiable function defining inequality constraints.
$L(\boldsymbol{x}(t),\boldsymbol{u}(t))$	Running cost function.
t_f	Trajecotry time.
t_k	Time instant or knot point k.
Δt	Time increment.
$\boldsymbol{x}_k$	State at the k-th knot point.
$\boldsymbol{u}_k$	Action at the k-th knot point.
N	Number of collocation intervals.
$\boldsymbol{p}_k(t)$	Lagrange interpolation polynomial at the k-th collocation interval.
$l_j(t)$	Basis of Lagrange interpolation polynomial.
d	Degree of the interpolation polynomial $\boldsymbol{p}_k(t)$.
$t_{k,i}$	Collocation time (or point) i at the k-th collocation interval.
$\boldsymbol{x}_{k,i}$	State at the i-th collocation time and k-th collocation interval.
$\boldsymbol{x}_{k,i}$	Action at the i-th collocation time and k-th collocation interval.
$\dot{\boldsymbol{p}}_k(t)$	Time derivative of the Lagrange interpolation polynomial $\boldsymbol{p}_k(t)$.
$\boldsymbol{D}$	Constant differentiation matrix of the Lagrange interpolation polynomial.
$\boldsymbol{w}$	Optimization variables of the NLP.
$C(\boldsymbol{w})$	Discrete version of the optimization cost.
$\boldsymbol{U}_g$	An $n_x \times d_{\mathcal{X}}$ matrix whose columns provide an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_g}\mathcal{X}$.
β'	Baumgarte stabilizing gain of the position constraint.
α'	Baumgarte stabilizing gain of the velocity constraint.
$\boldsymbol{g}_{\text{stab}}$	Baumgarte stabilized first order forward dynamics.
$\boldsymbol{x}'_k$	State drifted from $\mathcal{X}$ at the end of the k-th polynomial in the projection method.
$\boldsymbol{\mu}_k$	Auxiliary projection vector to eliminate the drift from $\mathcal{X}$ at the end of the k-th polynomial.
$\boldsymbol{y}_{k,i}$	Local coordinates at the i-th collocation time and k-th collocation interval.
$\boldsymbol{H}_k(\boldsymbol{x},\boldsymbol{y})$	Implicit function defining the inverse map $\boldsymbol{x} = \boldsymbol{\psi}_k(\boldsymbol{y})$.
$\boldsymbol{\mathcal{D}}(\boldsymbol{x},\dot{\boldsymbol{x}},\boldsymbol{u},\boldsymbol{\lambda})$	Implicit form of the forward dynamics.
$\boldsymbol{\mathcal{D}}_{\text{stab}}(\boldsymbol{x},\dot{\boldsymbol{x}},\boldsymbol{u},\boldsymbol{\lambda})$	Implicit form of the stabilized forward dynamics.
d_k	Projection distance at the k-th collocation interval.
$\boldsymbol{Q}_k$	An $n_x \times n_x$ orthonormal matrix from the QR decomposition of $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)$.
$\boldsymbol{R}_k$	An $n_x \times n_e$ upper triangular matrix from the QR decomposition of $\boldsymbol{F}_{\boldsymbol{x}}(\boldsymbol{x}_k)$.
$\boldsymbol{V}_k$	The frist n_e columns of $\boldsymbol{Q}_k$.
$\boldsymbol{U}_k$	The last $d_{\mathcal{X}}$ columns of $\boldsymbol{Q}_k$ defining an orthonormal basis of $\mathcal{T}_{\boldsymbol{x}_k}\mathcal{X}$.

e_K	Kinematic error of the trajectory.
e_D	Dynamic error of the trajectory.
E_K	Average of the kinematic error of the trajectory.
E_D	Average of the dynamic error of the trajectory.

LQR

$\boldsymbol{A}$	State matrix of the linearized system.
$\boldsymbol{B}$	Action matrix of the linearized system.
$\boldsymbol{c}$	Affine term of the linearized system.
J	LQR cost function to minimize.
$\boldsymbol{u}^*$	LQR optimal control function.
t_f^*	Optimal final time from the LQR steering.
$\boldsymbol{R}$	Symmetric positive-definite penalizing high control actions.
H	Hamiltonian function from the Pontryiagin's minimum principle in the LQR steering.
$\boldsymbol{r}$	Evolution of the linear system when no control is applied in the LQR steering.
$\boldsymbol{G}_\mathrm{r}$	Weighted continuous reachability Gramian in the LQR steering.
t_max	Expected maximum optimal time t_f^* in the LQR steering.
$\boldsymbol{x}_0(t)$	Reference state trajectory.
$\boldsymbol{u}_0(t)$	Reference action trajectory.
$\bar{\boldsymbol{x}}$	State error given by $\boldsymbol{x} - \boldsymbol{x}_0$.
$\bar{\boldsymbol{u}}$	Action error given by $\boldsymbol{u} - \boldsymbol{u}_0$.
$\boldsymbol{Q}$	Positive semi-definite matrix penalizing deviations from the reference trajectory.
$\boldsymbol{K}$	LQR optimal control gain.
$\boldsymbol{S}$	Solution of the differential Riccati equation in the LQR.
$\tilde{\boldsymbol{Q}}$	Positive semi-definite matrix penalizing deviations from the reference trajectory in local coordinates.
$\boldsymbol{C}$	An $d_\mathcal{C} \times n_x$ output matrix selecting $d_\mathcal{C}$ components of the error signal $\bar{\boldsymbol{x}}$.
$\boldsymbol{y}_i$	Integral state in the LQR controller.
$\boldsymbol{y}_e$	Extended state including the integral state.
$\boldsymbol{A}_e$	Extended $\boldsymbol{A}$ matrix of the linearized system including the integral state.
$\boldsymbol{B}_e$	Extended $\boldsymbol{B}$ matrix of the linearized system including the integral state.
$\tilde{\boldsymbol{Q}}_i$	Positive semi-definite matrix penalizing deviations from the integral state $\boldsymbol{y}_i$.
$\boldsymbol{K}_i$	LQR optimal control gain for the integral state.

Computed-torque Controller

$\boldsymbol{\pi}(\boldsymbol{x}, t)$	Control law.
$\boldsymbol{\tau}_{\mathrm{CT}}$	Generalized force including Coriolis, gravity, and friction terms for computed-torque control.
$\boldsymbol{u}'$	Auxuliary control variable from the computed-torque control.
$\boldsymbol{K}_p$	Positive-definite position gain matrix from the computed-torque control.
$\boldsymbol{K}_v$	Positive-definite velocity gain matrix from the computed-torque control.
$\boldsymbol{q}_0(t)$	Reference configuration trajectory.
$\dot{\boldsymbol{q}}_0(t)$	Reference velocity trajectory.
$\boldsymbol{e}(t)$	Configuration error.
$\boldsymbol{\theta}$	Vector of $d_{\mathcal{C}}$ independent coordinates of $\boldsymbol{q}$.
$\boldsymbol{m}$	Regular parametrization of $\boldsymbol{q}$, such that $\boldsymbol{q} = \boldsymbol{m}(\boldsymbol{\theta})$.
$\boldsymbol{q}_i$	Subset of independent coordinates from $\boldsymbol{q}$.
$\boldsymbol{q}_d$	Subset of dependent coordinates from $\boldsymbol{q}$.
$\boldsymbol{Q}_i$	Matrix selecting the independent coordinates $\boldsymbol{q}_i$.
$\boldsymbol{Q}_d$	Matrix selecting the dependent coordinates $\boldsymbol{q}_d$.